Activation Steering for Chain-of-Thought Compression

Seyedarmin Azizi[1]*, Erfan Baghaei Potraghloo[1], Souvik Kundu[2], Massoud Pedram[1]

[1]University of Southern California, USA [2]Intel AI, USA

Abstract

Large language models (LLMs) demonstrate strong performance on multi-step reasoning tasks by producing intermediate explanations, commonly referred to as chains of thought (CoTs). However, the generated rationales are typically verbose, consuming many additional tokens, and thus degrading throughput and increasing inference energy consumption. Interestingly, we find that verbose and concise CoTs correspond to distinct regions in the model's intermediate activation space, suggesting that verbosity is a steerable latent attribute. Building on this observation, we develop an inference-time method to automatically steer the model response towards concise reasoning traces without updating model parameters. Our method, dubbed *ASC* (Activation-Steered Compression), generates concise CoTs by directly adjusting internal representations via activation steering. A key component of ASC is **Contrastive Energy-Based Steering (CES)**, a principled procedure to learn a *single* steering vector from a small set of verbose–concise CoT pairs by optimizing a length-normalized contrastive energy objective. To further ensure reliable steering and preserve general utility, CES enforces a differentiable **KL trust region** during steering vector optimization, explicitly constraining the distribution shift within a specified budget. With only 100 pairs of verbose–concise examples, ASC reduces the generated token length by as much as 69.4% across five reasoning benchmarks (MATH500, GSM8K, LiveCodeBench, GSM8K-Hard, and AQuA-RAT) while maintaining accuracy across models with 1.5B, 7B, 8B, and 32B parameters. On MATH500, ASC achieves an end-to-end inference speed-up of 2.7× on an 8B model.

*Correspondence author (seyedarm@usc.edu)

1 Introduction

Explicit reasoning traces, commonly known as chains of thought (CoTs), significantly enhance the performance of large language models (LLMs) in multistep reasoning and agentic tasks, including mathematical problem solving and code generation [30, 7, 28]. However, this advantage often comes with the drawback of generating unnecessarily lengthy and verbose rationales [4, 31]. Due to the large number of generated tokens, this verbosity increases both the computational burden and the LLM key-value (KV) cache memory cost. Thus, long CoTs pose challenges for deployment in throughput-sensitive and memory-limited scenarios [5]. Additionally, such *overthinking* endangers the quality of generation, due to often noisy and redundant self-verification steps [4].

Existing methods for CoT reasoning compression can be broadly categorized into three components: *(i) retraining-based method* that fine-tunes a model to produce shorter rationales, using techniques such as knowledge distillation [23] or reasoning within compact latent tokens [34]; *(ii) prompt-engineering method* that employs carefully designed instructions to encourage models to "reason briefly" by utilizing contrastive demonstrations or symbolic sketches over verbose prose [6, 33]; and *(iii) heuristic early-exit method* that halts generation once a confidence or entropy threshold is reached [32]. However, these approaches come with their associated limitations. For example, retraining-based methods incur costly training overhead on top of a pre-trained model, limiting off-the-shelf adoption. Prompt-based approaches are brittle: reasoning models often fail to faithfully obey length directives, and aggressive compression typically degrades answer quality. Thus, an efficient and reliable alternative for concise CoT generation remains underexplored.

Findings of the Association for Computational Linguistics: ACL 2026, pages 36676–36687

To mitigate this gap, we first investigate the difference between verbose CoTs and their concise counterparts in an intermediate-layer activation space of the model (see Section 4). We find that the two CoT types are separable on the t-SNE cluster plots [27], suggesting that the model internally represents verbose and concise reasoning styles in distinct regions. Based on this observation, we then ask the following fundamental question: ***Is it possible to constrain the CoT token generation of an LLM by modifying its hidden representations without additional model training?***

We answer this question affirmatively by presenting **Activation-Steered Compression (ASC)**, an inference-time activation steering method, which reduces the CoT verbosity by learning a single steering vector and injecting it into the residual stream during decoding. ASC does not modify model weights: it learns only a single vector on top of a frozen base model and applies it as an activation-level intervention at inference time. As a result, ASC is a lightweight solution for open-weight models where internal activations are accessible. Moreover, ASC is orthogonal to existing compression approaches, such as prompt-based conciseness constraints [31] or early-exit heuristics [32], which can be composed with steering.

Crucially, ASC replaces the commonly used *mean-of-differences* (MoD) activation steering heuristic with a standalone, principled method: **Contrastive Negative Log-Likelihood Energy-Based Steering (CES).** [1] MoD constructs a steering vector by averaging activation differences between two sets of exemplars; while simple, this procedure offers no guarantee that the resulting direction will induce consistent or meaningful changes in the model's output distribution. In practice, MoD vectors can be noisy, highly sensitive to layer selection, and brittle to the choice of steering strength, often requiring careful manual calibration to avoid negligible effects or gibberish generations. Moreover, because MoD operates at the level of model's hidden state only, it does not directly optimize for any sequence-level behavior, leaving a gap between activation steering and actual generation outcomes.

CES addresses these limitations by learning the steering vector through a contrastive sequence-level objective that explicitly ranks concise reasoning traces above verbose ones under the steered model. By coupling this objective with an **explicit KL trust region**, CES provides both a direct optimization signal aligned with the desired generation behavior and a principled mechanism to control distributional drift, yielding more reliable and robust steering than heuristic activation averaging.

In general, our contributions can be summarized as follows:

1. We conceptualize CoT verbosity as a *latent, steerable dimension* of model behavior, reframing the concise CoT generation problem as a representation-level control problem rather than an output-level post-processing problem.

