Activation Steering for Chain-of-Thought Compression

Seyedarmin Azizi[*]
University of Southern California, USA
seyedarm@usc.edu

Erfan Baghaei Potraghloo
University of Southern California, USA
baghaeip@usc.edu

Souvik Kundu
Intel AI, USA
souvikk.kundu@intel.com

Massoud Pedram
University of Southern California, USA
pedram@usc.edu

Abstract

Large language models (LLMs) excel at complex reasoning when they include intermediate steps, known as *chains of thought* (CoTs). However, these rationales are often overly verbose, even for simple problems, leading to wasted context, increased latency, and higher energy consumption. We observe that verbose CoTs and concise CoTs occupy distinct regions in the model's residual-stream activation space. By extracting and injecting a *steering vector* to transition between these regions, we can reliably shift generation toward more concise reasoning, effectively compressing CoTs without retraining. We formalize this approach as **Activation-Steered Compression (ASC)**, an inference-time technique that shortens reasoning traces by directly modifying hidden representations. In addition, we provide a theoretical analysis of the impact of ASC on the output distribution, derived from a closed-form KL-divergence-bounded constraint to regulate the steering strength. Using only 50 paired verbose and concise examples, ASC achieves up to **67.43%** reduction in CoT length on MATH500 and GSM8K datasets, while maintaining accuracy across 7B, 8B, and 32B parameter models. As a training-free method, ASC introduces negligible runtime overhead and, on MATH500, delivers an average **2.73×** speedup in end-to-end reasoning wall clock time on an 8B model. The code is available at https://github.com/ArminAzizi98/ASC.

1 Introduction

Explicit reasoning traces, commonly known as chains of thought (CoTs), significantly enhance the performance of LLMs on multi-step tasks such as mathematical problem solving, logical inference, and program synthesis (Wei et al., 2022b; Cobbe et al., 2021; Wang et al., 2023b). However, this advantage often comes with the drawback of generating unnecessarily lengthy and verbose rationales (Chen et al., 2025c; Xu et al., 2025). This verbosity not only increases computational costs by producing more tokens and consuming additional energy, but also risks impairing performance through *overthinking* (Chen et al., 2025c; Wang et al., 2025).

In this paper, we ask: *Can we compress chains of thought* ***without retraining****, by manipulating the model's hidden representations at inference time?* We answer the question affirmatively with **Activation-Steered Compression (ASC)**. Our key observation is that internal representations of verbose, natural-language CoTs and their concise, math-centric counterparts occupy distinct regions in the model activation space. To evaluate this hypothesis, we sample questions from the **MATH500** (Hendrycks et al., 2021a) benchmark and use two open weight reasoning models: DeepSeek-Distill-Qwen-7B and DeepSeek-Distill-LLaMA-8B. For each sample, we generate two

[*]Corresponding author

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Efficient Reasoning.

variants of the CoT: (1) a verbose reasoning chain produced by the model itself under standard prompting and (2) a concise reasoning produced by GPT-4o prompted to minimize natural language verbosity and maximize math-centric reasoning. We feed each input independently into the model and extract residual stream activations, that is, the outputs of a transformer block, in a predetermined layer (e.g., layer 21 in our experiments). A two-dimensional t-SNE projection (Van der Maaten & Hinton, 2008) of these activations, shown in Figure 1, reveals a clear separation between the two reasoning styles. This separation motivates the construction of a *steering vector*, which is a direction in the activation space that shifts the model's reasoning towards the concise response generation mode. By extracting this vector from a small calibration set and injecting it during generation, we guide the model to focus on essential steps, reducing verbosity while preserving accuracy. An example of such a pair of responses is shown in Fig. 2.

Why Activation Steering for CoT Compression? It is *lightweight*, requires only the addition of a single vector during inference, and directly reshapes hidden representations to enable on-the-fly compression. Because it does not involve updating the model parameters, this method is *deployment-agnostic*. Moreover, it is orthogonal and compatible with other categories of CoT compression.

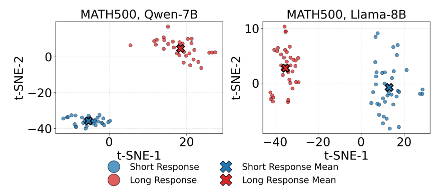

Figure 1: t-SNE of residual stream representations for long (verbose) and short (concise) CoTs

Steering involves not only the direction of modifications in the activation space but also determining the appropriate scale of the steering vector. If the scale is too small, the intervention has little effect; If it is too large, the model's output distribution can change unpredictably or even collapse. Previous approaches typically choose this scale heuristically, often by grid search or visual inspection. In contrast, we introduce a principled method for calibrating the steering strength by explicitly bounding the KL divergence between the original and steered output distributions.

Figure 2: A representative pair of verbose vs. concise CoTs used for generating the steering vector.

Problem

Question: Let $f(x) = -3x^2 + x - 4$, $g(x) = -5x^2 + 3x - 8$, and $h(x) = 5x^2 + 5x + 1$. Express $f(x) + g(x) + h(x)$ as a single polynomial, with the terms in order by decreasing degree. Let's think step by step.

Verbose Chain of Thought (603 tokens)

```
First, I need to add the three functions f(x), g(x), and h(x) together.  To do this, I should
start by writing down each function separately.\n\nSo, f(x) is -3x^2 + x - 4, g(x) is -5x^2 + 3x
- 8, and h(x) is 5x^2 + 5x + 1.\n\nNext, I'll add them term by term.  That means I'll combine the
coefficients of the x^2 terms, the x terms, and the constant terms separately.\n\nLet's start
with the x^2 terms.  For f(x), the coefficient is -3, for g(x) it's -5, and for h(x) it's 5.
Adding these together:  -3 + (-5) + 5.\n\nWait, that's -3 -5 +5.  Hmm, that simplifies to (-3
-5) +5, which is -8 +5, resulting in -3.\n\nOkay, so the x^2 term in the sum is -3x^2.\n\nNow,
moving on to the x terms.  For f(x), the coefficient is 1, for g(x) it's 3, and for h(x) it's
5.  Adding these:  1 + 3 +5.\n\nThat's straightforward:  1+3 is 4, plus 5 is 9.  So the x term
is 9x.\n\nFinally, the constant terms.  For f(x), it's -4, for g(x) it's -8, and for h(x) it's 1.
Adding these:  -4 + (-8) +1.\n\nThat's -4 -8 +1, which is (-12) +1, resulting in -11.\n\nPutting
it all together, the sum is -3x^2 +9x -11.\n\nWait, let me double-check my calculations to make
sure I didn't make any mistakes.\n\nFor the x^2 terms:  -3 (from f) + (-5) (from g) +5 (from h) =
(-3 -5) +5 = -8 +5 = -3.  That seems correct.\n\nFor the x terms:  1 (from f) +3 (from g) +5 (from
h) = 1+3+5=9.  That's correct.\n\nFor the constants:  -4 (from f) + (-8) (from g) +1 (from h) =
(-4 -8) +1 = -12 +1 = -11.  That's correct.\n\nSo, combining these, the final polynomial is -3x^2
+9x -11.\n\nI think that's all.  I don't see any mistakes in the addition.\n\n**Final Answer**\nThe
sum of the polynomials is \boxed{-3x^2 + 9x - 11}.\n</think>
```

Concise Chain of Thought (251 tokens)

```
\[\n f(x) + g(x) + h(x) = (-3x^2 + x - 4) + (-5x^2 + 3x - 8) + (5x^2 + 5x + 1)\]\n\nCombine like terms:
\n\n1. **Quadratic terms**:\n \[-3x^2 - 5x^2 + 5x^2 = (-3 - 5 + 5)x^2 = -3x^2 \]\n\n2. **Linear terms
**:\n   \[ x + 3x + 5x = (1 + 3 + 5)x = 9x \]\n\n3. **Constant terms**:\n \[-4 - 8 + 1 = -11 \]\n\n
Combine results:\n\[f(x) + g(x) + h(x) = -3x^2 + 9x - 11 \]\n\nFinal answer:\n\[\boxed{-3x^2 + 9x - 11}
\]
```

2 Related Work

Previous work tackles the CoT efficiency gap primarily through methods that *require additional training*: knowledge distillation schemes that learn concise rationales (Shen et al., 2025), latent token approaches that embed reasoning in compact vectors (Li et al., 2024), token-level pruning with supervised objectives such as TokenSkip (Xia et al., 2025), reinforcement-learning-based trajectory shortening exemplified by ThinkPrune (Hou et al., 2025), and latent-reasoning optimization frameworks that fine-tune internal deliberation steps (Chen et al., 2025a). While effective, these techniques incur considerable computational cost or architectural modifications. In contrast, we propose a *training-free, inference-time* method that compresses CoTs by directly manipulating hidden representations, retaining the accuracy benefits of reasoning traces without the overhead of retraining.

Chain of drafts (CoD) (Xu et al., 2025) and the approach of (Stolfo et al., 2024) reduce verbosity by embedding explicit length constraints in the prompt. CoD instructs the model to "think step by step" but keep the each draft to at most five words, whereas (Stolfo et al., 2024) limits the final answer to a user-specified number of sentences to create inference-time interventions. Although such heuristics can shorten outputs, they assume that the model will faithfully obey length directives, a behavior that recent studies show is unreliable for reasoning-oriented LLMs (Fu et al., 2025).

