Efficient Reasoning via Constrained Optimization in Latent Space

Zhinan Hou [1] **Xingchen Li**[1] **Keyou You**[1*]

[1]Tsinghua University

Abstract

Large Reasoning Models (LRMs) have shown remarkable reasoning capabilities, yet they still suffer from overthinking, generating redundant reasoning steps which incur substantial token consumption. Existing methods, such as suppressing reflective keywords or forcing shorter reasoning lengths, attempt to mitigate this issue but inevitably truncate necessary steps and induce underthinking, thereby compromising performance. To address this dilemma, we investigate the latent representations and observe that efficient reasoning steps naturally cluster into a concentrated region in latent space, while those deviating from this region tend to produce verbose sequences. To leverage this, we keep reasoning focused within this region via a quadratic program which projects deviating hidden states back into the region. Then we propose a novel training-free framework to achieve efficient reasoning that reduces token generation costs without sacrificing performance. Extensive experiments conducted on four models ranging from 1.5B to 14B, and across six benchmarks in math reasoning, coding, and scientific QA, validate the effectiveness of our method, up to a 12.1% improvement in accuracy while reducing generated tokens by 11.8% to 52.8%. Codes are available at https://github.com/hzn18/Opt4Reasoning.

1 Introduction

Large Reasoning Models (LRMs) have significantly enhanced the ability of artificial intelligence to solve complex tasks [1, 2]. A key technique for this success is Chain of Thought (CoT) reasoning, which enables models to dynamically scale test time computation by generating a sequence of intermediate steps before producing the final answer [3]. This allows the model to break down complex problems into more manageable components, leading to more reliable and accurate outcomes.

However, long CoT introduces significant inference overhead due to the autoregressive nature of these models. Moreover, excessively long reasoning processes frequently induce *overthinking*, a phenomenon where models become trapped in problematic behaviors like meaningless self-hesitation, infinite loops, and redundant verbosity, which finally compromise accuracy [4]. To address this issue, recent studies have attempted to shorten reasoning chains through techniques such as early exit [4, 5] or compression via reinforcement learning and supervised fine-tuning [6, 7]. Yet, these approaches tend to bias models toward pursuing overly short reasoning paths. As a result, they may inevitably abandon valuable reasoning steps and degrade overall performance, inducing the problem of *underthinking* [8]. Therefore, a central challenge is *how to achieve efficient reasoning that reduces token consumption while maintaining or even improving reasoning performance.*

To achieve this, it is crucial to understand how efficient reasoning differs from redundant generation at the representation level. Accordingly, we analyze the hidden representations of reasoning steps in

*Corresponding author: youky@tsinghua.edu.cn

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

the latent space. We discover that efficient reasoning steps naturally form a concentrated region while redundant steps such as meaningless double-checking or irrelevant computations, usually scatters around this region. Then, we examined the geometric distance of the entire reasoning trajectory to this concentrated region. Our analysis reveals that the further a trajectory deviates from this region, the higher the probability that the model produces an incorrect answer or generates an exceptionally long sequence of tokens.

Inspired by these geometric insights, we aim to keep reasoning focused within the concentrated region of the latent space. During the inference process, if the hidden state of the model attempts to deviate from this region, we correct it by projecting the state to the closest point within the region, which is formulated as a Quadratic Programming (QP) problem. This allows us to steer the reasoning back into the target region while preserving the essential semantic information of the current task. In doing so, the model is implicitly guided to generate more effective reasoning steps. Interestingly, this methodology naturally aligns with biological mechanisms of advanced cognition, which similarly depend on the *inhibition* of neural signals to prevent cognitive activity from descending into unproductive divergence [9].

We conducted extensive experiments across four models ranging from 1.5B to 14B to verify the effectiveness of our method. First, we tested it on mathematical datasets with different levels of difficulty including GSM8K, MATH 500, AMC 2023, and AIME 2025. The results show that our method achieves a token reduction while consistently improving model accuracy. Moreover, it also demonstrates strong cross-domain generalization capability in tasks such as LiveCodeBench (code generation) and Diamond GPQA (knowledge-based QA). Furthermore, we integrate the method into the vLLM engine and achieve a 24.9% to 38.4% reduction in end-to-end inference latency, highlighting its high efficiency for real-world deployment.

2 Related Work

Efforts to achieve efficient reasoning in LRMs have gained significant traction to enhance both inference efficiency and output quality. Among them, training-based methods typically use supervised fine-tuning to compress the reasoning process [10, 11] or use reinforcement learning to enable the model to adaptively generate chains of thought of different lengths [12, 13]. However, these methods incur significant computational costs and are orthogonal to inference-time interventions like ours, so we do not directly compare them in this work. Existing training-free methods include early exiting [5, 14, 15] and representation steering [8, 16, 17, 18, 19]. Specifically, representation steering dynamically adjusts the reasoning trajectory by directly intervening in the model's latent activation space during inference. Our work falls into this category. Existing representation steering methods are usually reactive and merely push the latent state away from problematic behaviors (e.g. overthinking), which inevitably risks over-correction and induces underthinking. Differently, our work explicitly considers *where the efficient reasoning state should actually reside*. We reveal the geometric concentration of efficient reasoning within the latent space and proactively project deviant hidden states back into this concentrated region via constrained optimization. More related works can be seen in Appendix A.

3 Problem Formulation and Our Objective

Given a question q, a large reasoning model, denoted as π, autoregressively generates a sequence of intermediate reasoning steps $c = [c_1, c_2, \ldots, c_N]$ before producing the final answer y. This process can be formally expressed as:

$$c_t \sim \pi(\cdot|c_{<t}, q), \quad y \sim \pi(\cdot|c, q) \tag{1}$$

where c_t is the reasoning step generated at time step t, and $c_{<t}$ represents all preceding steps.

Naturally, we are interested in the contribution of one individual reasoning step to the final outcome. To quantify this contribution, we evaluate its impact on the model's expected correctness and remaining generation length. Specifically, after observing a partial trajectory up to step t (denoted as $c_{\leq t}$), the expected probability of the model ultimately reaching the correct answer y^* can be written as $\mathbb{P}_{(c_{>t},y)\sim\pi(\cdot|c_{\leq t},q)}(y = y^*)$. Similarly, the expected number of remaining tokens required to complete the generation from this step is given by $\mathbb{E}_{(c_{>t},y)\sim\pi(\cdot|c_{\leq t},q)}\mathrm{Len}(c_{>t}, y)$.