2. We introduce CES, a principled procedure for learning a single steering vector by optimizing a contrastive, length-normalized energy objective on paired verbose–concise reasoning traces. CES is lightweight and does not require updates to model parameters.

3. We ensure stable steering via an explicit **KL trust region** during CES optimization, constraining the steered distribution within a specified budget on a general text set, thereby preserving general utility.

4. We conduct extensive experiments on five reasoning benchmarks (MATH500, GSM8K, LiveCodeBench, GSM8K-Hard, AQuA-RAT) and four model sizes (1.5B, 7B, 8B, and 32B), showing that ASC reduces CoT length by up to **69.35%** without accuracy degradation, and outperforms optimization-based steering baselines (BiPO, strong MoD). On MATH500, ASC achieves an end-to-end inference speedup of **2.7**$\times$ on an 8B model.

2 Background

Chain-of-Thought Prompts. 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" [30]. Several enhancements have refined this approach. For example, self-consistency [28] samples multiple rationales and selects the response supported by the majority. Although effective, these methods often significantly increase the length of the output. A recent study [4] showed that the o1-style

[1] Throughout the paper, we use the terms "Contrastive Negative Log-Likelihood Energy-Based Steering" and "Contrastive Energy-Based Steering" interchangeably.

reasoning models frequently produce excessively long CoTs — even for simple questions like "What is 2 + 3?". This is potentially due to unnecessary self-verification and backtracking. We term these inefficiencies as ***verbosity***, and aim to address them through inference-time activation-level steering.

Activation Steering. Linear activation editing has emerged as a lightweight alternative to fine-tuning. Activation Addition (ActAdd) demonstrates that adding a direction corresponding to "$<$ |positive sentiment| $>$" can change the tone of the output [26]. Activation engineering has now found use cases in style transfer [15], factual correction [18], and gender debiasing [16]. However, existing work has yet to investigate the correlation between activation steering and CoT efficiency.

3 Related Work

Previous work has primarily sought to achieve CoT efficiency through methods that *require additional training*. For example, knowledge distillation schemes learn concise rationales [23], while latent token approaches embed reasoning in compact vectors [34]. Apart from these, reinforcement-learning-based trajectory shortening[2] (e.g., THINKPRUNE [13]), and latent-reasoning optimizations to fine-tune internal deliberation steps [2] have also been effective. However, these techniques incur considerable computational cost or architectural modifications.

In addition to training-based approaches, there are methods that include *prompt engineering*, which involves carefully crafting prompts to enforce concise responses [6, 33]. Recently proposed *early exit methods* have also demonstrated effectiveness in concise token generation based on confidence-based thresholding [32]. However, these methods require significant manual selection of prompts and hyperparameters and generally incur a non-negligible drop in generation quality.

The chain of drafts (CoD) [31] and activation-steered instruction-following approaches [24] reduce verbosity by embedding explicit length constraints in the prompt. For example, [24] limits the final answer to a specified number of sentences by controlling it via an activation-steering-based method. 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 models [10].

Optimization-based steering. A recent line of work learns a single steering direction by optimizing sequence-level objectives on paired data rather than computing mean activation differences. BiPO [1] formalizes this via a DPO-inspired objective that operates on likelihood ratios relative to the unsteered model. CES differs in both its objective and control mechanism: (i) CES ranks verbose vs. concise trajectories using a length-normalized energy under the steered model rather than reference-relative likelihood ratios, aligning the learning signal with concise reasoning, and (ii) CES enforces a KL trust region during optimization, directly constraining distribution shift within a specified budget on a general text set. This makes CES a targeted and controllable steering procedure, and empirically outperforms BiPO at matched calibration budgets (Section 6.2, Table 3).

Amortized steering. Approaches such as CASAL [20] amortize steering into weights through submodule training. ASC is complementary: it is strictly inference-time (plug-and-play, reversible), optimizes a single vector from $\approx$100 pairs, and composes with other inference-time controls. The direction identified by the ASC could later be distilled into weights when activation-level access is not available.

The closest work to ours is SEAL [3], 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–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, model-training-free complement to SEAL's category-based calibration.

4 Motivational Study

To understand the differences between the representations of verbose natural-language CoTs and their concise counterparts, we design a key experiment. Specifically, we sample questions from the MATH500 [12] and GSM8K [7] benchmarks and

[2]The trajectory of a prompt response refers to the entire sequence of steps, intermediate thoughts, actions, and results that a language model or AI agent takes to generate its final output.

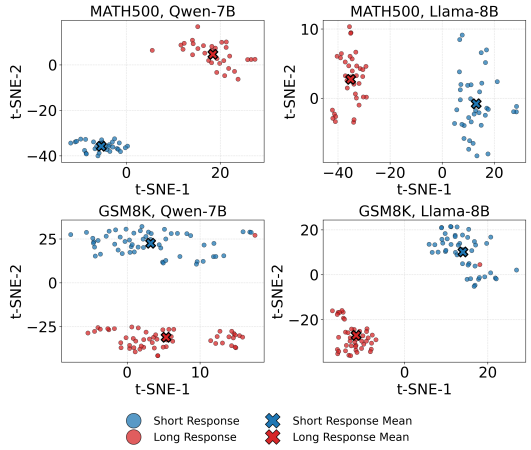


Figure 1: t-SNE visualization of residual stream representations for verbose and concise CoT responses.