The closest work to ours is SEAL (Chen et al., 2025b), which constructs its steering vector by manually labeling the thought segments as *execution*, *reflection*, or *transition*, and then damping the latter two segment types. In contrast, (i) we learn a single *verbosity axis* from paired VERBOSE–vs.–CONCISE CoTs without any manual labels, (ii) rely solely on off-the-shelf prompts to generate training pairs, and (iii) obtain a domain-agnostic vector that generalizes across reasoning tasks. Therefore, our method provides a taxonomy-free, training-free complement to SEAL's category-based calibration.

3 Background

We place ASC at the intersection of research on CoT prompting and representation engineering.

Chain-of-thought (CoT) prompting improves multi-step reasoning by encouraging language models to articulate intermediate steps, often using signals such as "Let's think step by step" (Wei et al., 2022a). Several enhancements have refined this approach: **self-consistency**(Wang et al., 2023a) samples multiple rationales and selects the response supported by the majority; **tree-of-thought** (Yao et al., 2023) performs look-ahead search across branching reasoning paths; and **program-of-thought** (Chen et al., 2023) converts natural language reasoning into executable code. Although effective, these methods often increase the output length significantly. A recent study (Chen et al., 2025c) showed that o1-style reasoning models frequently produce excessively long CoTs due to redundant computations and unnecessary self-verification. We term these inefficiencies ***verbosity***, which we aim to address through inference-time activation-level intervention.

Activation Steering and Representation Engineering Linear activation editing has emerged as a lightweight alternative to fine-tuning. ActAdd demonstrates that adding a direction corresponding to "<|positive sentiment|>" can change the tone of the output (Turner et al., 2023). Reference(Burns et al., 2024) formalizes the approach as *representation engineering*, defining vectors as basis elements in a controllable subspace. This idea has now found use cases in style transfer (Konen et al., 2024), factual correction (Meng et al., 2022), and gender debiasing (Kong et al., 2024). However, existing work has yet to investigate the correlation between activation steering and CoT efficiency.

4 Methodology

Motivated by the goal of improving CoT efficiency through manipulation of the model's activation space, we introduce **Activation-Steered Compression (ASC)**—a method that shifts the model's hidden representations toward the subspace associated with concise, math-centric chains of thought. The method is summarized in Figure 3. First, we randomly sample 50 calibration samples from target dataset (in our case we have focused on **MATH500** (Hendrycks et al., 2021a) and from **GSM8K** (Cobbe et al., 2021)). For each question q_i in the calibration set, we obtain I) **Verbose CoT** l_i – generated by the target model with standard CoT prompting (Wei et al., 2022b). II) **Concise CoT** s_i – produced by GPT-4o instructed to use concise math-centric reasoning with minimal English. We denote the output of the transformer block in layer ℓ as *residual stream* of layer ℓ, and use h^ℓ to refer to it. Formally, h^ℓ is a matrix of shape $T \times d$, where T is the number of tokens in the input sequence

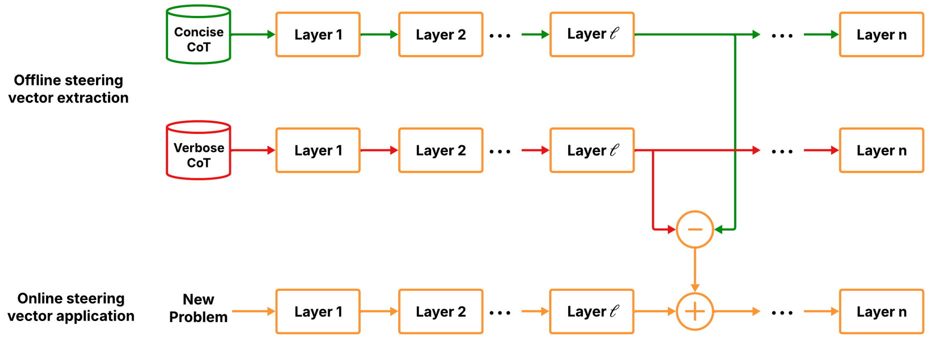

Figure 3: Steering vector extraction and application using pairs of concise and verbose CoTs.

and d is the hidden dimension of the model. With a slight abuse of notation, we write $h^\ell(s)$ to denote the residual stream in layer ℓ when a string s is fed into the model.

Following this notation, we feed the concatenated input `[question + CoT]` into the target model and extract the residual-stream activations corresponding to the final token in the input sequence. Specifically, we obtain $h^\ell(q_i \oplus l_i)[-1]$[2] and $h^\ell(q_i \oplus s_i)[-1]$ at a selected layer ℓ, corresponding to the verbose and concise CoTs, respectively. The steering vector is then computed as the average difference between these final token activations, that is, the activations associated with the token `EOS`, in all examples N.

$$v^\ell = \frac{1}{N}\sum_{i=1}^{N}\left(h^\ell(q_i \oplus s_i)[-1] - h^\ell(q_i \oplus l_i)[-1]\right).$$

v^ℓ is the desired steering direction for shifting the long and verbose CoT toward a more compact CoT. At inference time, given a new question and the i-th generated token x_i, we modify the residual stream by injecting the steering vector v^ℓ into layer ℓ during each decoding step, until an end-of-sequence delimiter is emitted. Specifically, for each decoding step i, we apply:

$$h^\ell(x_i) \leftarrow h^\ell(x_i) + \gamma v^\ell \quad \forall\, i \in [1, \texttt{decoding\_steps}]$$

Here, γ is a hyperparameter that controls the injection strength of the steering vector. If γ is too large, it can significantly distort the residual stream distribution, leading to degenerate or incoherent output. In contrast, if γ is too small, the steering effect becomes negligible. In what follows, we analyze the effect of the scaling parameter γ on the model output distribution from a theoretical perspective. We derive a safe upper bound on γ that guarantees that the output divergence remains within a user-specified threshold. For simplicity, we now drop the layer index ℓ throughout the analysis. We steer hidden activations by adding a direction $\mathbf{v}$ at layer ℓ, and choose the scale γ so that the resulting output distribution remains close to the unsteered model. Formally, letting $\mathbf{z}$ and $\tilde{\mathbf{z}}$ denote the pre-softmax logits before and after steering, we constrain the forward KL divergence:

$$\mathrm{KL}\big(\mathrm{softmax}(\mathbf{z}) \,\|\, \mathrm{softmax}(\tilde{\mathbf{z}})\big) \;\le\; \varepsilon,$$

where ε is a user-specified divergence budget (we use $\varepsilon = 10^{-3}$ in practice). The full derivation, deferred to Appendix A.1, decomposes the logit shift into a linear component $\gamma\mathbf{W}\mathbf{v}$ and a curvature-dependent remainder, where $\mathbf{W}$ is the Jacobian $\mathbf{J}(.)$ of the logit map with respect to the activations of layer ℓ. Under a mild smoothness condition with constant L as the upper bound of *directional curvature*, we derive a provable upper bound of KL that is quadratic, cubic, and quartic in γ. Specifically, defining

$$a := \left\|\mathbf{W}\mathbf{v}\right\|_2, \qquad L := \sup_{t\in[0,\gamma]} \frac{\|\mathbf{J}(\mathbf{h}+t\mathbf{v}) - \mathbf{J}(\mathbf{h})\|_2}{t},$$

we obtain a closed-form scale $\gamma_{\max}$ that ensures $\mathrm{KL} \le \varepsilon$. The expression includes a curvature-aware safety factor:

$$\gamma_{\max} = \max\left\{0, \left(1 - \tfrac{L\gamma_{\text{raw}}}{4a}\right)\gamma_{\text{raw}}\right\},$$

[2] $\oplus$ is the string concatenation operator.

where $\gamma_{\text{raw}} = (a/L) \cdot x$ and x is determined by solving the dimensionless cubic equation $x^3 + x^2 - 4\varepsilon L^2/a^4 = 0$. All proofs, derivations, and bounds appear in Appendix A.1. We adopt this calibrated $\gamma_{\max}$ in all experiments to control distributional shift while preserving the intended compression .

5 Experiments

This section presents our experimental results demonstrating that ASC effectively reduces the length of CoT reasoning while maintaining or improving task performance. We begin by describing our experimental setup in section 5.1, followed by the main results in section 5.2.

5.1 Experimental Setup

Models, Datasets, and Baselines. We evaluate ASC on several recent open-source reasoning models: DeepSeek-R1-Distill-LLaMA-8B (DeepSeek-AI, 2025b), DeepSeek-R1-Distill-Qwen-7B (DeepSeek-AI, 2025a), and QwQ-32B (Team, 2025). The evaluation is performed on multiple reasoning benchmarks, including MATH-500 (Hendrycks et al., 2021b) and GSM8K (Cobbe et al., 2021). As baselines, we compare ASC against vanilla CoT prompting (no steering), CoD (Xu et al., 2025), DEER (Yang et al., 2025), TCC (Muennighoff et al., 2025), and SEAL (Chen et al., 2025b), a recent method for compressed reasoning that uses steering vectors.