By comparing these expected values before step c_t (conditioned on $c_{<t}$) and after step c_t (conditioned on $c_{\leq t}$), we can isolate the contribution of c_t. Thus, we introduce the concept of **progress**, a two-dimensional vector $p(c_t; c_{<t}, q) = [p_{\text{acc}}, p_{\text{len}}]$, which explicitly quantifies the impact of step c_t on both the final accuracy and the generation length:

$$\begin{aligned} p_{\text{acc}} &= \mathbb{P}_{(c_{>t},y)\sim\pi(\cdot|c_{\leq t},q)}(y = y^*) - \mathbb{P}_{(c_{\geq t},y)\sim\pi(\cdot|c_{<t},q)}(y = y^*) \\ p_{\text{len}} &= \mathbb{E}_{(c_{\geq t},y)\sim\pi(\cdot|c_{<t},q)}\text{Len}(c_{\geq t}, y) - \mathbb{E}_{(c_{>t},y)\sim\pi(\cdot|c_{\leq t},q)}\text{Len}(c_{>t}, y) - \text{Len}(c_t) \end{aligned} \tag{2}$$

where y^* is the ground-truth answer, and $\text{Len}(\cdot)$ denotes the token length.

In Eq. (2), p_{acc} represents the change in the probability of reaching the correct answer y^*, where a positive value indicates the step has moved the model closer to success. p_{len} measures the reduction in the total expected token length. A positive value implies that the step has shortened the overall path to the solution. This formulation naturally captures typical reasoning behaviors. For instance, a negative p_{acc} reflects a drop in correctness, which is often seen in cases where the reasoning is too shallow or terminates prematurely, i.e., *underthinking*. Similarly, a negative p_{len} means that the token cost of the step is greater than the amount it reduces from the remaining reasoning process. Such steps usually correspond to the behaviors in *overthinking* such as unnecessary computations or redundant double-checking, which inevitably increase the overall length of the reasoning process.

Empirical Example for Understanding Progress. To further illustrate the concept of progress, the following example presents a reasoning trajectory generated by DeepSeek-R1-Distill-Qwen-1.5B. For each reasoning step c_t, we sample the model 10 times conditioned on $c_{\leq t}$ and $c_{<t}$, then compute the differences in the accuracy and the expected remaining length as the estimation of progress. We find that steps with negative progress are often associated with self-correction loops, hesitation, or repetitive reformulation, which increase token cost without improving correctness. In contrast, steps with positive progress typically correspond to necessary computation or preliminary conclusions that move the solution forward.

Example 1: Empirical Example of Progress in CoT

QUESTION:

How many integers n satisfy the condition $100 < n < 200$ and the condition n has the same remainder whether it is divided by 6 or by 8?

PROGESS IN COT:

*(p = [0.0, +1322]) Okay, so I have this problem here: I need to find how many integers n satisfy two conditions. ... (p = [-0.1, -612]) Alright, so let's break it down. The first condition is straightforward: ... So, mathematically, this can be written as: $n \equiv r \pmod 6$ and $n \equiv r \pmod 8$ ... (p = [0.0, -620]) But wait, hold on. Let me check: if r is the remainder when n is divided by 6, then it's $0 \leq r < 6$. ... (p = [+0.1, +539]) So, for each r, we can find the number of k such that 24k + r is between 101 and 199. (p = [0.0, -941]) So, let's solve for k in each case.\n\n ... (p = [+0.1, +818]) But n=24*4 +5=96 +5=101, which is the lower bound. So, k=4,5,6,7,8, which is 5 values. ... (p = [-0.1, -827]) Wait, 6 is not equal to 0. So, that's a problem. Wait, so 102 is not included. ... (p = [-0.0, -669]) Alternatively, perhaps we can think in terms of the least common multiple.(p = [0.0, +15]) **Final Answer**The number of integers n that satisfy the given conditions is \\boxed{25}.*

Intuitively, an efficient reasoning step c_t should simultaneously improve accuracy and promote generation efficiency. We formalize this by defining the set $S(c_{<t}, q)$ as:

$$S(c_{<t}, q) := \{c_t \mid p(c_t; c_{<t}, y) \geq [0, 0]\} \tag{3}$$

Each step in the set $S(c_{<t}, q)$ is both beneficial for reaching the correct answer and efficient in terms of generation length. Consequently, our objective is to guide the LRM to favor steps within the set (3), thereby ensuring that the resultant reasoning trajectory is both accurate and concise.

4 Analysis of Reasoning Steps in Latent Space

The definition in Eq. (3) provides a conceptual formulation for efficient reasoning. However, explicitly constructing this set is computationally prohibitive. Evaluating the progress vector p requires extensive rollouts, incurring massive computational overhead. As a practical alternative, we seek an empirical proxy for this target set.

4.1 A Geometric Perspective on Efficient Reasoning

A natural way is to leverage LRM itself to generate efficient reasoning paths. Specifically, we prompt the model with the ground-truth answer and conciseness instructions (see Appendix F.2), encouraging it to produce compact deductions as final outputs. Then we discard the intermediate `<think>` block and retain only the finalized output after `</think>`. As observed in the specific example provided in Appendix B.2, the steps in the output typically exhibit positive progress and closely align with the definition of our target set $S(c_{<t}, q)$. We define such outputs as *Direct CoT*. For comparison, we also collect responses generated from the question alone, i.e., *Vanilla CoT*.

We conduct experiments on the MATH dataset [20] using DeepSeek-R1-Distill-Qwen-1.5B [2]. Specifically, we randomly sample 300 questions from the default training set, generating one Direct CoT and one Vanilla CoT for each question. During the inference process, we concatenate these CoTs with the prompt of Vanilla CoT and then extract the hidden representations corresponding to the step delimiters (e.g., "\n \n") across different model layers. We then apply Principal Component Analysis (PCA) to project these hidden representations into a low-dimensional latent space.

As shown in Figure 1, the representations of Direct CoTs form a concentrated region, whereas those of Vanilla CoTs are dispersed and encapsulate this inner region. Moreover, as shown in Figure 7, the Vanilla CoT representations falling outside this concentrated region likely correspond to redundant steps such as getting stuck in repetitive loops or redundant self-verification. Motivated by this geometric observation, we adopt the concentrated region formed by Direct CoTs as our empirical proxy for the target set in Eq. (3). For clarity, we refer to this region as the **efficient reasoning set**, denoted by $\mathcal{H}$. We also include additional visualizations for other models in Appendix C.1.

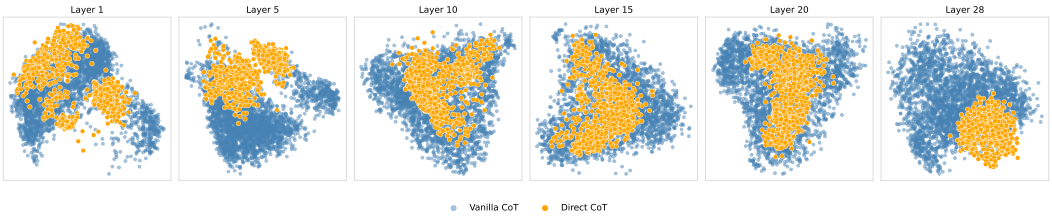


Figure 1: Visualization of different reasoning steps in the latent space. The horizontal and vertical axes correspond to the first and second PCA components, respectively.