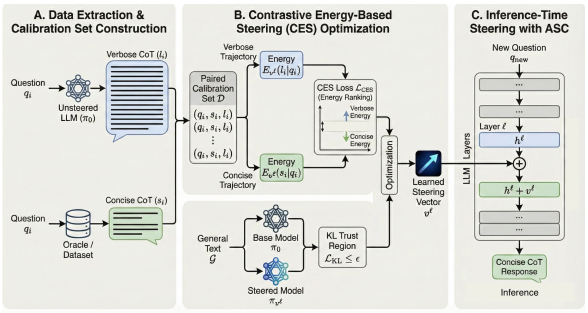


Figure 2: ASC implementation overview using Contrastive Energy-Based Steering (CES) to learn a single steering vector under a KL trust region.

use two open-weight reasoning models: DeepSeek-Distill-Qwen-7B and DeepSeek-Distill-LLaMA-8B. For each sample, we generate two versions of the CoT: (1) a verbose reasoning chain produced by the model itself under standard prompting and (2) a concise reasoning chain produced by GPT-4o prompted to minimize natural language verbosity. An example of such a pair of responses is shown in Figure 5 in Appendix D. We feed each input independently into the model and extract the output of a transformer block at a predetermined later layer (e.g., layer 16; see Appendix A). We term this output the *residual stream activations*. We select a later layer because activation spaces for different types of thought are generally easier to separate at deeper layers [3]. We then visualize the two-dimensional t-SNE projection [27] of these two categories of CoT activations, depicted in Figure 1. Interestingly, the plot reveals a clear separation between activation clustering for the two reasoning styles. This motivates the construction of a *steering vector* that can guide the model's hidden representations toward concise responses.

5 Methodology

We now present **Activation-Steered Compression (ASC)**, a method that shifts the model's hidden representations toward the subspace associated with concise, math-centric CoTs. The method is summarized in Figure 2 and is detailed in the following sections. ASC relies on a single activation-addition direction to shift generation from verbose to concise CoTs. The main question is: how should we *learn* this direction from a small calibration set in a way that directly targets the desired behavior (concise traces) while avoiding uncontrolled drift that can degrade general utility? CES answers this by optimizing a steering vector with a contrastive objective and an explicit KL trust region.

5.1 Setup and calibration pairs

We assume access to a small calibration set of paired trajectories $\mathcal{D} = \{(q_i, s_i, l_i)\}_{i=1}^{N}$, where q_i is a question, s_i is a concise CoT, and l_i is a verbose CoT for the same question. The verbose CoT l_i is generated by the target model with standard CoT prompting [30]. The concise CoT s_i is obtained via one of two routes: (a) *ground-truth reference solutions* already present in the benchmark's training split, or (b) *oracle-generated traces* produced by a strong reasoning model (e.g., GPT-4o/GPT-5) prompted to use concise reasoning with minimal English. For route (b), we apply strict correctness filtering: only oracle-generated traces producing the verified correct final answer are retained, ensuring the steering vector learns to compress *style* rather than content. The oracle receives only the question and a conciseness instruction; it does not access the target model's internal computations, and the steering vector is learned entirely within the target model's own activation space (Section 5), so the oracle serves only as a textual reference for style. All 100 pairs are sampled from the training/-calibration splits, not the evaluation set, to avoid test-set contamination. This paired construction ensures that the supervision targets a *style dimension* (verbosity) while maintaining the semantic intent (solving the same q_i).

5.2 Latent intervention and steered policy

Let π_0 denote the frozen base model distribution. Let $h^\ell(x_t) \in \mathbb{R}^d$ denote the residual-stream activation[3] at transformer layer ℓ at decoding step t.

[3]By residual stream we mean the per-token hidden-state vectors carried along the residual pathway between trans-

CES defines a steered policy π_{v^ℓ} by adding a single vector v^ℓ to the residual stream at a chosen layer ℓ during decoding:

$$h^\ell(x_t) \leftarrow h^\ell(x_t) + v^\ell \quad \forall\, t \in [1, \texttt{decoding\_steps}]\,. \tag{1}$$

The vector is transmitted across time steps because verbosity is expressed globally across a trajectory rather than at a single token. This intervention is identical to the standard activation-addition steering; CES changes only how v^ℓ is obtained.

5.3 Mean-of-Differences (MoD) Steering Baseline

Before introducing our proposed objective, we describe the standard Mean-of-Differences (MoD) method, which serves as a baseline for extracting a steering vector. Given a set of calibration pairs (q_i, l_i, s_i), where q_i is the question, l_i is the verbose response and s_i is the concise response, the MoD steering vector v^ℓ is computed as:

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

where h^ℓ is the residual stream in layer ℓ, and $\oplus$ denotes string concatenation. This vector is then used to steer the model's activation toward concise reasoning. However, the MoD requires manual tuning of steering strength and layer selection, with no guarantee that the steering will consistently guide the generation toward the desired behavior. This makes it sensitive to these choices and often unstable across different models or tasks. In the next sections, we define length-normalized energy and contrastive energy steering to address these issues.

5.4 Length-normalized energy definition

Negative log-likelihood (NLL) and energy in Energy-Based Models (EBMs) are closely linked: minimizing NLL assigns low energy to high-probability states, making energy scores a natural metric for ranking model behaviors.

For a prompt q and any response trajectory $y = (y_1, \ldots, y_{|y|})$, we define the length-normalized energy under π_{v^ℓ} as the length-normalized negative log-likelihood:

$$E_{v^\ell}(y \mid q) = \frac{1}{|y|} \sum_{t=1}^{|y|} -\log \pi_{v^\ell}(y_t \mid q, y_{<t})\,. \tag{2}$$

This definition has two important properties. First, $E_{v^\ell}(y \mid q)$ is a per-token cross-entropy of the steered model on the fixed trajectory y. Second, dividing by $|y|$ prevents a trivial bias toward shorter strings that arises if one uses summed NLL; CES is not rewarding shortness per se, but rather increasing the *average tokenwise likelihood* of pursuing concise trajectories relative to verbose ones.