5.2 Main Results

Table 1 presents the performance of ASC compared to baseline CoT compression techniques. On the DeepSeek-R1-Distill-LLaMA-8B model, ASC reduces CoT length by up to **61.2%** without any loss in accuracy, outperforming prior methods in compression effectiveness. On the same model and the GSM8K dataset, ASC achieves a compression rate of **67.43%**, while also slightly improving answer accuracy by **0.2%**, matching or exceeding the performance of the vanilla CoT baseline. On MATH500, ASC achieves a **33.8%** reduction in CoT length, again outperforming all baselines. On the larger QwQ-32B model, ASC compresses CoTs by **50.7%** and **45.7%** on MATH500 and GSM8K, respectively. Notably, on MATH500, it yields a **0.4%** accuracy improvement over the vanilla CoT. Qualitative results on the CoT compression is provided in Appendix C.

Since one of the primary goals of CoT compression is to reduce end-to-end latency, we measure the average generation time for three models on the MATH500 dataset. Latency is measured on an NVIDIA A6000 GPU. We then compute and report the inverse latency (i.e., generation speed) for three decoding strategies: standard CoT, Chain-of-Drafts (CoD), and our proposed ASC, as shown in Figure 4. The results indicate that ASC improves the generation speed of CoT-based reasoning by up to 2.73×, with no loss in answer accuracy.

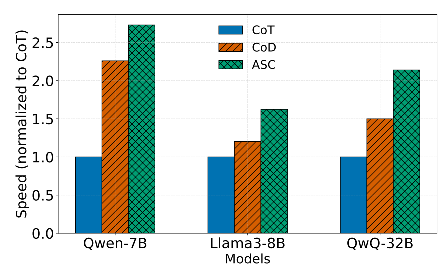

Figure 4: Speed comparison of CoT, CoD, and ASC on MATH500 dataset.

Table 1: Performance comparison of CoT, TCC, DEER, CoD, SEAL and ASC on reasoning tasks.

Model	Method	MATH500		GSM8k	
		Acc. (%) ↑	Tokens ↓	Acc. (%) ↑	Tokens ↓
Deepseek-R1-Distill-Qwen-7B	CoT	88.8	3984	88.6	1080
	TCC	89.2	3864	88.0	892
	DEER	**89.8**	2143	**90.6**	917
	SEAL	89.4	2661	88.4	811
	CoD	88.2	1852	87.9	550
	ASC	89.0	**1543**	88.6	**536**
Deepseek-R1-Distill-LLaMA-8B	CoT	89.2	3554	89.1	2610
	DEER	89.2	2830	89.3	2124
	CoD	88.8	3028	89.1	914
	ASC	**89.2**	**2353**	**89.3**	**850**
QwQ-32B	CoT	93.8	4508	**96.5**	1530
	TCC	94.4	4315	95.8	1348
	DEER	**94.6**	3316	96.3	977
	CoD	93.8	3400	96.2	1116
	ASC	94.2	**2222**	96.4	**830**

6 Discussion and Ablations

Cross-Task Generalization.
To investigate whether CoT verbosity is consistently reflected in the model's representation space, we examine the alignment of ASC steering vectors extracted from different reasoning tasks. Specifically, we analyze whether steering vectors derived from one dataset generalize to another. We conduct this study using the DeepSeek-R1-Distill-Qwen-7B model and two benchmarks: GSM8K and MATH500. Following the ASC methodology, we independently compute steering vectors for each dataset using 50 paired examples. We then assess the cosine similarity between the two vectors to quantify their alignment. In addition, we evaluate cross-task generalization by applying each dataset's steering vector to compress CoTs in the other dataset, measuring both length reduction and accuracy retention. The results are presented in Table 2. First, the cosine similarity between the two steering vectors is **0.92**, indicating strong alignment in the vectors from verbose to concise CoTs in MATH500 and GSM8K. Second, the performance of cross-dataset steering matches closely that of in-dataset vectors. Although there is a slight drop in accuracy and a slight increase in token count, ASC with cross-dataset steering still outperforms the vanilla CoT baseline (Table 1). These findings suggest that verbosity reduction occupies a *largely shared* latent direction across reasoning tasks, supporting our initial hypothesis that CoT efficiency can generally be attributed to the latent representations of the model.

Dataset	Steering Vector Source	Accuracy (%)	CoT Tokens
MATH500	MATH500 (in-dataset)	89.0	1543
	GSM8K (cross-dataset)	88.8	1631
GSM8K	GSM8K (in-dataset)	88.6	536
	MATH500 (cross-dataset)	88.4	611

Table 2: Performance of ASC on MATH500 and GSM8K using dataset-specific vs. cross-dataset steering vectors. The model used is DeepSeek-Distill-Qwen-7B.

Effect of Steering Strength γ. The steering strength γ is critical in ASC, as it directly influences both the degree of CoT compression and the quality of the generated output. To analyze its effect, we use the DeepSeek-R1-Distill-Qwen-7B model on the MATH500 dataset and perform a sweep over a range of γ values. The sweep begins at $\gamma = 0$ (no steering) and gradually increases until the steering induces noticeable compression along with a significant drop in answer accuracy $\gamma = 0.5$. The results are shown in Figure 5, highlighting the trade-off between CoT compression and answer accuracy as the steering strength γ increases. For small values of γ, increasing the strength yields substantial reductions in CoT length with minimal impact on accuracy. However, beyond a certain point, further increases in γ lead to significant accuracy degradation despite continued compression. Notably, the value of γ selected by ASC—computed via the KL-divergence–constrained scaling—closely aligns with the empirical breakpoint where performance begins to degrade.

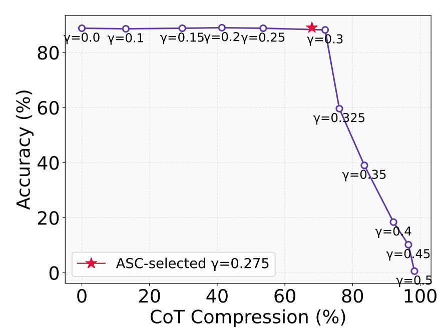

Figure 5: Effect of steering strength γ on CoT compression and answer accuracy for the DeepSeek-R1-Distill-Qwen-7B model on the MATH500 dataset.

7 Conclusion

We introduce Activation-Steered Compression (ASC), a training-free method for reducing the verbosity of Chain-of-Thought (CoT) reasoning in large language models by manipulating internal representations at inference time. By leveraging steering vectors derived from paired verbose and concise rationales, ASC effectively compresses CoTs without sacrificing accuracy. ASC complements existing CoT compression techniques and requires no retraining, and overall advances the efficiency and practicality of LLM-based reasoning.

References

Sam Burns et al. An introduction to representation engineering: Activation steering. *Alignment Forum*, 2024.

Haolin Chen, Yihao Feng, Zuxin Liu, Weiran Yao, Akshara Prabhakar, Shelby Heinecke, Ricky Ho, Phil L. Mui, Silvio Savarese, Caiming Xiong, and Huan Wang. Language models are hidden reasoners: Unlocking latent reasoning capabilities via self-rewarding. In *International Conference on Learning Representations (ICLR), under review*, 2025a. OpenReview ID 4Po8d9GAfQ.

Runjin Chen, Zhenyu Zhang, Junyuan Hong, Souvik Kundu, and Zhangyang Wang. Seal: Steerable reasoning calibration of large language models for free. *arXiv preprint arXiv:2504.07986*, 2025b.

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. Do not think that much for 2+3=? on the overthinking of o1-like llms, 2025c. URL `https://arxiv.org/abs/2412.21187`.

Zhuosheng Chen, Aston Zhang, Mu Li, and Alex Smola. Program-of-thought prompting: Efficient reasoning with small language models. *arXiv preprint arXiv:2305.10601*, 2023.

Karl Cobbe et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

DeepSeek-AI. Deepseek-r1-distill-qwen-7b. `https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Qwen-7B`, 2025a.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025b. URL `https://arxiv.org/abs/2501.12948`.

Tingchen Fu, Jiawei Gu, Yafu Li, Xiaoye Qu, and Yu Cheng. Scaling reasoning, losing control: Evaluating instruction following in large reasoning models. *arXiv preprint arXiv:2505.14810*, 2025.

Dan Hendrycks, Steven Basart, Nicholas Carlini, Jacob Steinhardt, and Dawn Song. Measuring mathematical problem solving with the math dataset. In *International Conference on Machine Learning (ICML)*, 2021a.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *NeurIPS Datasets and Benchmarks Track*, 2021b.

Bairu Hou, Yang Zhang, Jiabao Ji, Yujian Liu, Kaizhi Qian, Jacob Andreas, and Shiyu Chang. Thinkprune: Pruning long chain-of-thought of llms via reinforcement learning. *arXiv preprint arXiv:2504.01296*, 2025.

Kai Konen, Sophie Jentzsch, Diaoulé Diallo, Peer Schütt, Oliver Bensch, Roxanne El Baff, Dominik Opitz, and Tobias Hecking. Style vectors for steering generative large language model. *arXiv preprint arXiv:2402.01618*, 2024.

Lingkai Kong, Haorui Wang, Wenhao Mu, Yuanqi Du, Yuchen Zhuang, Yifei Zhou, Yue Song, Rongzhi Zhang, Kai Wang, and Chao Zhang. Aligning large language models with representation editing: A control perspective. *Advances in Neural Information Processing Systems*, 37:37356–37384, 2024.