4.2 Computation of the Efficient Reasoning Set $\mathcal{H}$

We now detail the computation method for the set $\mathcal{H}$. At a high level, we project the high-dimensional hidden representations into a lower-dimensional latent space, and subsequently compute boundary hyperplanes to enclose the target set $\mathcal{H}$. Specifically, let $\mathbf{X}_D$ and $\mathbf{X}_V$ denote the hidden representations of Direct and Vanilla CoTs, respectively. We first project these into a k-dimensional latent space via a PCA matrix $\mathbf{U} \in \mathbb{R}^{k \times d}$ ($k \ll d$), yielding $\mathbf{Z}_D = \mathbf{U}\mathbf{X}_D$ and $\mathbf{Z}_V = \mathbf{U}\mathbf{X}_V$. The intuition for constructing $\mathcal{H}$ is to maximize the inclusion of $\mathbf{Z}_D$ while maximally excluding $\mathbf{Z}_V$. Because the dispersed $\mathbf{Z}_V$ representations geometrically encapsulate the concentrated $\mathbf{Z}_D$ region, we aim to establish hyperplanes from multiple directions to enclose this target region.

To find these directions, we first apply K-Means to partition the $\mathbf{Z}_V$ representations into M distinct clusters $\{\mathbf{Z}_V^{(1)}, \ldots, \mathbf{Z}_V^{(M)}\}$. For each cluster $m \in \{1, \ldots, M\}$, we train a linear SVM to separate the target region $\mathbf{Z}_D$ from this specific cluster $\mathbf{Z}_V^{(m)}$. This procedure yields M initial half-spaces, each defined by a directional boundary $\mathbf{w}_m^\top \mathbf{z} + b_m \leq 0$. We then further relax each hyperplane outward

by adjusting its intercept as follows:

$$b'_m = -\max\left(P_\alpha\left(\{\mathbf{w}_m^\top \mathbf{z} \mid \mathbf{z} \in \mathbf{Z}_D\}\right),\ \mathbf{w}_m^\top \bar{\mathbf{z}}_D\right) \tag{4}$$

where $P_\alpha(\cdot)$ denotes the α-th percentile of the projected scalar values, and $\bar{\mathbf{z}}_D$ is the mean vector of the $\mathbf{Z}_D$. These two components serve different purposes. The first term allows the shifted boundary to retain at least an α proportion of the Direct CoT steps. The second term ensures that the boundary never excludes the center of the Direct CoT.

Finally, the efficient reasoning set $\mathcal{H}$ can be obtained through the intersection of the half-spaces bounded by these M adjusted hyperplanes:

$$\mathcal{H} = \left\{\mathbf{z} \in \mathbb{R}^k \mid \mathbf{w}_m^\top \mathbf{z} + b'_m \leq 0, \quad \forall m \in \{1, \ldots, M\}\right\} \tag{5}$$

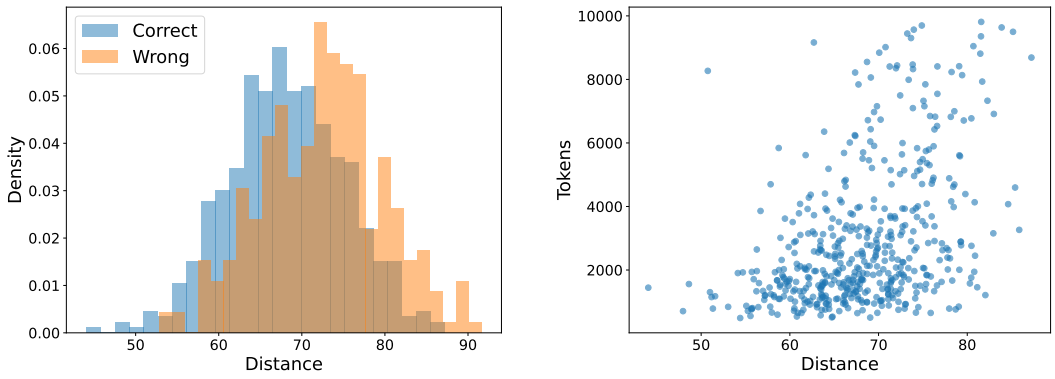
This formulation bounds the Direct CoT reasoning space into a convex polytope. Specially, because the mean vector $\bar{\mathbf{z}}_D$ satisfies all M linear constraints by design, it serves as a guaranteed feasible point, ensuring that the intersection $\mathcal{H}$ is strictly non-empty.

4.3 Reasoning Performance and Distance to $\mathcal{H}$

We further investigate what happens when steps in a reasoning sequence frequently fall outside the region $\mathcal{H}$. Intuitively, if $\mathcal{H}$ represents the efficient reasoning, then deviating from $\mathcal{H}$ could correspond to inefficient reasoning, such as generating redundant steps. Let a generated trajectory be denoted as $c = [c_1, \ldots, c_N]$, with z_i representing the latent representation of the step c_i. We define the distance from the reasoning trajectory to the set $\mathcal{H}$ as:

$$d(c, \mathcal{H}) = \frac{1}{N} \sum_{i=1}^{N} \min_{u \in \mathcal{H}} \|z_i - u\|_2 \tag{6}$$

Then, we compute the distance $d(c, \mathcal{H})$ of each previously collected *Vanilla CoT* trajectory and evaluate how it relates to reasoning performance. As illustrated in Figure 2, greater distance indicates worse reasoning outputs. Specifically, Figure 2(a) shows that correct reasoning trajectory generally stays close to the set $\mathcal{H}$, whereas incorrect ones drift much further away. Additionally, Figure 2(b) reveals a clear positive correlation (Spearman's $\rho = 0.4977$) between $d(c, \mathcal{H})$ and total token length, indicating that sequences further from the set $\mathcal{H}$ may be unnecessarily long. Collectively, these findings indicate that deviating from the set $\mathcal{H}$ corresponds to a higher risk of failure and computational overthinking.



(a) Reasoning Accuracy vs. Distance

(b) Total Token Length vs. Distance

Figure 2: (a) The distribution of distances to the set $\mathcal{H}$ for trajectories with correct and incorrect answers. (b) The relation between the distance $d(c, \mathcal{H})$ and the total token length of the reasoning trajectory.

5 Representation Steering via Constrained Optimization

Building upon the observation above, we propose a novel steering mechanism. Our goal is to actively correct the model's reasoning trajectory by steering its hidden states back into the efficient reasoning region bounded by $\mathcal{H}$.