5.5 Contrastive energy ranking objective

CES learns v^ℓ by ranking the concise trace below the verbose trace in energy:

$$\mathcal{L}_{\text{CES}}(v^\ell) = \mathbb{E}_{(q,s,l)\sim\mathcal{D}} \Big[\text{softplus}\big(E_{v^\ell}(s \mid q) - E_{v^\ell}(l \mid q)\big) \Big]. \tag{3}$$

What the gradient of $\mathcal{L}_{\text{CES}}$ is doing. For a calibration triplet (q_i, s_i, l_i), define the length-normalized energy gap

$$\Delta_i(v^\ell) \triangleq E_{v^\ell}(s_i \mid q_i) - E_{v^\ell}(l_i \mid q_i), \tag{4}$$

and the per-pair loss as $\mathcal{L}_i(v^\ell) = \text{softplus}\big(\Delta_i(v^\ell)\big)$ .. Using $\frac{d}{dx}\text{softplus}(x) = \sigma(x)$, the gradient is

$$\nabla_{v^\ell}\mathcal{L}_i(v^\ell) = \sigma\Big(\Delta_i(v^\ell)\Big) \cdot \Big(\nabla_{v^\ell} E_{v^\ell}(s_i \mid q_i) - \nabla_{v^\ell} E_{v^\ell}(l_i \mid q_i)\Big). \tag{5}$$

Since we have $\nabla_{v^\ell} E_{v^\ell}(y \mid q) = -\frac{1}{|y|}\sum_{t=1}^{|y|} \nabla_{v^\ell} \log \pi_{v^\ell}(y_t \mid q, y_{<t})$, under gradient descent updates $v^\ell \leftarrow v^\ell - \eta \nabla_{v^\ell}\mathcal{L}_i$, when $\Delta_i(v^\ell)$ is positive, the sigmoid factor is close to one, and the update approximately performs

$$v^\ell \leftarrow v^\ell - \eta\, \nabla_{v^\ell} E_{v^\ell}(s_i \mid q_i) + \eta\, \nabla_{v^\ell} E_{v^\ell}(l_i \mid q_i), \tag{6}$$

which decreases the length-normalized NLL of the concise trace s_i (increasing its per-token log-likelihood) while increasing the length-normalized NLL of the verbose trace l_i (decreasing its per-token log-likelihood) for the same prompt. When $\Delta_i(v^\ell)$ is negative, $\sigma(\Delta_i(v^\ell))$ shrinks and the magnitude of the update decreases, stabilizing optimization once the concise trace is already preferred.

5.6 KL trust region for utility preservation

Pure optimization of $\mathcal{L}_{\text{CES}}$ can produce v^ℓ that overfits the calibration pairs and causes unwanted distribution drift. CES therefore enforces a trust region

former layers (i.e., the $T \times d$ activations h^ℓ at layer ℓ).

relative to the base model π_0 on a general text distribution $\mathcal{G}$ using a tokenwise KL divergence:

$$\mathcal{L}_{\mathrm{KL}}(v^\ell) = \mathbb{E}_{x\sim\mathcal{G}}\left[\frac{1}{|x|}\sum_{t=1}^{|x|}\mathrm{KL}\Big(\pi_0(\cdot \mid x_{<t}) \,\|\, \pi_{v^\ell}(\cdot \mid x_{<t})\Big)\right]. \tag{7}$$

We optimize under an explicit budget ϵ:

$$\min_{v^\ell} \mathcal{L}_{\mathrm{CES}}(v^\ell) \quad \text{s.t.} \quad \mathcal{L}_{\mathrm{KL}}(v^\ell) \le \epsilon\,. \tag{8}$$

Interpretation. The constraint limits how far π_{v^ℓ} can move from π_0 on generic text. This conceptually resembles trust-region policy optimization in RL [21], allowing for targeted behavioral change (verbosity) while discouraging broad degradation in unrelated capabilities. Unconstrained optimization of v^ℓ can overfit the calibration pairs by pushing the residual stream into regions where the model's token distribution becomes sharply distorted, leading to fragile behavior (e.g., sudden accuracy collapse) under small changes in scaling or prompts. The trust region directly prevents this failure mode by limiting the average token-wise deviation of the steered policy from the base model on a broad text distribution $\mathcal{G}$.

5.7 Optimization in practice

In practice, we optimize the steering vector using a penalized objective rather than a dynamically adjusted dual formulation. Specifically, we minimize

$$\mathcal{L}(v^\ell) = \mathcal{L}_{\mathrm{CES}}(v^\ell) + \lambda \cdot \max\big(\mathcal{L}_{\mathrm{KL}}(v^\ell) - \epsilon,\ 0\big), \tag{9}$$

where λ is a fixed coefficient and ϵ is the target KL budget. This hinge-style penalty[4] encourages the optimizer to satisfy the trust-region constraint while avoiding unnecessary suppression of the CES objective once the KL budget is met. This parallels the practical success of PPO-style relaxations of hard trust-region constraints in reinforcement learning [22]: rather than solving a constrained dual formulation at every step, a penalized proxy yields stable updates at a fraction of the optimization complexity. This simple formulation yields stable optimization behavior and allows us to use a single KL budget ϵ consistently across models and tasks. We set $\lambda = 20$ and $\epsilon = 2 \times 10^{-2}$, selected once via preliminary tuning and then kept fixed across all models and tasks.

Inference. After learning v^ℓ, ASC applies the same activation addition during standard autoregressive decoding for new prompts, producing concise CoTs without any model fine-tuning.