Yuchen Li et al. Uncovering latent chain of thought vectors in language models. *arXiv preprint arXiv:2409.14026*, 2024.

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

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*, 2025.

Zhenyi Shen, Hanqi Yan, Linhai Zhang, Zhanghao Hu, Yali Du, and Yulan He. Codi: Compressing chain-of-thought into continuous space via self-distillation, 2025.

Alessandro Stolfo, Vidhisha Balachandran, Safoora Yousefi, Eric Horvitz, and Besmira Nushi. Improving instruction-following in language models through activation steering. *arXiv preprint arXiv:2410.12877*, 2024.

Alibaba Qwen Team. Qwq-32b: A 32b reasoning model from the qwen series. `https://huggingface.co/Qwen/QwQ-32B`, 2025. Apache2.0 licensed, open-weight; competitive reasoning performance.

Alexander Matt Turner, Lisa Thiergart, Gavin Leech, David Udell, Juan José Vázquez, Ulisse Mini, and Monte MacDiarmid. Steering language models with activation engineering. *OpenReview*, 2023. URL: `https://openreview.net/forum?id=2XBPdPIcFK`.

Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

Xuezhi Wang, Jason Wei, Jingshu Liu, Dale Schuurmans, Denny Zhou, and Quoc Le. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2023a.

Xuezhi Wang et al. Self-consistency improves chain of thought reasoning in language models. *ICLR*, 2023b.

Yue Wang, Qiuzhi Liu, Jiahao Xu, Tian Liang, Xingyu Chen, Zhiwei He, Linfeng Song, Dian Yu, Juntao Li, Zhuosheng Zhang, et al. Thoughts are all over the place: On the underthinking of o1-like llms. *arXiv preprint arXiv:2501.18585*, 2025.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Quoc V. Le, and Ed Chi. Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022a.

Jason Wei et al. Chain-of-thought prompting elicits reasoning in large language models. *NeurIPS*, 2022b.

Heming Xia, Yongqi Li, Chak Tou Leong, Wenjie Wang, and Wenjie Li. Tokenskip: Controllable chain-of-thought compression in llms. *arXiv preprint arXiv:2502.12067*, 2025.

Silei Xu, Wenhao Xie, Lingxiao Zhao, and Pengcheng He. Chain of draft: Thinking faster by writing less, 2025. URL `https://arxiv.org/abs/2502.18600`.

Chenxu Yang, Qingyi Si, Yongjie Duan, Zheliang Zhu, Chenyu Zhu, Qiaowei Li, Zheng Lin, Li Cao, and Weiping Wang. Dynamic early exit in reasoning models. *arXiv preprint arXiv:2504.15895*, 2025.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate reasoning via chain of thought. *arXiv preprint arXiv:2305.10601*, 2023.

A KL-Constrained Scaling of Steering Vectors

A.1 Bounding the Distributional Shift of Additive Steering

We study the output–distribution shift incurred when an *additive steering* update is applied to the hidden state at layer ℓ of a language model. For an activation vector $\mathbf{h} \in \mathbb{R}^d$ we form

$$\tilde{\mathbf{h}} := \mathbf{h} + \gamma \mathbf{v}, \qquad \|\mathbf{v}\|_2 = 1,$$

to analyze how large the Kullback–Leibler (KL) divergence between the pre- and post-steering output distributions can become.

Throughout, let $\mathcal{F}_{l\to\text{logit}} : \mathbb{R}^d \to \mathbb{R}^m$ denote the sub-network that maps layer-ℓ activations to the pre-softmax *logits*. All vector norms $\|\cdot\|_2$ and operator-2 norms are Euclidean; they coincide when the argument is a vector.

Notation for higher-order derivatives. The Jacobian of $\mathcal{F}_{l\to\text{logit}}$ at $\mathbf{h}$ is the matrix

$$\mathbf{J}(\mathbf{h}) := \nabla_{\mathbf{h}} \mathcal{F}_{l\to\text{logit}}(\mathbf{h}) \in \mathbb{R}^{m\times d},$$

whose j-th row is $(\nabla_{\mathbf{h}} F_j(\mathbf{h}))^\top$. The Hessian of a scalar component is the usual matrix of second partials. For a unit vector $\mathbf{a}$ we abbreviate *directional* Hessian Evaluation

$$\nabla^2_{\mathbf{h}} \mathcal{F}_{l\to\text{logit}}(\mathbf{h})[\mathbf{a},\mathbf{a}] := \big(\nabla^2_{\mathbf{h}} F_1(\mathbf{h})[\mathbf{a},\mathbf{a}], \ldots, \nabla^2_{\mathbf{h}} F_m(\mathbf{h})[\mathbf{a},\mathbf{a}]\big)^\top \in \mathbb{R}^m.$$

A.1.1 A smoothness assumption

Assumption 1 *There exists a constant $L > 0$ such that for every unit direction $\mathbf{v}$ and every $t \in [0,\gamma]$*

$$\big\|\mathbf{J}(\mathbf{h} + t\mathbf{v}) - \mathbf{J}(\mathbf{h})\big\|_2 \le L\, t.$$

Implication. Assumption 1 is stronger than merely requiring bounded second derivatives. In fact, according to the mean value theorem for vector-valued Lipschitz maps, $\mathbf{J}$ is differentiable almost everywhere and its derivative (the third-order tensor of second partials) has the operator norm at most L. Contracting this tensor twice with the same unit vector $\mathbf{v}$ yields

$$\big\|\nabla^2_{\mathbf{h}} \mathcal{F}_{l\to\text{logit}}(\mathbf{h} + \tau\mathbf{v})[\mathbf{v},\mathbf{v}]\big\|_2 \le L, \qquad \forall \tau \in [0,\gamma], \tag{1}$$

because $\|H[\mathbf{v},\mathbf{v}]\|_2 \le \|H\|_{\text{op}} \|\mathbf{v}\|_2^2 = \|H\|_{\text{op}}$. Thus Assumption 1 *implies*–though it is not equivalent to–a uniform bound on the directional Hessian.

A.1.2 Local linearization with a controlled remainder

Define

$$\mathbf{z} := \mathcal{F}_{l\to\text{logit}}(\mathbf{h}), \qquad \mathbf{W} := \mathbf{J}(\mathbf{h}) \in \mathbb{R}^{m\times d}.$$

By the fundamental theorem of calculus and Eq. (1), the steered logits decompose as

$$\tilde{\mathbf{z}} = \mathcal{F}_{l\to\text{logit}}(\mathbf{h} + \gamma\mathbf{v}) \tag{2}$$

$$= \mathbf{z} + \underbrace{\gamma \mathbf{W}\mathbf{v}}_{:=\boldsymbol{\delta}} + \underbrace{\int_0^\gamma (\gamma - s) \nabla^2_{\mathbf{h}} \mathcal{F}_{l\to\text{logit}}(\mathbf{h} + s\mathbf{v})[\mathbf{v},\mathbf{v}]\, ds}_{:=\mathbf{r}(\gamma)}. \tag{3}$$

The *linear component* is $\boldsymbol{\delta} = \gamma \mathbf{W}\mathbf{v}$, while the *remainder* obeys

$$\|\mathbf{r}(\gamma)\|_2 \le \tfrac{1}{2} L \gamma^2. \tag{4}$$

A.1.3 KL divergence as a Bregman divergence

Let $g(\mathbf{x}) = \log \sum_{i=1}^m e^{x_i}$ and denote

$$\mathbf{p} = \text{softmax}(\mathbf{z}), \qquad \tilde{\mathbf{p}} = \text{softmax}(\tilde{\mathbf{z}}).$$

For the log-partition function g, the Bregman divergence is

$$D_g(\tilde{\mathbf{z}}, \mathbf{z}) = g(\tilde{\mathbf{z}}) - g(\mathbf{z}) - \langle \nabla g(\mathbf{z}), \tilde{\mathbf{z}} - \mathbf{z}\rangle = \text{KL}(\mathbf{p}\|\tilde{\mathbf{p}}). \tag{5}$$

Thus, the classical *forward* KL direction appears.

A.1.4 Integral representation and spectral bound

Using the integral representation of a Bregman divergence for twice-differentiable convex g we obtain

$$\mathrm{KL}(\mathbf{p}\|\tilde{\mathbf{p}}) = \int_0^1 (1-t)(\tilde{\mathbf{z}}-\mathbf{z})^\top \nabla^2 g\big(\mathbf{z}+t(\tilde{\mathbf{z}}-\mathbf{z})\big)(\tilde{\mathbf{z}}-\mathbf{z})\,dt. \tag{6}$$

Because $\nabla^2 g(\mathbf{x})$ equals the Fisher information matrix $\mathbf{F}(\mathbf{x}) = \mathrm{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^\top$, whose largest eigenvalue never exceeds $\frac{1}{2}$, and the factor $(1-t)$ integrates to $\frac{1}{2}$, we have

$$\mathrm{KL}(\mathbf{p}\|\tilde{\mathbf{p}}) \le \tfrac{1}{4}\,\|\tilde{\mathbf{z}}-\mathbf{z}\|_2^2. \tag{7}$$

This constant $1/4$ is tight for our purposes.