Specifically, we apply the intervention exclusively to the step delimiter tokens "\n \n". Let $\mathbf{x}_t \in \mathbb{R}^d$ denote its hidden representation of such a token at a specific layer. We first project this representation into our k-dimensional latent space via PCA, obtaining $\mathbf{z}_t = \mathbf{U}\mathbf{x}_t$. If $\mathbf{z}_t$ falls outside the set $\mathcal{H}$, we seek the closest point within $\mathcal{H}$ to serve as the corrected latent state. Since $\mathcal{H}$ is a convex polytope defined by the intersection of M half-spaces, finding this closest point $\mathbf{z}_t^*$ naturally formulates as a constrained Quadratic Programming (QP) problem:

$$\mathbf{z}_t^* = \arg\min_{\mathbf{u}\in\mathbb{R}^k} \frac{1}{2}\|\mathbf{z}_t - \mathbf{u}\|_2^2 \quad \text{s.t.} \quad \mathbf{w}_m^\top \mathbf{u} + b'_m \leq 0, \quad \forall m \in \{1, \ldots, M\} \tag{7}$$

For simplicity, we denote this projection operation as $\mathbf{z}_t^* = \text{Proj}_{\mathcal{H}}(\mathbf{z}_t)$. As established in Section 4.2, the set $\mathcal{H}$ is guaranteed to be non-empty (containing at least the mean vector $\bar{\mathbf{z}}_{dir}$). Consequently, this QP problem is strictly feasible and admits a unique global optimum.

Once the corrected latent representation is computed, we map this correction back to the original high-dimensional space. Notably, simply replacing the original representation with the inverse PCA transform (i.e., $\mathbf{U}^\top \text{Proj}_{\mathcal{H}}(\mathbf{U}\mathbf{x}_t)$) would discard all information residing in the orthogonal complement of the PCA subspace, potentially destroying task-specific semantic details. Therefore, we preserve the orthogonal residual and only modifies the components within the principal subspace as follow:

$$\mathbf{x}_t \leftarrow \mathbf{x}_t + \lambda \mathbf{U}^\top \big(\text{Proj}_{\mathcal{H}}(\mathbf{U}\mathbf{x}_t) - \mathbf{U}\mathbf{x}_t\big) \tag{8}$$

where λ is a positive coefficient to control the intervention strength. Notably, we introduce λ into the formulation primarily to facilitate sensitivity analysis (see Section 6.4) rather than as a hyperparameter requiring tuning. We fix $\lambda = 1.0$ as the default setting.

Furthermore, the computational overhead of this intervention is marginal compared to the original forward pass. This stems from two factors. First, the intervention is infrequent, targeting only the few step delimiters within a sequence; Second, the QP projection is computationally efficient by construction, as it operates in a low-dimensional latent space with a limited number of constraints. In our implementation, the problem involves only 10–16 decision variables and 8 linear constraints.

GPU Acceleration for Solving QP. Despite its low mathematical complexity, QP solvers typically execute on the CPU, which introduces additional latency due to the frequent data transfer between the CPU and GPU. This could noticeably degrade the overall throughput, especially in a highly optimized LRM inference pipeline. To eliminate this bottleneck, we accelerate the optimization process by exploiting the structural properties of Multi-Parametric Quadratic Programming (mp-QP). Specifically, mp-QP theory establishes that the optimal solution $\mathbf{z}_t^*$ can be analytically expressed as a continuous piecewise linear function of the input parameter $\mathbf{z}_t$ [21]. Practically, this allows us to partition the continuous latent space into a finite set of predefined regions offline. During inference, solving the specific QP instance is reduced to querying the region index and applying its corresponding affine transformation matrix. We precompute and deploy these mappings directly on the GPU as tensor operations, eliminating the CPU-GPU communication overhead and ensuring high-throughput reasoning. A detailed introduction of the quadratic programming problem and the explicit acceleration via offline mp-QP are provided in Appendix E.

6 Experiments

6.1 Experimental Setups

Models and Datasets. We evaluate the effectiveness of our proposed method across six diverse benchmarks. These include four mathematical reasoning datasets: GSM8K [22], MATH500 [20], AMC 2023 [23], and AIME 2025 [24]; one scientific reasoning dataset: GPQA Diamond [25]; and one code reasoning dataset: LiveCodeBench [26]. Model performance is measured using Accuracy (**Acc**, ↑) and the average number of generated tokens (**Tok**, ↓). Given the limited number of samples in datasets AMC 2023 and AIME 2025, we conduct 16 sampling rounds per instance

Table 1: Performance comparison against baselines. For each column within a model group, the best result is **bolded** and the second-best is underlined.

Methods	MATH								PROGRAMMING		SCIENCE	
	GSM8K		MATH500		AMC2023		AIME2025		LiveCodeBench		GPQA-Diamond	
	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.
DeepSeek-R1-Distill-Qwen-1.5B												
Vanilla	81.5	1489	82.1	4476	64.4	7708	17.3	11811	32.1	10013	33.3	8241
CCoT	76.6	**577**	82.0	3832	68.8	7012	22.7	11077	30.6	9824	33.3	7592
DEER	74.2	672	72.2	**2578**	62.5	5139	20.2	9601	23.7	8227	29.7	7655
SEAL	82.2	924	82.4	3460	72.5	5382	23.5	9904	30.5	8111	33.8	6620
STIR	81.2	909	79.8	3363	70.0	6157	23.1	10572	31.7	9632	26.8	7027
ReBalance	81.0	895	83.0	3697	67.5	6292	24.4	10600	30.2	9196	22.7	9071
Ours	**83.1**	853	**83.2**	2949	**75.2**	**4821**	**26.7**	**8502**	**32.7**	**7656**	**36.4**	**5651**
Δ vs. Van.	(+1.6)	(-42.7%)	(+1.1)	(-34.1%)	(+10.8)	(-37.5%)	(+9.4)	(-28.0%)	(+0.6)	(-23.5%)	(+3.1)	(-31.4%)
DeepSeek-R1-Distill-Qwen-7B												
Vanilla	90.9	1129	91.6	3647	86.3	5846	26.9	11209	58.9	8320	48.9	7194
CCoT	89.4	**679**	91.8	3225	90.2	5522	33.8	10695	59.5	8077	50.5	7101
DEER	90.1	753	89.6	**2245**	85.0	4107	31.9	9442	54.2	7867	49.1	7166
SEAL	91.0	825	90.6	2838	85.8	4813	32.1	9422	57.1	7060	50.5	5456
STIR	90.6	730	88.2	2669	85.0	5244	36.5	10303	59.7	7739	46.9	6035
ReBalance	90.5	1016	92.1	3384	**92.5**	5075	33.5	10258	55.0	7749	**52.0**	6833
Ours	**91.3**	716	**92.2**	2599	91.1	**4083**	**39.0**	**9042**	**61.2**	**6788**	51.0	**4599**
Δ vs. Van.	(+0.4)	(-36.6%)	(+0.6)	(-28.7%)	(+4.8)	(-30.2%)	(+12.1)	(-19.3%)	(+2.3)	(-18.4%)	(+2.1)	(-36.1%)
Qwen3-4B												
Vanilla	94.9	2476	95.2	4587	95.9	7588	60.2	16835	87.7	7818	53.2	8080
CCoT	94.8	1237	95.4	3834	96.6	6518	63.8	16034	86.7	7160	53.5	6836
DEER	94.3	1331	94.8	3947	95.0	6204	62.1	15972	76.2	6993	53.0	7744
SEAL	95.0	1275	94.6	4015	90.8	6667	64.4	14944	87.0	**6559**	50.0	6739
STIR	94.4	1387	91.4	4075	92.5	6798	57.9	**12768**	84.0	7170	55.0	6792
ReBalance	95.3	1645	95.2	4137	97.0	6289	60.6	15008	85.5	7475	53.9	7038
Ours	**95.4**	**1168**	**95.8**	**3550**	**97.5**	**5898**	**67.1**	14809	**88.5**	6898	**57.5**	**6698**
Δ vs. Van.	(+0.5)	(-52.8%)	(+0.6)	(-22.6%)	(+1.6)	(-22.3%)	(+6.9)	(-12.0%)	(+0.8)	(-11.8%)	(+4.3)	(-17.1%)
Qwen3-14B												
Vanilla	95.6	1433	96.2	4559	96.9	7277	67.1	16019	90.5	7252	63.6	7323
CCoT	95.8	**969**	96.5	3623	97.2	5852	69.6	14349	90.1	6465	62.1	6530
DEER	96.0	1051	94.4	**3262**	94.8	6065	65.8	13983	85.3	**5687**	61.7	6971
SEAL	93.6	1181	95.4	3788	96.3	6193	61.7	14714	88.7	6411	62.9	6421
STIR	95.9	1197	92.4	3825	95.3	6295	67.7	13239	86.8	6680	62.6	6923
ReBalance	95.6	1740	95.4	4222	97.0	5999	**71.0**	15340	89.9	6312	63.1	6528
Ours	**96.4**	1011	**96.8**	3510	**98.3**	**5751**	70.4	**13144**	**91.0**	6210	**64.1**	**6309**
Δ vs. Van.	(+0.8)	(-29.4%)	(+0.6)	(-23.0%)	(+1.4)	(-21.0%)	(+3.3)	(-17.9%)	(+0.5)	(-14.4%)	(+0.5)	(-13.8%)