6 Experiments

6.1 Experimental Setup

Models, Datasets, and Baselines. We evaluate ASC on several recent open-source reasoning models: DeepSeek-R1-Distill-LLaMA-8B [8], DeepSeek-R1-Distill-Qwen-1.5B [9], DeepSeek-R1-Distill-Qwen-7B [9], and QwQ-32B [25]. The evaluation is performed on multiple reasoning benchmarks, including MATH-500 [12], GSM8K [7] and LiveCodeBench [14]. As baselines, we compare ASC against vanilla CoT prompting (no steering), CoD [31], DEER [32], TCC [19], and SEAL [3].

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. All reported accuracies are **pass@1** under this sampling configuration on all benchmarks, including code generation on LiveCodeBench. 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`. For each task, we reserve 100 verbose–concise response pairs drawn from the task's training/calibration split to optimize the steering vector (a separate vector per benchmark in the main tables; cross-task transfer is studied in Section 7). All other hyperparameters are provided in Appendix A.

6.2 Main Results

Table 1 presents the performance of ASC compared to baseline CoT compression techniques. On the DeepSeek-R1-Distill-Qwen-7B model, ASC reduces CoT length by up to **69.35%** without sacrificing accuracy, outperforming prior methods in compression effectiveness. On the llama model and the GSM8K dataset, ASC achieves a compression rate of **67.43%** and slightly improves response accuracy by **0.2%**, matching or exceeding the vanilla CoT

[4] A hinge-style penalty gives zero loss for confident, correct predictions (outside the margin), a small loss for near-boundary points, and an increasing loss for misclassifications.

baseline. On MATH500, ASC achieves a **33.8%** reduction in CoT length, again outperforming all baselines while maintaining equivalent accuracy.

Table 1: Performance comparison of CoT, TCC, DEER, CoD, SEAL, and ASC on reasoning tasks. M1, M2, M3, and M4 denote Deepseek-R1-Distill-Qwen-7B, Deepseek-R1-Distill-LLaMA-8B, and QwQ-32B, and DeepSeek-R1-Distill-Qwen-1.5B

Model	Method	MATH500		GSM8k	
		Acc. (%) ↑	Tokens ↓	Acc. (%) ↑	Tokens ↓
M1	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	88.8	**1221**	88.8	**508**
M2	CoT	89.2	3554	89.1	2610
	DEER	89.2	2830	89.3	2124
	CoD	88.8	3028	89.1	914
	ASC	**89.2**	**2286**	**89.3**	**850**
M3	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	94.0	**2165**	96.4	**801**
M4	CoT	69.0	5826	**76.0**	1570
	DEER	67.8	2497	74.7	984
	CoD	69.1	3494	75.4	1044
	ASC	**70.4**	**2133**	75.4	**684**

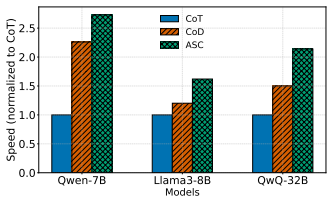


Figure 3: Speed comparison of CoT, CoD, and ASC. We used MATH500 for this experiment.

On the larger QwQ-32B model, ASC compresses CoTs by **52.0%** and **47.64%** on MATH500 and GSM8K, respectively. Notably, on MATH500, it also yields a **0.4%** improvement in accuracy over the vanilla CoT. Upon inspection, we find that the high token count in some model responses arises primarily from either examples exceeding their token budget or exhibiting excessive backtracking and thought switching during generation. This aligns with the observations of [29], who show that LLMs similar to o1 tend to generate longer responses when frequently switching between reasoning paths without pursuing any one. This behavior, termed *underthinking*, often manifests as verbose outputs filled with abandoned or partially developed reasoning trajectories. Among the models evaluated, QwQ-32B appears particularly susceptible to this issue. ASC mitigates this behavior by promoting concise reasoning and earlier stopping, thereby suppressing the long CoTs. In summary, across all models and datasets, ASC consistently achieves the **highest CoT compression** while preserving accuracy.

Model	CoT Method	Accuracy (%)↑	CoT Tokens↓
DeepSeek-R1-Distill-Qwen-7B	Vanilla CoT	38.7	6380
	DEER	40.3	2582
	ASC	**41.1**	**1735**
QwQ-32B	Vanilla CoT	**58.8**	8050
	DEER	56.6	3677
	ASC	57.5	**3038**

Table 2: Performance of ASC on LiveCodeBench.

To further demonstrate the effectiveness of ASC, we evaluated it on the LiveCodeBench [14] code generation task, a standard benchmark for reasoning models. We use DeepSeek-R1-Distill-Qwen-7B and QwQ-32B as the base models, and compare ASC against the vanilla CoT baseline and DEER. The results are presented in Table 2. For both models, ASC achieves the lowest CoT token count, reducing the vanilla CoT usage by factors of 3.67× and 2.64×, respectively. Moreover, ASC outperforms both vanilla CoT and DEER on the Qwen-7B model, and also surpasses vanilla CoT on QwQ-32B.

Comparison with optimization-based steering. We compare ASC against BiPO [1], a DPO-inspired optimization-based steering method, and against MoD with its steering strength α swept to find the best operating points that remain *safe* (within 2% of vanilla-CoT accuracy). For BiPO, we adapt the official implementation and train with the same 100-pair calibration set as ASC. Table 3 reports results on MATH500 for two model scales. Across both models, ASC achieves higher compression than both BiPO and the best safe MoD operating point while matching or exceeding their accuracy. Notably, MoD requires per-model α calibration to remain accurate; ASC uses a fixed configuration (no α tuning) across all models (see Section 7.2).