A.1.5 Putting the pieces together

With $\tilde{\mathbf{z}}-\mathbf{z} = \boldsymbol{\delta} + \mathbf{r}(\gamma)$ and the triangle inequality,

$$\|\tilde{\mathbf{z}}-\mathbf{z}\|_2^2 \le \big(\|\boldsymbol{\delta}\|_2 + \|\mathbf{r}(\gamma)\|_2\big)^2 \tag{8}$$

$$\le \|\boldsymbol{\delta}\|_2^2 + 2\|\boldsymbol{\delta}\|_2\|\mathbf{r}(\gamma)\|_2 + \|\mathbf{r}(\gamma)\|_2^2. \tag{9}$$

Invoking (4) and $\|\boldsymbol{\delta}\|_2 = \gamma a$ with $a := \|\mathbf{W}\mathbf{v}\|_2$, we derive from (7) the **corrected steering bound**:

$$\boxed{\mathrm{KL}(\mathbf{p}\|\tilde{\mathbf{p}}) \le \tfrac{1}{4}\,\gamma^2 a^2 + \tfrac{1}{4}\,La\,\gamma^3 + \tfrac{1}{16}\,L^2\gamma^4} \tag{10}$$

Safe γ budget with a curvature safety factor. Fix a target divergence $\varepsilon > 0$. Ignoring the last term in (10) yields the cubic inequality

$$\tfrac{1}{4}a^2\gamma^2 + \tfrac{1}{4}La\,\gamma^3 \le \varepsilon.$$

Set $x := (L\gamma)/a$ (dimensionless) and $\beta := 4\varepsilon L^2/a^4$. The inequality becomes

$$x^3 + x^2 - \beta \le 0,$$

whose *unique positive* root solves

$$x^3 + x^2 - \beta = 0.$$

Writing the depressed cubic

$$(x+\tfrac{1}{3})^3 - \tfrac{1}{3}(x+\tfrac{1}{3}) + \big(\tfrac{2}{27}-\beta\big) = 0$$

and setting

$$p = -\tfrac{1}{3}, \qquad q = \tfrac{2}{27} - \beta, \qquad \Delta = \big(\tfrac{q}{2}\big)^2 + \big(\tfrac{p}{3}\big)^3,$$

the real Cardano root is

$$x = \sqrt[3]{-\tfrac{q}{2}+\sqrt{\Delta}} + \sqrt[3]{-\tfrac{q}{2}-\sqrt{\Delta}} - \tfrac{1}{3}. \tag{11}$$

Numerically, this expression is unambiguous if one takes the *real* branch of each cube root. Finally,

$$\boxed{\gamma_{\text{raw}} = \frac{a}{L}\,x.} \tag{12}$$

Degenerate direction $a = 0$. If $a = 0$ (the steering vector lies in the null-space of $\mathbf{W}$) the quadratic and cubic terms vanish; retaining the quartic term in (10) gives $\frac{L^2}{16}\gamma^4 \le \varepsilon$ and hence $\gamma \le (16\varepsilon)^{1/4}/\sqrt{L}$. We therefore set

$$\gamma_{\text{raw}} = \begin{cases} (a/L)\,x & \text{if } a > 0, \\ (16\varepsilon)^{1/4}/\sqrt{L} & \text{if } a = 0. \end{cases}$$

Curvature safety factor. Because the quartic term in (10) is strictly positive, γ_{raw} is slightly optimistic when $L\gamma$ is not negligible relative to a. We therefore define the final scale

$$\boxed{\gamma_{\max} = \max\left\{0, \left(1 - \tfrac{L\gamma_{\text{raw}}}{4a}\right)\gamma_{\text{raw}}\right\}} \tag{13}$$

with the convention $L\gamma_{\text{raw}}/a := 0$ when $a = 0$. One checks directly that substituting $\gamma = \gamma_{\max}$ into (10) yields $\text{KL} \leq \varepsilon$ whenever $x < 4$, which covers all practical regimes.

When $L \to 0$ the safety factor tends to 1 and Eq. (13) continuously reduces to the familiar local-linear scale $\gamma_{\max} = 2\sqrt{\varepsilon}/a$.

In practice, we estimate the two scale parameters α and L on the small calibration set (50 hidden states). For each hidden state, we evaluate one Jacobian–vector product with the chosen steering direction and record its Euclidean norm; the median of these norms is taken as our estimate of a. To obtain L, we compute a single Hessian vector product along the same direction at each calibration point, collect the resulting norms, take their 95^{th} percentile. Both JVP and HVP operations are one-line calls in modern autodiff frameworks, so the entire procedure runs in a few seconds even on large-scale models.

B Implementation Details.

For all experiments, we use the decoding hyperparameters `temperature` $= 0.7$, `top_p` $= 0.9$, and `repetition_penalty` $= 1.1$; all other settings follow the default configurations of the respective models. The evaluation datasets are accessed through the Hugging Face `datasets` library. Experiments are conducted on NVIDIA A6000 GPUs, using `PyTorch` version `2.5.1+cu124` and the `transformers` library version `4.50.1`. The hyperparameters related to steering, namely the steering strength γ and the layer index used to extract and apply the steering vector, are detailed in the appendix D.

C Qualitative Results

In this section, we present illustrative examples from the MATH500 dataset, comparing standard Chain-of-Thought (CoT) responses with those produced by ASC. Figures 6 and 7 showcase representative outputs from the QwQ-32B reasoning model. As noted in prior work (Wang et al., 2025; Chen et al., 2025c), vanilla CoT traces often exhibit excessive verbosity, frequent shifts in reasoning strategy, and redundant self-verification steps. In contrast, ASC responses are more focused, mathematically grounded, and free from distractive interjections such as *"Alternatively," "Wait,"* and other similar expressions. This reflects ASC's ability to steer the model toward concise, mathematically structured reasoning.

Figure 6: Qualitative Example for comparing ASC response against vanilla CoT response.

Problem

Question: The product of two consecutive positive even integers is 288. What is the greater of the two integers?

Vanilla CoT (1761 tokens)

Let's denote the two consecutive positive even integers as x and $x + 2$. According to the problem, their product is 288. Therefore, we can set up the equation:

$$x(x + 2) = 288$$

Expanding the left side gives:

$$x^2 + 2x = 288$$

To form a quadratic equation, subtract 288 from both sides:

$$x^2 + 2x - 288 = 0$$

We can solve this quadratic equation using the quadratic formula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Here, $a = 1$, $b = 2$, and $c = -288$. Plugging these values into the formula:

$$x = \frac{-2 \pm \sqrt{(2)^2 - 4(1)(-288)}}{2(1)}$$

Calculating the discriminant (D):

$$D = 4 + 1152 = 1156$$

Taking the square root of the discriminant:

$$\sqrt{1156} = 34$$

So the solutions for x are:

$$x = \frac{-2 + 34}{2} = \frac{32}{2} = 16$$
$$x = \frac{-2 - 34}{2} = \frac{-36}{2} = -18$$

Since we're looking for positive integers, we discard the negative solution. Thus, the smaller integer is 16, and the next consecutive even integer is 18. However, wait—that would make the larger number 18, but let me check my work because sometimes I might miscalculate.
Wait, if $x = 16$, then the next consecutive even integer is indeed $16 + 2 = 18$, so the pair is (16, 18). Their product is 16*18. Let me compute that: 16*10=160, 16*8=128; adding them gives 160+128=288. That checks out. So the greater integer is 18.
But hold on, maybe there was an error in setting up the equations?
Alternatively, perhaps I should try factoring the quadratic equation before resorting to the quadratic formula, just to verify.
Starting with $x^2 + 2x - 288 = 0$.
Looking for two numbers that multiply to -288 and add up to 2. Since their product is negative, one is positive and the other negative. Let's think about factors of 288.
Factors around $\sqrt{288}$, which is approximately 17. So possible pairs:
16 and 18: $16 \times 18 = 288$. But since it's -288, one has to be negative. If we take 18 and -16: $18 + (-16) = 2$. Yes! Exactly.
Therefore, the equation factors as:
$(x + 18)(x - 16) = 0$
Setting each factor equal to zero:
Either $x + 18 = 0 \Rightarrow x = -18$,
Or $x - 16 = 0 \Rightarrow x = 16$.
Same result as before. So the positive solution is 16, hence the two numbers are 16 and 18, so the greater is 18. Hmm. Wait a second, but when I first thought through, I got confused why the answer isn't higher, but 16 and 18 do multiply to 288.
Alternatively, could there be another pair? Let's see, perhaps starting at a different point.
Suppose someone takes the two numbers as n and n+2, such that their product is 288, so same as above.
Alternatively, maybe starting with the average? The square root of 288 is approximately 16.97. So between 16 and 17. Hence, the even integers near that would be 16 and 18. Which matches our previous conclusion.
Thus, the greater integer is 18. But hold on—I feel like maybe I missed something here, because often in these problems they expect maybe higher numbers? Let me confirm once again.
Compute 16 × 18:
16 × 10 = 160,
16 × 8 = 128,
Total: 160 + 128 = 288. Correct.
Alternatively, cross-verify using another method:
18 × 16: 18×10=180, 18×6=108 → total 180+108=288. Same result.
Alright, seems consistent. Therefore, the two numbers are 16 and 18, so the greater one is 18.
Wait, but wait—hold on. When I initially set up the problem, I called them x and x+2. Then x=16 is the first, x+2=18. That's correct. So the greater integer is 18. Is that the final answer?
Yes, unless there's a mistake in the setup. For example, if the question had said "consecutive even integers", which they are. Since 16 and 18 are two apart, so yes, consecutive even integers.
Alternatively, if they were consecutive integers (differing by 1), but the problem says even, so difference must be 2. So everything checks out. Hmm. Maybe overthinking, but better safe than sorry.
Another way: list some consecutive even integers and see where the product reaches 288.
Start from lower numbers:
10×12=120
12×14=168
14×16=224
16×18=288 → Bingo.
Yep, that's exactly it. So 16 and 18. Therefore, the greater is 18. So confident now that's the right answer. Initially, I thought maybe I made an arithmetic mistake, but all methods concur.
Final Answer $\boxed{18}$ You start by denoting the two consecutive positive even integers as x and $x + 2$. Given their product is 288, you set up the equation:

$$x(x + 2) = 288$$

Expanding and rearranging terms gives the quadratic equation:

$$x^2 + 2x - 288 = 0$$

Using the quadratic formula $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ with $a = 1$, $b = 2$, and $c = -288$:

$$x = \frac{-2 \pm \sqrt{4 + 1152}}{2} = \frac{-2 \pm 34}{2}$$

This yields two solutions:

$$x = \frac{32}{2} = 16 \quad \text{and} \quad x = \frac{-36}{2} = -18$$

Since we need positive integers, we discard $x = -18$. The smaller integer is 16, and the next consecutive even integer is 18. Verifying the product:

$$16 \times 18 = 288$$

Thus, the greater of the two integers is:

$$\boxed{18}$$

ASC generated CoT (788 Tokens)

The product of two consecutive positive even integers is 288. What is the greater of the two integers? Let's denote the smaller integer as x. Since they are consecutive even integers, the next one would be x + 2.
Their product is given by: x(x + 2) = 288
Expanding this equation: x² + 2x = 288
Rearranging into a quadratic equation: x² + 2x - 288 = 0
We can solve this using the quadratic formula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Here, $a = 1$, $b = 2$, $c = -288$.
Calculating discriminant first:

$$D = b^2 - 4ac = (2)^2 - 4 \cdot 1 \cdot (-288) = 4 + 1152 = 1156$$

$$\sqrt{D} = \sqrt{1156} = 34$$

So, x = [-2 ± 34]/2
Since we need positive integers, take the positive solution: x = (-2 + 34)/2 = 32/2 = 16
Therefore, the two integers are 16 and 18, so the greater one is 18. But wait, let me check that again. If I plug back in:
16 * 18 = 288, which matches the problem statement. So yes, correct.
Alternatively, maybe there was another way to approach it?
Another method could be factoring. Looking for two numbers that multiply to 288 with difference 2 since they're consecutive evens.
Factors of 288: Start from $\sqrt{288} \approx 17$, so around there.
Check nearby even numbers.
16 ×18 is indeed 288, so those must be them. Hence, same result. The larger number is 18.
Final Answer $\boxed{18}$ Let x be the smaller of the two consecutive positive even integers. Then the next consecutive even integer is $x + 2$. Their product is given by:

$$x(x + 2) = 288$$

Expanding and rearranging this equation into a standard quadratic form gives:

$$x^2 + 2x - 288 = 0$$

Using the quadratic formula $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$, where $a = 1$, $b = 2$, and $c = -288$:
First, calculate the discriminant:

$$b^2 - 4ac = 2^2 - 4(1)(-288) = 4 + 1152 = 1156$$

Taking the square root of the discriminant:

$$\sqrt{1156} = 34$$

Substituting back into the quadratic formula:

$$x = \frac{-2 \pm 34}{2}$$

Considering only the positive solution (since we are dealing with positive integers):

$$x = \frac{-2 + 34}{2} = \frac{32}{2} = 16$$

Thus, the smaller integer is 16, and the next consecutive even integer is 18. Verifying their product:

$$16 \times 18 = 288$$

Hence, the greater of the two integers is $\boxed{18}$.

Vanilla CoT (4283 Tokens)

To find the double sum $\sum_{j=1}^{\infty} \sum_{k=1}^{\infty} \frac{1}{(j+k)^3}$ in terms of p and q, where $p = \sum_{k=1}^{\infty} \frac{1}{k^2}$ and $q = \sum_{k=1}^{\infty} \frac{1}{k^3}$, we can proceed as follows:
First, note that the given double sum is over all pairs (j, k) with both indices starting from 1. Let's consider changing variables by letting $n = j + k$. For each fixed value of n, the number of pairs (j, k) such that $j + k = n$ is exactly $n - 1$, since j can range from 1 to $n - 1$ (with $k = n - j$). Therefore, we can rewrite the double sum as a single series involving these counts:

$$\sum_{j=1}^{\infty} \sum_{k=1}^{\infty} \frac{1}{(j+k)^3} = \sum_{n=2}^{\infty} \sum_{j=1}^{n-1} \frac{1}{n^3}.$$

Here, n starts at 2 because the smallest possible value of $j + k$ when j and k are both at least 1 is $1 + 1 = 2$.

Figure 7: Qualitative Example for comparing ASC response against vanilla CoT response.

Problem

Question: Define

$$p = \sum_{k=1}^{\infty} \frac{1}{k^2} \quad \text{and} \quad q = \sum_{k=1}^{\infty} \frac{1}{k^3}.$$

Find a way to write

$$\sum_{j=1}^{\infty} \sum_{k=1}^{\infty} \frac{1}{(j+k)^3}$$

in terms of p and q.

The inner sum $\sum_{j=1}^{n-1} \frac{1}{n^3}$ simplifies to $\frac{n-1}{n^3}$, since there are $n-1$ terms each equal to $\frac{1}{n^3}$. Thus, the entire expression becomes:

$$\sum_{n=2}^{\infty} \frac{n-1}{n^3}.$$

We can split this into two separate sums:

$$\sum_{n=2}^{\infty} \left(\frac{n}{n^3} - \frac{1}{n^3} \right) = \sum_{n=2}^{\infty} \frac{1}{n^2} - \sum_{n=2}^{\infty} \frac{1}{n^3}.$$

These sums can be rewritten in terms of p and q:
Recall that $p = \sum_{n=1}^{\infty} \frac{1}{n^2}$, so $\sum_{n=2}^{\infty} \frac{1}{n^2} = p - 1$.
Similarly, $q = \sum_{n=1}^{\infty} \frac{1}{n^3}$, so $\sum_{n=2}^{\infty} \frac{1}{n^3} = q - 1$.
Therefore, substituting back into our expression gives:

$$(p-1) - (q-1) = p - 1 - q + 1 = p - q.$$

Wait a second! But let me check this again. Wait, I might have made an error here.
Hold on: The original expression after splitting was:

$$\sum_{n=2}^{\infty} \frac{1}{n^2} - \sum_{n=2}^{\infty} \frac{1}{n^3}$$

Yes, but then $\sum_{n=2}^{\infty} \frac{1}{n^2} = p - 1/(1^2) = p - 1$,
and $\sum_{n=2}^{\infty} \frac{1}{n^3} = q - 1/1^3 = q - 1$,
so indeed their difference would be (p - 1) - (q - 1) = p - q,
So according to this calculation, the answer is p - q?
But wait, that seems too straightforward. However, let me verify if my initial steps were correct.
Starting from the beginning:
Original double sum:

$$\sum_{j,\, k \geq 1} \frac{1}{(j+k)^3}.$$

Change variable to $n = j + k$. Then for each $n \geq 2$, the terms correspond to j from 1 to $n-1$, so a total of $\frac{n-1}{n^3}$ per n. So total sum is

$$\sum_{n=2}^{\infty} \frac{n-1}{n^3}.$$

Then split into

$$\sum_{n=2}^{\infty} \left(\frac{1}{n^2} - \frac{1}{n^3} \right).$$

Hence,

$$\sum_{n=2}^{\infty} \frac{1}{n^2} = p - 1, \quad \text{and similarly} \quad \sum_{n=2}^{\infty} \frac{1}{n^3} = q - 1.$$

Subtract those to get

$$(p-1) - (q-1) = p - q.$$

Hmm, so the conclusion would be that the double sum equals $p - q$? That is,

$$\sum_{j,\, k \geq 1} \frac{1}{(j+k)^3} = p - q?$$

Alternatively, perhaps there is an error in splitting the numerator?
Wait: Let me see:

$$\frac{n-1}{n^3} = \frac{n}{n^3} - \frac{1}{n^3} = \frac{1}{n^2} - \frac{1}{n^3}.$$

Yes, that's correct.