to ensure evaluation stability. We report the mean performance in the main text and the standard deviations in Appendix D.5. We evaluate our approach using widely adopted LRMs, including DeepSeek-R1-Distill-Qwen-1.5B/7B [2] and Qwen3-4B/14B [27].

Baselines. We compare our approach against six training-free baselines. **Vanilla** represents the direct evaluation of the model without any intervention. The remaining methods can be categorized as follows. For prompt-based methods, we compare against **CCoT** [28], which explicitly instructs the model to think concisely. For early-exit methods, we compare against **DEER** [5], which self-truncate CoT sequences by early exit during generation. Most importantly, because our proposed approach is closely related to steering-based methods, we select three representative methods from this category for a comprehensive comparison: **SEAL** [16], **STIR** [17], and **ReBalance** [8]. Specifically, SEAL computes a steering vector to calibrate the CoT process, guiding the model toward more concise reasoning. STIR formulates reasoning enhancement as a dynamic latent trajectory control problem, injecting context-specific steering vector via a library of self-distilled internal reasoning

tools. ReBalance utilizes real-time confidence to modulate a steering vector, pruning redundancy during overthinking and promoting exploration during underthinking.

Implementation Details. Our implementations are built upon a modified version of the Easysteer [29] library to integrate the projection operation, utilizing the vLLM [30] backend for high-throughput inference. For the QP solving, we precompute an mp-QP via the PDAQP solver [31]. During the inference phase, solving the QP is transformed into a region querying and affine mapping process, implemented using PyTorch operators. Detailed hyperparameter and decoding settings are provided in Appendix F. For baseline methods, we use their official implementations and the default settings. All experiments were conducted on a server equipped with eight 80GB NVIDIA A800 GPUs.

6.2 Main Results

Performance on Mathematical Reasoning. As shown in Table 1, our proposed method consistently outperforms the other baselines across all four mathematical benchmarks and various model scales, successfully reducing token counts while improving accuracy. We observe that baseline methods such as DEER and CCoT yield lower token counts on some datasets. For DEER, while it uses fewer tokens on MATH500 for the 1.5B model, its aggressive truncation causes a catastrophic 9.9% drop in accuracy, indicating severe underthinking that skips necessary logical deductions. Another baseline, CCoT, achieves lower token generation on simpler tasks like GSM8K. One reason is that GSM8K requires relatively shallow reasoning, allowing explicit instructions to compress the output. However, CCoT's performance degrades noticeably on more complex benchmarks.

Cross-Domain Generalization. To investigate the generalizability of our method beyond mathematical domains, we extended our evaluation to code generation (LiveCodeBench) and scientific knowledge reasoning (GPQA-Diamond). As detailed in Table 1, our method maintains its strong performance in these distinct domains. These results demonstrate that our method can generalize well to improve generation efficiency and accuracy across diverse reasoning tasks.

Further Supplementary Evaluations. Due to space limitations, we provide extensive supplementary experiments in Appendix D. First, we demonstrate the cross-domain and cross-difficulty transferability of our method in Table 3, showing that efficient reasoning sets derived from basic math or science domains can successfully generalize to complex competition-level math and coding tasks. Second, our semantic change analysis in Table 4 reveals that the proposed method effectively suppresses redundant self-verification (Reflection) and perspective-shifting (Transition) behaviors. To further investigate whether the proposed method introduces any unintended side effects on general capabilities, we evaluate our method on the Creative Writing v3 benchmark in Table 5, demonstrating that the model's creative expression are preserved. Moreover, we conduct a Pass@k performance analysis on AMC23 and AIME2025 to investigate whether our method affects the model's exploration capability. The results in Table 6 indicates that our approach maintains or improves the Pass@k performance.

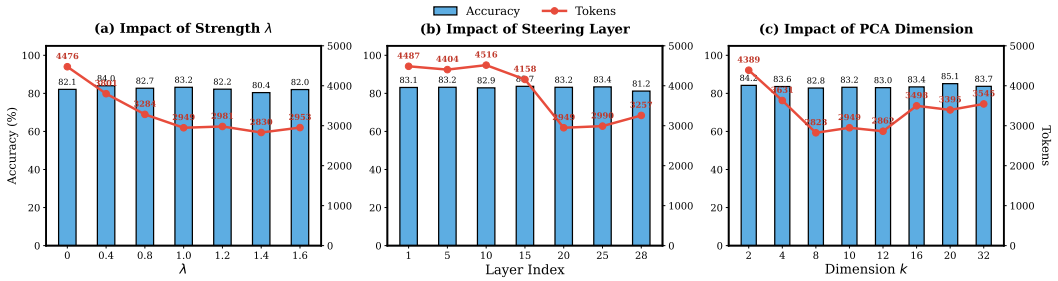


Figure 3: The impact of the key hyperparameters for performance in MATH500: (a) Intervention Strength. (b) Steering Layer. (c) PCA dimension.