Stress-testing on harder / distinct reasoning. To probe generalization beyond MATH500/GSM8K/LiveCodeBench, we evaluate ASC on two additional benchmarks: GSM8K-Hard [11], which substitutes larger, harder numeric values into GSM8K questions, and AQuA-RAT [17], a multiple-choice algebraic word-problem benchmark. We use DeepSeek-R1-Distill-Qwen-7B with the default ASC

Model	Method	Acc. (%) ↑	Token Ratio ↓	Compr. (%) ↑
DeepSeek-R1 Distill-Qwen-7B	Vanilla CoT	88.8	1.00×	—
	BiPO [1]	89.0	0.57×	43.0
	MoD (α=0.65, safe)	89.0	0.39×	61.3
	MoD (α=0.75, safe)	88.6	0.37×	63.5
	ASC (ours)	88.8	**0.31**×	**69.4**
DeepSeek-R1 Distill-Qwen-1.5B	Vanilla CoT	69.0	1.00×	—
	BiPO [1]	70.0	0.45×	55.0
	MoD (α=0.30, safe)	69.2	0.49×	51.4
	MoD (α=0.35, safe)	68.6	0.43×	56.7
	ASC (ours)	**70.4**	**0.37**×	**63.4**

Table 3: Comparison with optimization-based steering (BiPO) and strength-swept MoD on MATH500. For MoD we report the best operating points that remain within 2% of vanilla-CoT accuracy. All methods use the same 100-pair calibration budget as ASC.

configuration. As shown in Table 4, ASC maintains or improves accuracy (+0.4% on GSM8K-Hard, 0.0% on AQuA-RAT) while reducing reasoning tokens by 18–25%. These results support the meta-review concern about under-thinking risks: even on harder arithmetic (GSM8K-Hard) and on a format (multiple choice) where concise selection-style reasoning is natural, ASC does not sacrifice accuracy.

Benchmark	Method	Acc. (%) ↑	Tokens ↓	Compr. ↑
GSM8K-Hard	Vanilla CoT	53.5	3575	—
	ASC (ours)	**53.9**	**2942**	**17.7%**
AQuA-RAT	Vanilla CoT	85.8	2222	—
	ASC (ours)	85.8	**1676**	**24.6%**

Table 4: ASC on harder / distinct reasoning benchmarks with DeepSeek-R1-Distill-Qwen-7B. ASC preserves or improves accuracy while cutting 18–25% of tokens.

End-to-end latency. Since one of the primary goals of CoT compression is to reduce end-to-end response latency, we measure the average generation time for three models, DeepSeek-R1-Distill-LLaMA-8B, DeepSeek-R1-Distill-Qwen-7B, and QwQ-32B, 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 3. Specifically, the reported speed value for a method is obtained by dividing the average end-to-end response time of the CoT baseline by that of the method, averaged over all evaluation samples; a value of 2.7× therefore means ASC completes generation 2.7× faster than standard CoT in wall-clock. The results indicate that ASC improves the generation speed of CoT-based reasoning by up to 2.7× on MATH500, with no loss in answer accuracy. Analogous speedups on GSM8K are reported in Appendix E (up to 3.26× on LLaMA3-8B), confirming the latency gains transfer across benchmarks.

7 Discussion and Ablations

7.1 Cross-Task Generalization

In this section, 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 conducted 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 100 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 5. 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 is attributable to latent representations.

Dataset	Steering Vector Source	Accuracy (%) ↑	CoT Tokens ↓
MATH500	MATH500 (in-dataset)	88.8	1221
	GSM8K (cross-dataset)	88.8	1482
GSM8K	GSM8K (in-dataset)	88.8	508
	MATH500 (cross-dataset)	88.4	611

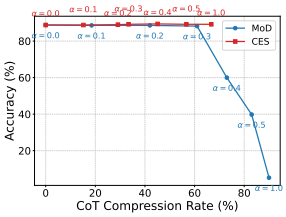
Table 5: Performance of ASC on MATH500 and GSM8K using dataset-specific vs. cross-dataset steering vectors. The model used is DeepSeek-Distill-Qwen-7B.

7.2 CES vs Mean of Differences Steering

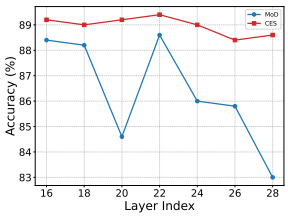
A key limitation of MoD steering is that its direction cannot be applied directly and must be scaled by a hand-tuned strength α,

$$h^{\ell}(x_t) \leftarrow h^{\ell}(x_t) + \alpha v^{\ell}, \tag{10}$$

making the accuracy–compression trade-off highly sensitive. Figure 4a (DeepSeek-R1-Distill-Qwen-7B on MATH500) shows an abrupt regime change:



(a) Effect of the steering strength α on MATH500 accuracy.



(b) Effect of the steering layer on MATH500 accuracy.

Figure 4: Comparison of MoD and CES

increasing α from 0.4 to 0.5 improves compression from 73% to 83%, but decreases accuracy from 60.0% to 39.8%; by $\alpha = 1.0$, the accuracy collapses to 5.2% while forcing close to a 90% compression. This instability is consistent with MoD's heuristic nature, where scaling a difference vector can push hidden states off-manifold and yield degenerate generations. In contrast, CES exhibits a smooth and reliable trade-off over the same sweep $\alpha \in \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 1.0\}$: the accuracy remains essentially unchanged (88.8% $\rightarrow$ 89.2%) while the compression increases steadily up to 69.35% at $\alpha = 1.0$. We emphasize that this sweep is an *ablation* of the already-learned vector; in all main-table experiments, ASC applies v^ℓ directly (i.e., $\alpha = 1$) as in Eq. (1) and does not tune a strength multiplier. The learned vector's norm is shaped by the CES objective itself under the explicit KL trust region (fixed budget $\epsilon = 2 \times 10^{-2}$ across all models and tasks), which is what makes **per-task strength tuning unnecessary**. This is in sharp contrast to the MoD, which must be calibrated per model.