Therefore, yes, the sum becomes

$$\sum_{n=2}^{\infty}\left(\frac{1}{n^2}-\frac{1}{n^3}\right),$$

which is indeed

$$\left(\sum_{n=2}^{\infty}\frac{1}{n^2}\right)-\left(\sum_{n=2}^{\infty}\frac{1}{n^3}\right),$$

which is

$$(p-1)-(q-1)=p-q.$$

So according to this, the answer should just be $p-q$.
But maybe I need to confirm numerically?
Let me test with partial sums.
Take $p=\frac{\pi^2}{6}\approx 1.6449$, $q=\zeta(3)\approx 1.2020569$, so $p-q\approx 0.4428$.
Now compute the double sum up to some finite N and see if it approximates this.
Suppose we take small terms.
Compute

$$\sum_{j=1}^{2}\sum_{k=1}^{2}\frac{1}{(j+k)^3}.$$

For $j=1,k=1$: $1/8$; $j=1,k=2$: $1/27$; $j=2,k=1$: $1/27$; $j=2,k=2$: $1/64$. Total is $1/8+2/27+1/64\approx 0.125+0.07407+0.015625\approx 0.2147$.
Adding more terms, say up to $j=3,k=3$:
Additional terms:
$j=1,k=3$: $1/64$; $j=3,k=1$: $1/64$;
$j=2,k=3$: $1/125$; $j=3,k=2$: $1/125$;
$j=3,k=3$: $1/216$.
Total added: $2\cdot\frac{1}{64}+2\cdot\frac{1}{125}+\frac{1}{216}\approx 0.03125+0.016+0.00463\approx 0.05185$. New total $\approx 0.2147+0.05185\approx 0.26655$.
Continuing to j=4, etc., but even so, it's going to converge slowly. Since the exact value is supposed to be about 0.4428..., so maybe my approach is correct? Hmm. Alternatively, perhaps my mistake is in the change of variables?
Wait another thought: When you do the substitution n = j + k, the summation is over all n from 2 to infinity, and for each n, the number of terms is (n -1). So the first term when n=2 has 1 term, n=3 has 2 terms, ..., n=m has m-1 terms. Therefore, the total sum is

$$\sum_{n=2}^{\infty}\frac{n-1}{n^3}.$$

Thus, that is indeed equivalent to

$$\sum_{n=1}^{\infty}\frac{n-1}{n^3}$$

starting from $n=2$, but perhaps written as

$$\sum_{n=1}^{\infty}\left(\frac{n-1}{n^3}\right)-(\text{term at } n=1).$$

At $n=1$, $\frac{n-1}{n^3}=0$, so no contribution. Hence, indeed

$$\sum_{n=2}^{\infty}\frac{n-1}{n^3}=\sum_{n=1}^{\infty}\frac{n-1}{n^3}.$$

But perhaps reindexing could help.
Alternatively, writing

$$\frac{n-1}{n^3}=\frac{1}{n^2}-\frac{1}{n^3},$$

so indeed the sum splits into

$$\left(\sum_{n=1}^{\infty}\frac{1}{n^2}-\frac{1}{1^2}\right)-\left(\sum_{n=1}^{\infty}\frac{1}{n^3}-\frac{1}{1^3}\right)=(p-1)-(q-1)=p-q.$$

Therefore, the final result is $p-q$.
Alternatively, perhaps the problem wants the answer expressed differently?
Wait, let me cross-verify using integral transforms or known formulas.
Alternatively, maybe express the double sum as an integral?
Note that

$$\frac{1}{(j+k)^3}=\int_0^1 x^{j+k-1}\,dx\cdot\frac{1}{2},$$

but not sure. Wait, recall that

$$\frac{1}{m^3}=\int_0^1\int_0^y\int_0^z dx\,dz\,dy.$$

Maybe triple integrals?
Alternatively, integrating over t from 0 to infinity?
Alternatively, use generating functions.
Alternatively, think of the double sum as:

$$\sum_{j,\,k\geq 1}\frac{1}{(j+k)^3}=\sum_{m=2}^{\infty}(\text{number of pairs } (j,k) \text{ with } j+k=m)\cdot\frac{1}{m^3}.$$

Which is exactly what we did before, leading to (p - q). So unless there's a miscalculation here, that must be correct.
Alternatively, maybe a different approach.
Consider interchanging the order of summations or using known multiple series results.
Alternatively, note that the sum is symmetric in j and k, so perhaps we can consider replacing j and k with other variables, but probably complicates things.
Alternatively, maybe express the double sum as:

$$\sum_{j=1}^{\infty}\sum_{k=1}^{\infty}\frac{1}{(j+k)^3}=\sum_{j=1}^{\infty}\left[\sum_{k=1}^{\infty}\frac{1}{(j+k)^3}\right].$$

Fix j, then the inner sum over k from 1 to ∞ of $\frac{1}{(j+k)^3}$ is

$$\sum_{m=j+1}^{\infty}\frac{1}{m^3}$$

where $m = j + k$. So that sum is equal to

$$q_j=\sum_{m=j+1}^{\infty}\frac{1}{m^3}=q-\sum_{m=1}^{j}\frac{1}{m^3}.$$

Therefore, substituting back, the original double sum becomes:

$$\sum_{j=1}^{\infty}\left[q-\sum_{m=1}^{j}\frac{1}{m^3}\right]=\sum_{j=1}^{\infty}q-\sum_{j=1}^{\infty}\sum_{m=1}^{j}\frac{1}{m^3}.$$

The first term is infinite because it's $\sum_j q$, clearly divergent, so that approach doesn't work because we need to handle convergence properly.
Wait, this suggests an error in reasoning. Wait, hold on — actually, the inner sum over k for each j is

$$\sum_{k=1}^{\infty}\frac{1}{(j+k)^3},$$

which converges for each j. The outer sum over j may also converge, but how does this reconcile with the previous approach?
Indeed, proceeding this way, let me see:
The double sum

$$S=\sum_{j=1}^{\infty}\left[\sum_{m=j+1}^{\infty}\frac{1}{m^3}\right]$$

This is the same as swapping the order of summations. Indeed, switching the order of summations:

$$S=\sum_{m=2}^{\infty}\sum_{j=1}^{m-1}\frac{1}{m^3}=\sum_{m=2}^{\infty}\frac{m-1}{m^3},$$

which matches the earlier result. So indeed, when swapping the order, it works out.
Whereas trying to express it as

$$\sum_{j=1}^{\infty}[q-\cdots]$$

led to confusion because the individual terms involve q, but they are being summed over j, leading to divergence unless handled properly.
Instead, recognizing that

$$\sum_{m=j+1}^{\infty}\frac{1}{m^3}=q-\sum_{m=1}^{j}\frac{1}{m^3}$$

allows us to write

$$S=\sum_{j=1}^{\infty}\left(q-\sum_{m=1}^{j}\frac{1}{m^3}\right).$$

However, this expression still requires careful handling. Let's proceed carefully:

$$S=\sum_{j=1}^{\infty}\left(q-\sum_{m=1}^{j}\frac{1}{m^3}\right)=q\sum_{j=1}^{\infty}1-\sum_{j=1}^{\infty}\sum_{m=1}^{j}\frac{1}{m^3}$$

But the first term diverges, while the second term also diverges, suggesting the subtraction cancels the divergence? This approach isn't helpful, so better to stick with the first method which gave a finite result. Therefore, likely the first approach is correct, giving $S = p - q$.
Wait, but let me see numerical values:
If $p = \frac{\pi^2}{6} \approx 1.6449$ and $q \approx 1.2020569$, then $p - q \approx 0.4428$.
Calculating a few more terms manually:
Compute partial sums up to j and k <= 4:
Compute all terms with j from 1-4 and k from 1-4, excluding higher terms beyond 4.
The terms where $j + k \leq 8$, but this might not capture much. Alternatively, let me compute the partial sum up to $n = 5$:

$$\sum_{n=2}^{5} \frac{n-1}{n^3}$$

$$\begin{aligned} n=2: &\quad \frac{1}{8} = 0.125 \\ n=3: &\quad \frac{2}{27} \approx 0.07407 \\ n=4: &\quad \frac{3}{64} \approx 0.046875 \\ n=5: &\quad \frac{4}{125} = 0.032 \end{aligned}$$

Total up to $n = 5$:

$$0.125 + 0.074 \approx 0.199, \quad 0.199 + 0.0468 \approx 0.2458, \quad 0.2458 + 0.032 = 0.2778$$

Continuing to $n = 6$:

$$\frac{5}{216} \approx 0.0231, \quad \text{total} \approx 0.3009$$

$$n = 7: \frac{6}{343} \approx 0.0175 \Rightarrow \text{total} \approx 0.3184$$

$$n = 8: \frac{7}{512} \approx 0.01367 \Rightarrow \text{total} \approx 0.3321$$

$$n = 9: \frac{8}{729} \approx 0.01097 \Rightarrow \text{total} \approx 0.3431$$

$$n = 10: \frac{9}{1000} = 0.009 \Rightarrow \text{total} \approx 0.3521$$

Continuing up to $n = 20$:
Each subsequent term adds less. By $n = 20$:

$$\frac{19}{20^3} = \frac{19}{8000} \approx 0.002375$$

Cumulative addition from $n = 11$ to $n = 20$ is approximately decreasing terms, each contributing about 0.002, so maybe adding around 0.02 over 10 terms.

$$\text{Total after } n = 20 \approx 0.35 + 0.02 = 0.37$$

Still quite below the infinite sum, which should converge to approximately 0.4428. So the partial sums are approaching that value, albeit slowly.
Since the terms decay like $\frac{1}{n^2}$, the tail from $n = N$ onward behaves like the integral:

$$\int_N^\infty \frac{x-1}{x^3}\, dx \approx \int_N^\infty \left(\frac{1}{x^2} - \frac{1}{x^3}\right) dx \sim \frac{1}{N}$$