6.3 Quantitative Evaluation of Efficiency

To validate the efficiency of our GPU-accelerated projection (**Explicit**), we compare it against an **Online** baseline using the widely adopted CvxpyLayer solver [32] to solve QP. We evaluate both settings across two dimensions: the computational overhead and the inference latency in vLLM.

Computational Overhead Analysis. Table 8 presents the detailed computational cost comparison for solving a single QP instance. As the results demonstrate, the computational overhead introduced by our **Explicit** method is exceptionally small. While the **Online** solver requires a few milliseconds, our GPU acceleration drastically reduces this solving time to less than a millisecond (e.g., 0.3 ms for the 1.5B model), requiring less than 2 MB of additional GPU memory across all evaluated models. A detailed analysis on the computational overhead is provided in Appendix E.4.

Table 2: Efficiency Analysis on A800 GPU.

Method	# Tokens	TPS ↑	TPR (s) ↓
DeepSeek-R1-Distill-Qwen-1.5B			
Baseline	4678	5596	0.83
Ours (Online)	3074	4682	0.66 (-21.2%)
Ours (Explicit)	2812	5472	0.51 (-38.4%)
DeepSeek-R1-Distill-Qwen-7B			
Baseline	3631	2659	1.36
Ours (Online)	2569	2262	1.14 (-16.2%)
Ours (Explicit)	2685	2638	1.02 (-25.0%)
Qwen3-4B			
Baseline	4593	1242	3.70
Ours (Online)	3608	1178	3.06 (-17.3%)
Ours (Explicit)	3504	1339	2.62 (-29.2%)
Qwen3-14B			
Baseline	4337	786	5.51
Ours (Online)	3520	719	4.88 (-11.4%)
Ours (Explicit)	3484	842	4.14 (-24.9%)

Efficiency Analysis. Table 2 presents the end-to-end efficiency results against native `vLLM` on MATH500 using NVIDIA A800 GPUs. We generate 3 responses per problem and evaluate the end-to-end inference latency using Time Per Request (TPR). Our method consistently achieves lower TPR across all models compared to the baseline, an acceleration primarily driven by the drastic reduction in generated tokens. Notably, our **Explicit** version achieves a Tokens Per Second (TPS) rate highly comparable to the native baseline. These results strongly demonstrate the practicability of our approach for high-throughput, real-world deployment. The slight token variations between **Ours (Online)** and **Ours (Explicit)** stem from stochastic decoding.

6.4 Ablation Study

In this subsection, we conduct ablation studies on the MATH500 dataset using the DeepSeek-R1-Distill-Qwen-1.5B model to evaluate the impact of the key hyperparameters in our methods. See Appendix G for more details about ablation study of other models.

Impact of Intervention Strength λ. Figure 3(a) shows that λ in $[1.0, 1.6]$ yield substantial token reduction. However, we observe a slight trade-off where increasing λ leads to a marginal decrease in accuracy. This means that the default choice of $\lambda = 1.0$ achieves an appropriate balance.

Impact of Steering Layer. Figure 3(b) illustrates the sensitivity of steering performance to the chosen intervention layer. Our results indicate that middle-to-deep layers yield the better performance. This observation can be explained by the typical forward pass dynamics of LLMs. Intervening at shallow layers is ineffective because these layers primarily process token-level representations rather than abstract reasoning features. Conversely, corrections at the final layers yield diminishing returns, as these representations are to summarize the context for next-token prediction.

Impact of PCA Dimension k. Figure 3(c) demonstrates the effect of the latent space dimension k. Low dimensions lead to information loss, making it difficult to construct a useful set $\mathcal{H}$. In contrast, very high dimensions introduce unnecessary noise that hinders the effectiveness of the projection. The performance remains stable and efficient when k is within the range of $[8, 16]$.

Impact of Cluster Number M. Figure 12 shows that utilizing a small number of clusters (e.g., $M \leq 4$) yields marginal token reduction. This limited efficiency gain occurs because fewer linear constraints form an inaccurate and coarse convex polytope as $\mathcal{H}$. As M increases, the efficient reasoning set boundary becomes more precise, resulting in stabilized token reduction and stronger efficiency gains. However, a larger M introduces more constraints to the QP problem, which correspondingly increases the solving time during inference.

Impact of Boundary Proportion α. As illustrated in Figure 11, adjusting α within the range of 20% to 100% consistently achieves notable token reduction. However, smaller values of α lead to a slight degradation in model accuracy. This performance drop occurs because a low α creates a highly restrictive set, causing aggressive interventions that may inadvertently discard useful semantic variations. Conversely, when $\alpha = 100\%$, the token reduction becomes less pronounced. This is likely because a fully inclusive set $\mathcal{H}$ may also encompass "overthinking" behaviors.

7 Conclusion

In this work, we investigated the phenomenon of overthinking and underthinking in LRMs from a geometric perspective. By analyzing the reasoning steps in latent space, we identified that efficient reasoning steps naturally form a concentrated region, while those deviating from this region tend to produce verbose sequence. Based on this observation, we proposed a training-free method that utilizes quadratic programming to steer the model's hidden states back into the region during the inference. Our empirical results demonstrate that the proposed method can improve reasoning performance while reducing token generation costs.

Acknowledgments and Disclosure of Funding

This work was supported by National Natural Science Foundation of China (62325305), and the BNRist project (No. BNR2024TD03003). We thank Yee Hin Chong from Tsinghua University for supporting this work with GPU resources.

References

[1] Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

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

[3] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[4] Yuyang Wu, Yifei Wang, Ziyu Ye, Tianqi Du, Stefanie Jegelka, and Yisen Wang. When more is less: Understanding chain-of-thought length in llms. *arXiv preprint arXiv:2502.07266*, 2025.

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

[6] Shijue Huang, Hongru Wang, Wanjun Zhong, Zhaochen Su, Jiazhan Feng, Bowen Cao, and Yi R Fung. Adactrl: Towards adaptive and controllable reasoning via difficulty-aware budgeting. *arXiv preprint arXiv:2505.18822*, 2025.

[7] Zeju Li, Jianyuan Zhong, Ziyang Zheng, Xiangyu Wen, Zhijian Xu, Yingying Cheng, Fan Zhang, and Qiang Xu. Making slow thinking faster: Compressing llm chain-of-thought via step entropy. In *The Fourteenth International Conference on Learning Representations*, 2026.

[8] Yulin Li, Tengyao Tu, Li Ding, Junjie Wang, Hui-Ling Zhen, Yixin Chen, Yong Li, and Zhuotao Tian. Efficient reasoning with balanced thinking. In *The Fourteenth International Conference on Learning Representations*, 2026.

[9] Ofer Yizhar, Lief E Fenno, Matthias Prigge, Franziska Schneider, Thomas J Davidson, Daniel J O'shea, Vikaas S Sohal, Inbal Goshen, Joel Finkelstein, Jeanne T Paz, et al. Neocortical excitation/inhibition balance in information processing and social dysfunction. *Nature*, 477(7363):171–178, 2011.