We also evaluate the robustness of the choice of steering layer. Figure 4b reports MATH500 accuracy at a fixed compression level of $\approx 50\%$ while varying the intervention layer from 16 to 28. MoD is substantially layer-sensitive, with an accuracy ranging from 88.6% down to 83.0%, while CES remains stable across layers (88.4%–89.4%). This robustness follows from CES's sequence-level optimization, which adapts the learned direction to each layer's representational geometry rather than relying on raw activation differences; consequently, CES does not require careful task-specific layer selection to avoid accuracy degradation.

7.3 Effect of ASC on overthinking

ASC reduces overthinking behaviors that lengthen CoT. We quantify two events from model-generated text using simple trigger-based matching. A *self-verification* event begins when the model enters an explicit checking mode (e.g., "Wait, let's check", "to confirm", "double-check", "make sure") and ends when it resumes forward derivation or emits a final conclusion. A *backtracking* event begins when the model retracts a line of reasoning (e.g., "Alternatively,","Another thought",) and then switches to an alternative route. Table 6 reports the average number of these events per solution. The steered model produces substantially fewer self-verifications and backtracks than the unsteered baseline, indicating that steering not only shortens CoTs but also suppresses hesitation loops that contribute little to correctness. This suggests that the contrastive energy objective indirectly discourages latent representations associated with unproductive reasoning patterns, steering generation toward more direct problem-solving trajectories.

Setting	Self-verifications ↓	Backtracking ↓
Unsteered	3.16 ± 0.88	5.34 ± 1.16
Steered (ASC)	1.48 ± 0.67	2.62 ± 0.75

Table 6: Mean $\pm$ standard deviation of self-verification and backtracking events per solution (50 MATH500 problems; DeepSeek-R1-Distill-Qwen-7B).

8 Conclusion

We presented ASC, a novel activation steering framework that reduces CoT verbosity without updating model weights. ASC uses Contrastive Energy-Based Steering to learn a single steering vector from paired verbose–concise traces via a length-normalized contrastive energy objective, while a KL trust region constrains distributional drift. Across multiple reasoning benchmarks and model sizes, ASC achieves substantial CoT compression and speedups without accuracy loss, and is markedly less sensitive than mean-of-differences steering layer choice, while not needing to tune the steering strength. These results suggest that verbosity is a controllable latent axis that can be reliably steered through targeted interventions.

Acknowledgments

This work was supported in part by a grant from the Software and Hardware Foundations (SHF) of the National Science Foundation (NSF).

9 Limitations

Our approach requires access to internal activations and backpropagation through the model to optimize the steering vector, limiting its applicability to open-weight or otherwise introspectable deployments and excluding typical API-only LLM settings. Although CES uses only a small number of paired traces, the learned steering direction can still reflect biases of the pairing procedure (e.g., teacher-generated "concise" rationales), and its behavior may depend on the representativeness of the calibration set for the target domain. Finally, while the KL trust region empirically improves stability and utility preservation (and is explicitly computed on *non-reasoning* general text to constrain drift on broadly applicable capabilities), it does not provide a formal guarantee of downstream task correctness, robustness, or safety. In particular, we do not conduct a comprehensive audit of how ASC steering affects non-reasoning behaviors such as factuality, alignment, or refusal of safety-sensitive prompts; a systematic study of these axes, including whether the learned verbosity direction is entangled with safety-relevant directions in activation space, is an important direction for future work.

References

[1] Yuanpu Cao, Tianrong Zhang, Bochuan Cao, Ziyi Yin, Lu Lin, Fenglong Ma, and Jinghui Chen. 2024. Personalized steering of large language models: Versatile steering vectors through bi-directional preference optimization. *Advances in Neural Information Processing Systems*, 37:49519–49551.

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

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

[4] 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. 2025. Do not think that much for 2+3=? on the overthinking of o1-like llms. *Preprint*, arXiv:2412.21187.

[5] Jeffrey Cheng and Benjamin Van Durme. 2024. Compressed chain of thought: Efficient reasoning through dense representations. *arXiv preprint arXiv:2412.13171*.

[6] Yew Ken Chia, Guizhen Chen, Luu Anh Tuan, Soujanya Poria, and Lidong Bing. 2023. Contrastive chain-of-thought prompting. *arXiv preprint arXiv:2311.09277*.

[7] Karl Cobbe and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

[8] DeepSeek-AI. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

[9] DeepSeek-AI. 2025. Deepseek-r1-distill-qwen-7b. https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Qwen-7B.

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

[11] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. PAL: Program-aided language models. In *International Conference on Machine Learning (ICML)*. Introduces the GSM-Hard benchmark used in this paper.

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

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

[14] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*.

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

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

[17] Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In *Proceedings of the 55th Annual Meeting*

of the Association for Computational Linguistics (ACL), pages 158–167.

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

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

[20] Xinchi Qiu and 1 others. 2025. Hallucination reduction with CASAL: Contrastive activation steering for amortized learning. *arXiv preprint arXiv:2510.02324*.

[21] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR.

[22] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms.