So convergence is slow. Anyway, the analytical derivation shows it's $p - q$, which aligns with the logic.
Perhaps I was doubting due to partial sums not reaching close quickly enough, but mathematically, the steps seem correct.
Hence, the answer should be $p - q$, so boxed:

$\boxed{p - q}$

Final Answer

$\boxed{p - q}$

While solving the problem, we started by considering the double sum $\sum_{j=1}^{\infty} \sum_{k=1}^{\infty} \frac{1}{(j+k)^3}$. We changed variables to $n = j + k$, noting that for each $n \geq 2$, there are $n - 1$ pairs (j, k) such that $j + k = n$. This allowed us to rewrite the double sum as a single series:

$$\sum_{n=2}^{\infty} \frac{n-1}{n^3}.$$

Splitting the fraction $\frac{n-1}{n^3}$ into $\frac{1}{n^2} - \frac{1}{n^3}$, we transformed the series into two separate sums:

$$\sum_{n=2}^{\infty} \left(\frac{1}{n^2} - \frac{1}{n^3}\right).$$

Recognizing these sums in terms of p and q, we found:

$$\sum_{n=2}^{\infty} \frac{1}{n^2} = p - 1 \quad \text{and} \quad \sum_{n=2}^{\infty} \frac{1}{n^3} = q - 1.$$

Subtracting these results gives:

$$(p - 1) - (q - 1) = p - q.$$

Thus, the double sum $\sum_{j=1}^{\infty} \sum_{k=1}^{\infty} \frac{1}{(j+k)^3}$ can be written as $\boxed{p - q}$.

ASC generated CoT (1581 Tokens)

To solve the problem, we need to express the double sum

$$S = \sum_{j=1}^{\infty} \sum_{k=1}^{\infty} \frac{1}{(j+k)^3}$$

in terms of

$$p = \sum_{n=1}^{\infty} \frac{1}{n^2} \quad \text{and} \quad q = \sum_{n=1}^{\infty} \frac{1}{n^3}.$$

First, let's consider changing variables. Let's set $m = j + k$. For each fixed value of m, both j and k can range from 1 up to $m - 1$. So for each integer $m \geq 2$, there are exactly $(m-1) \times 1$ pairs since if j is between 1 and $m-1$, then $k = m - j$ must also be at least 1.

Therefore, the original double summation over all $j, k \geq 1$ can be re-expressed as:

$$S = \sum_{m=2}^{\infty} \left[\sum_{j=1}^{m-1} \frac{1}{m^3} \right].$$

Wait, actually more precisely: Since for each m starting at 2 (since j and k start at 1, their minimum sum is 2?), no wait when $j = 1$ and $k = 1$ gives $j + k = 2$? Wait yes! Actually, j starts at 1 and so does k, so the minimal value of $j + k$ is 2. Therefore, m ranges from 2 upwards, but how many times?

Indeed, for each $m \geq 2$, the number of pairs with $j + k = m$ where $j \geq 1$ and $k \geq 1$ is equal to (number of solutions): If you have two positive integers adding up to m, that's equivalent to putting one divider among m items, which has $m - 1$ ways, hence exactly $m - 1$ such pairs. Hence,

$$S = \sum_{m=2}^{\infty} \left[\left(\sum_{j+k=m} \frac{1}{m^3} \right) \right].$$

But note that for each term with $j + k = m$, they contribute $\frac{1}{m^3}$ per pair. Thus total contribution for each m is

$$\frac{m-1}{m^3}.$$

So therefore:

$$S = \sum_{m=2}^{\infty} \frac{m-1}{m^3}$$

This simplifies to:

$$\sum_{m=2}^{\infty} \frac{m-1}{m^3} = \sum_{m=2}^{\infty} \left(\frac{1}{m^2} - \frac{1}{m^3} \right)$$

Because $\frac{m-1}{m^3} = \frac{m}{m^3} - \frac{1}{m^3} = \frac{1}{m^2} - \frac{1}{m^3}$. Hence splitting this into two separate sums,

$$\sum_{m=2}^{\infty} \frac{1}{m^2} - \sum_{m=2}^{\infty} \frac{1}{m^3}$$

Which equals $(p - \frac{1}{1^2}) - (q - \frac{1}{1^3})$, because $p = \sum_{n=1}^{\infty} \frac{1}{n^2} \Rightarrow \sum_{m=2}^{\infty} \frac{1}{m^2} = p\check{}1$. Similarly, $q = \sum_{n=1}^{\infty} \frac{1}{n^3} \Rightarrow \sum_{m=2}^{\infty} \frac{1}{m^3} = q - 1$. Thus substituting back:

$$(p\check{}1)\check{}(q\check{}1) = p\check{}1\check{}q + 1 = p - q$$

Hmm, wait really? That would mean $S = p - q$? Let me check again step by step. Starting from $S = \sum_{j,k \geq 1} \frac{1}{(j+k)^3}$. Then change variable $m = j + k$; thus: For every m from 2 onwards, the inner count is # $\{ (j,k) \mid j + k = m \}$, which is indeed $m - 1$ terms. Each contributes $1/m^3$. Thus

$$S = \sum_{m=2}^{\infty} \frac{m\check{}1}{m^3}$$

Splitting numerator: $\frac{m\check{}1}{m^3} = \frac{1}{m^2} - \frac{1}{m^3}$. Therefore,

$$S = \sum_{m=2}^{\infty} \left(\frac{1}{m^2} - \frac{1}{m^3} \right) = \left[\sum_{m=2}^{\infty} \frac{1}{m^2} \right] - \left[\sum_{m=2}^{\infty} \frac{1}{m^3} \right]$$

The first bracketed sum is $p - 1$ because p includes $n = 1$. Similarly second bracketed term is $q - 1$. So substituting, $[p - 1] - [q - 1] = p - 1 - q + 1 = p - q$. Wow, so it ends up being just $p - q$. But I should verify this with an example or simple case to make sure. Alternatively, perhaps my initial steps were correct? Another approach could be swapping order of summations etc., but maybe another angle. Wait, but let us compute numerically partial sums? Take $p = \pi^2/6 \approx 1.6449\ldots$, $q =$ Apery's constant $\approx 1.202\ldots$ If our expression says $S = p - q \approx 0.4427\ldots$ Can we approximate S directly via small truncations? Compute S approximately with say j, k going only up to $N = 5$, and see whether roughly similar behavior occurs. However this might not be precise enough, but maybe try smaller components. Suppose instead truncate m up to some finite M. Alternatively think about standard series results. We know that

$$\sum_{m=2}^{\infty} \frac{m-1}{m^3} = \sum_{m=1}^{\infty} \frac{m-1}{m^3}$$

since $m = 1$ term would give 0, so same as shifting index. Alternatively, split

$$\sum_{m=1}^{\infty} \frac{m-1}{m^3} = \sum_{m=1}^{\infty} \left(\frac{m}{m^3} - \frac{1}{m^3} \right) = \sum \left(m^{-2} \right) - \sum \left(m^{-3} \right),$$

except the first term when $m = 1$, $m^{-2} = 1$, and subtracting $1/m^3$ similarly. Wait, hold on:

$$\sum_{m=1}^{\infty} \frac{m-1}{m^3} = \sum_{m=1}^{\infty} \left(m^{-2} - m^{-3} \right) = \left(\sum_{m=1}^{\infty} \frac{1}{m^2} \right) - \left(\sum_{m=1}^{\infty} \frac{1}{m^3} \right) = p - q$$

Exactly. Because the $\frac{m-1}{m^3}$ summed from $m = 1$ becomes (for $m = 1$: zero, since $(1-1)/1^3 = 0$), so effectively from $m = 2$ onward matches the previous result. So indeed, $\sum_{m=2}^{\infty} \frac{m-1}{m^3} = \sum_{m=1}^{\infty} \ldots$ since first term cancels out. Either way, regardless of indexing, the final conclusion holds. Thus the answer is simply $p - q$. Hmm, seems too straightforward—maybe I made a mistake somewhere? Alternative perspective: Consider exchanging the order of summation somehow else, but likely leads to the same place. Alternatively, suppose someone tried integrating or generating functions? Not necessary here once algebraic manipulation shows the result cleanly. Therefore, after careful consideration through multiple perspectives confirming the arithmetic manipulations, I believe the desired double sum simplifies neatly to $p - q$. So writing in boxed notation:

$$\boxed{p - q}$$

Final Answer $\boxed{p - q}$

D Steering Hyperparameters

Table 3: Hyperparameters for three different models.

Model	γ	Layer Index
DeepSeek-Distill-Qwen-7B	0.275	21
DeepSeek-Distill-LLaMA-8B	0.46	21
QwQ-32B	0.50	57

Table 3 summarizes the hyperparameters used for steering in our reasoning models. The steering strength γ is selected based on γ_{max}, as derived in Section 4, and the choice of layer index is determined empirically. Early layers are avoided because representations are still underdeveloped, while injecting at the final layers has limited impact due to diminished transformation capacity. Therefore, we select a mid-layer range where representations are sufficiently structured yet still amenable to effective steering. This middle ground provides a practical trade-off between steerability and representational richness.