[10] Hang Yuan, Bin Yu, Haotian Li, Shijun Yang, Christina Dan Wang, Zhou Yu, Xueyin Xu, Weizhen Qi, and Kai Chen. Not all tokens are what you need in thinking. *arXiv preprint arXiv:2505.17827*, 2025.

[11] Bin Yu, Hang Yuan, Haotian Li, Xueyin Xu, Yuliang Wei, Bailing Wang, Weizhen Qi, and Kai Chen. Long-short chain-of-thought mixture supervised fine-tuning eliciting efficient reasoning in large language models. *arXiv preprint arXiv:2505.03469*, 2025.

[12] Feng Luo, Yu-Neng Chuang, Guanchu Wang, Hoang Anh Duy Le, Shaochen Zhong, Hongyi Liu, Jiayi Yuan, Yang Sui, Vladimir Braverman, Vipin Chaudhary, et al. Autol2s: Auto long-short reasoning for efficient large language models. *arXiv preprint arXiv:2505.22662*, 2025.

[13] Lingjie Jiang, Xun Wu, Shaohan Huang, Qingxiu Dong, Zewen Chi, Li Dong, Xingxing Zhang, Tengchao Lv, Lei Cui, and Furu Wei. Think only when you need with large hybrid-reasoning models. *arXiv preprint arXiv:2505.14631*, 2025.

[14] Anqi Zhang, Yulin Chen, Jane Pan, Chen Zhao, Aurojit Panda, Jinyang Li, and He He. Reasoning models know when they're right: Probing hidden states for self-verification. *arXiv preprint arXiv:2504.05419*, 2025.

[15] Ziqing Qiao, Yongheng Deng, Jiali Zeng, Dong Wang, Lai Wei, Guanbo Wang, Fandong Meng, Jie Zhou, Ju Ren, and Yaoxue Zhang. Concise: Confidence-guided compression in step-by-step efficient reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 8021–8040, 2025.

[16] Runjin Chen, Zhenyu Zhang, Junyuan Hong, Souvik Kundu, and Zhangyang Wang. Seal: Steerable reasoning calibration of large language models for free. In *Second Conference on Language Modeling*, 2025.

[17] Zhenning Shi, Yijia Zhu, Junhan Shi, Xun Zhang, Lei Wang, and Congcong Miao. Internalizing llm reasoning via discovery and replay of latent actions. *arXiv preprint arXiv:2602.04925*, 2026.

[18] Yao Huang, Huanran Chen, Shouwei Ruan, Yichi Zhang, Xingxing Wei, and Yinpeng Dong. Mitigating overthinking in large reasoning models via manifold steering. *arXiv preprint arXiv:2505.22411*, 2025.

[19] Tuc Nguyen and Thai Le. Atlas: Adaptive test-time latent steering with external verifiers for enhancing llms reasoning. *arXiv preprint arXiv:2601.03093*, 2026.

[20] 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. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[21] Alberto Bemporad, Manfred Morari, Vivek Dua, and Efstratios N Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3–20, 2002.

[22] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[23] Mathematical Association of America. American mathematics competitions (AMC) 10/12. Problems and Answer Keys, 2023.

[24] Mathematical Association of America. American invitational mathematics examination (AIME). Problems and Answer Keys, 2025.

[25] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. In *First conference on language modeling*, 2024.

[26] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2025.

[27] Qwen Team. Qwen3 technical report, 2025.

[28] Matthew Renze and Erhan Guven. The benefits of a concise chain of thought on problem-solving in large language models. In *2024 2nd International Conference on Foundation and Large Language Models (FLLM)*, pages 476–483. IEEE, 2024.

[29] Haolei Xu, Xinyu Mei, Yuchen Yan, Rui Zhou, Wenqi Zhang, Weiming Lu, Yueting Zhuang, and Yongliang Shen. Easysteer: A unified framework for high-performance and extensible llm steering. *arXiv preprint arXiv:2509.25175*, 2025.

[30] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626, 2023.

[31] Daniel Arnström and Daniel Axehill. A high-performant multi-parametric quadratic programming solver. In *2024 IEEE 63rd Conference on Decision and Control (CDC)*, pages 303–308, 2024.

[32] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and Z. Kolter. Differentiable convex optimization layers. In *Advances in Neural Information Processing Systems*, 2019.

[33] Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason E Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. In *Second Conference on Language Modeling*.

[34] Oscar Skean, Md Rifat Arefin, Dan Zhao, Niket Nikul Patel, Jalal Naghiyev, Yann Lecun, and Ravid Shwartz-Ziv. Layer by layer: Uncovering hidden representations in language models. In *International Conference on Machine Learning*, pages 55854–55875. PMLR, 2025.

[35] Jihoon Tack, Jack Lanchantin, Jane Yu, Andrew Cohen, Ilia Kulikov, Janice Lan, Shibo Hao, Yuandong Tian, Jason E Weston, and Xian Li. Llm pretraining with continuous concepts. In *Mechanistic Interpretability Workshop at NeurIPS 2025*.

[36] Halil Alperen Gozeten, M Emrullah Ildiz, Xuechen Zhang, Hrayr Harutyunyan, Ankit Singh Rawat, and Samet Oymak. Continuous chain of thought enables parallel exploration and reasoning. *arXiv preprint arXiv:2505.23648*, 2025.

[37] Sohee Yang, Elena Gribovskaya, Nora Kassner, Mor Geva, and Sebastian Riedel. Do large language models latently perform multi-hop reasoning? In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10210–10229, 2024.

[38] Zhen Zhang, Xuehai He, Weixiang Yan, Ao Shen, Chenyang Zhao, Shuohang Wang, Yelong Shen, and Xin Eric Wang. Soft thinking: Unlocking the reasoning potential of llms in continuous concept space. *arXiv preprint arXiv:2505.15778*, 2025.

[39] Li Li, Ziyi Wang, Yongliang Wu, Jianfei Cai, and Xu Yang. Cot vectors: Transferring and probing the reasoning mechanisms of llms. *arXiv preprint arXiv:2510.00579*, 2025.

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

[41] Chenlong Wang, Yuanning Feng, Dongping Chen, Zhaoyang Chu, Ranjay Krishna, and Tianyi Zhou. Wait, we don't need to" wait"! removing thinking tokens improves reasoning efficiency.

[42] Zhengkai Lin, Zhihang Fu, Ze Chen, Chao Chen, Liang Xie, Wenxiao Wang, Deng Cai, Zheng Wang, and Jieping Ye. Controlling thinking speed in reasoning models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

[43] Samuel J Paech. Eq-bench creative writing benchmark v3. `https://github.com/EQ-bench/creative-writing-bench`, 2025.