[23] Zhenyi Shen, Hanqi Yan, Linhai Zhang, Zhanghao Hu, Yali Du, and Yulan He. 2025. Codi: Compressing chain-of-thought into continuous space via self-distillation. *arXiv preprint arXiv:2502.21074*.

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

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

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

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

[28] Xuezhi Wang and 1 others. 2023. Self-consistency improves chain of thought reasoning in language models. *ICLR*.

[29] Yue Wang, Qiuzhi Liu, Jiahao Xu, Tian Liang, Xingyu Chen, Zhiwei He, Linfeng Song, Dian Yu, Juntao Li, Zhuosheng Zhang, and 1 others. 2025. Thoughts are all over the place: On the underthinking of o1-like llms. *arXiv preprint arXiv:2501.18585*.

[30] Jason Wei and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *NeurIPS*.

[31] Silei Xu, Wenhao Xie, Lingxiao Zhao, and Pengcheng He. 2025. Chain of draft: Thinking faster by writing less. *Preprint*, arXiv:2502.18600.

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

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

[34] Jason Zhang and Scott Viteri. 2024. Uncovering latent chain of thought vectors in language models. *arXiv preprint arXiv:2409.14026*.

A Steering-vector optimization hyperparameters.

We use the following **optimization hyperparameters** and setup to learn the steering vector across all reported models: Adam optimizer with learning rate 10^{-2}, batch size 2, weight decay 0, and 3000 optimization steps. Unless stated otherwise, we inject the vector at layer 16. For the KL trust region, we use a fixed budget $\epsilon = 2 \times 10^{-2}$ with Lagrangian coefficient $\lambda = 20$, and estimate the KL term using 2000 samples from the WikiText general-text dataset.

B Ablation on the KL budget ϵ and penalty coefficient λ

We ablate the two hyperparameters controlling the KL trust region on MATH500 with DeepSeek-R1-Distill-Qwen-7B. Table 7 varies the KL budget ϵ with λ fixed at its default, and Table 8 varies the penalty coefficient λ with ϵ fixed at its default. Both ablations confirm that ASC's behavior is predictable and well-behaved: loosening the trust region (larger ϵ) or weakening enforcement (smaller λ) produces more aggressive compression at a modest accuracy cost, while the default setting ($\epsilon = 2 \times 10^{-2}$, $\lambda = 20$) sits near the Pareto frontier. Accuracy varies by at most $\pm 0.4\%$ across $\epsilon \in [5 \times 10^{-3}, 5 \times 10^{-2}]$ and $\lambda \in \{10, 20\}$, so ASC is not sensitive to precise hyperparameter values within reasonable ranges. This is consistent with our use of a single fixed (ϵ, λ) configuration across all models and tasks in the main experiments.

C Compute cost of CES optimization

We quantify the one-time cost of learning a steering vector via CES. Setup: DeepSeek-R1-Distill-

ϵ	Accuracy (%) ↑	Avg Tokens ↓
1×10^{-1}	87.2	**1074**
5×10^{-2}	88.4	1204
2×10^{-2} (default)	**88.8**	1221
5×10^{-3}	**88.8**	1296

Table 7: KL budget ϵ ablation (with $\lambda = 20$ fixed). MATH500, DeepSeek-R1-Distill-Qwen-7B.

λ	Accuracy (%) ↑	Avg Tokens ↓
1	87.4	**1107**
10	**88.8**	1288
20 (default)	**88.8**	1221

Table 8: Penalty coefficient λ ablation (with $\epsilon = 2 \times 10^{-2}$ fixed). MATH500, DeepSeek-R1-Distill-Qwen-7B.

Qwen-7B with bf16 precision, 3–4k reasoning tokens per trace, 100 calibration pairs, 3000 CES optimization steps, batch size 2 (matching Appendix A), single NVIDIA A6000 GPU. Under this configuration, the full CES optimization completes in approximately **1.0 hour** of wall-clock ($\sim$1 minute per 50 steps) and uses approximately **28 GB** of peak GPU memory (weights + long-context activations/KV + autograd buffers). CES adds negligible parameter-state overhead beyond the forward/backward pass because it optimizes only a single d-dimensional vector.

Comparison to full fine-tuning. Full fine-tuning of the same 7B model requires storing optimizer state and gradients for all parameters, typically exceeding 70–90 GB of GPU memory and necessitating multi-GPU setups. CES avoids weight updates entirely, keeping memory close to inference-scale. At inference, ASC adds only a single activation-addition per decoding step (cf. Eq. (1)), introducing no measurable per-token latency overhead.

D Concise vs. verbose example pair of responses

Figure 5: 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.

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\[\boxed{-3x^2 + 9x - 11}
\]
```

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>
```

E Additional latency measurements on GSM8K

We repeat the end-to-end latency measurement of Figure 3 on GSM8K to verify that the speedups transfer across benchmarks. Table 9 reports the wall-clock generation speed on GSM8K for CoT, CoD, and ASC, normalized such that vanilla CoT on each model is 1.00×. ASC achieves the highest speedup across all three models and reaches up to 3.26× on LLaMA-3-8B. Consistent with the MATH500 results in the main text, the speedup correlates with the degree of token-level compression: models that were previously generating the most verbose reasoning (LLaMA-3-8B) benefit the most from ASC.

Model	CoT	CoD	ASC
DeepSeek-R1-Distill-Qwen-7B	1.00×	2.08×	**2.27**×
DeepSeek-R1-Distill-LLaMA-8B	1.00×	3.06×	**3.26**×
QwQ-32B	1.00×	1.55×	**1.98**×

Table 9: End-to-end generation speed on GSM8K, normalized to vanilla CoT per model (higher is better). Measured on a single NVIDIA A6000 GPU.