[44] Aayush Karan and Yilun Du. Reasoning with sampling: Your base model is smarter than you think. *arXiv preprint arXiv:2510.14901*, 2025.

[45] Daniel Arnström, Alberto Bemporad, and Daniel Axehill. A dual active-set solver for embedded quadratic programming using recursive LDL^T updates. *IEEE Transactions on Automatic Control*, 67(8):4362–4369, 2022.

[46] Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd. OSQP: An operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4):637–672, 2020.

[47] Zhu Liu, Cunliang Kong, Ying Liu, and Maosong Sun. Fantastic semantics and where to find them: Investigating which layers of generative llms reflect lexical semantics. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 14551–14558, 2024.

Appendix Table of Contents

A Related Work Details **15**

B More Discussions about Progress **15**

B.1 Property of Progress 15

B.2 Progress of Direct CoT and Vanilla CoT 16

C More Analysis in Pilot Experiment **18**

C.1 Visualization of Reasoning Steps under Different Models 18

C.2 The Further Analysis of Visualization in Latent Space 18

D Additional Experimental Results **19**

D.1 Cross-Domain and Cross-Difficulty Transferability 19

D.2 Analysis of Semantic Change 20

D.3 Creativity Analysis 21

D.4 Pass@k Performance Analysis 21

D.5 Statistical Reliability and Significance Testing 22

E Details on Solving Quadratic Programming **23**

E.1 The Definition of Quadratic Programming 23

E.2 Online Numerical Solving Approach 23

E.3 Explicit Acceleration via Offline mp-QP 24

E.4 Computational Overhead Analysis 24

F Details on Experiment Settings **24**

F.1 Notation Summary 25

F.2 Prompt Structure 25

F.3 Hyperparameter and Decoding Settings. 26

G Details on Ablation Study **26**

G.1 Impact of Intervention Strength λ 26

G.2 Impact of Steering Layer 27

G.3 Impact of PCA Dimension k 27

G.4 Impact of Boundary Proportion α 27

G.5 Impact of Cluster Number M 27

H Limitations **28**

I Broader Impacts **28**

J Case Study **29**

A Related Work Details

Implicit Reasoning in Latent Space. Recent studies reveal that large language models inherently perform latent reasoning within their hidden computations [33, 34, 35]. For instance, empirical investigations show that multiple parallel latent reasoning paths can coexist within the middle layers of transformer architectures [36], and that intermediate entities in multi-hop reasoning tasks can be directly recovered from these latent representations [37]. Collectively, these observations have catalyzed a surge of research into implicit efficient reasoning within the latent space. To harness this capability, "soft-thinking" approaches replace discrete thinking tokens with continuous concept tokens [38]. Another paradigm directly trains model to reason in a continuous latent space [33]. Additionally, several studies focus on compressing long, explicit CoT trajectories into dense vectors to reduce inference overhead [39, 40]. To further explore efficient reasoning in the latent space, our study reveals that efficient reasoning steps form a concentrated region in the latent space. Motivated by this, we formulate a quadratic program to keep the hidden state of the model focus within this region, implicitly guiding the model to generate more effective reasoning steps.

Mitigating Overthinking in LLMs. To alleviate the excessive computational burden of overthinking, recent post-training approaches aim to reshape the model's reliance on lengthy reasoning. These include supervised fine-tuning methods that utilize compressed CoT data [10, 11], as well as reinforcement learning-based methods that optimize the accuracy-efficiency trade-off through reward shaping [12, 13]. While empirically effective, these methods incur significant retraining costs.

Consequently, training-free mechanisms have gained substantial traction. One paradigm is *Early Exiting*, which halts the reasoning process once sufficient evidence is gathered, relying on either external monitors [14] or internal confidence signals [5, 15]. However, these termination decisions are often coarse-grained and risk discarding potentially valuable reasoning steps, inevitably inducing underthinking. Another promising training-free paradigm is *Representation Steering*. This approach intervenes directly within the model's latent activation space during inference to steer the reasoning trajectory and suppress redundant loops [18, 16]. Our work naturally falls into this category.

Efficient Reasoning via Representation Steering. Representation steering has attracted growing interest for enhancing reasoning efficiency, motivated by its training-free nature and mechanistic interpretability [18, 16, 19]. These methods first identify linear directions within the activation space that correspond to specific model behaviors. By steering these linear directions during inference, they can calibrate model output, such as suppressing redundant loops and meaningless verbosity to mitigate overthinking. However, steering techniques typically apply a static, unidirectional intervention, which may not capture the highly dynamic nature of a reasoning trajectory [17]. To address this, recent studies have developed adaptive mechanisms. For example, ReBalance adjusts steering direction and strength based on real-time confidence assessments [8], while STIR operates as a dynamic controller that dictates intervention timing through latent trajectory monitoring [17].

While empirically effective, these approaches above are reactive: they focus on pushing the latent state away from problematic behaviors, lacking a principled consideration of *where the efficient reasoning state should actually reside.* This distinction is particularly critical in the context of efficient reasoning; if the intervention merely repels the state from overthinking without a defined target, it inevitably risks over-correction and induces underthinking. To address this gap, our work reveals the geometric concentration of efficient reasoning within the latent space. We explicitly locate this concentrated region and formulate representation steering as a constrained optimization problem, directly projecting deviant hidden states into this region for correction.

B More Discussions about Progress

B.1 Property of Progress

In this subsection, we investigate the mathematical relationship between the step-wise progress and the performance (accuracy and sequence length) of the entire reasoning trajectory.

Property B.1 (Cumulative Progress)**.** *Given a question q and a generated reasoning trajectory $c = [c_1, c_2, \ldots, c_N]$, the cumulative sum of step-wise progress determines the performance of final*

generation relative to the baseline. Specifically:

$$\mathbb{P}_{y\sim\pi(\cdot|c,q)}(y=y^*) = \mathbb{P}_{(c,y)\sim\pi(\cdot|q)}(y=y^*) + \sum_{t=1}^{N} p_{acc}(c_t; c_{<t}, q) \tag{9}$$

$$\sum_{t=1}^{N} Len(c_t) + \mathbb{E}_{y\sim\pi(\cdot|c,q)}[Len(y)] = \mathbb{E}_{(c,y)\sim\pi(\cdot|q)}[Len(c,y)] - \sum_{t=1}^{N} p_{len}(c_t; c_{<t}, q) \tag{10}$$

Proof. **Proof for Eq. (9).** Let $P_t = \mathbb{P}_{(c_{>t},y)\sim\pi(\cdot|c_{\leq t},q)}(y = y^*)$ denote the expected accuracy after observing the partial trajectory up to step t. By definition, the step-wise accuracy progress is $p_{\text{acc}}(c_t) = P_t - P_{t-1}$. Summing this progress over all N steps yields a telescoping series:

$$\sum_{t=1}^{N} p_{\text{acc}}(c_t) = \sum_{t=1}^{N} (P_t - P_{t-1}) = P_N - P_0$$

Note the boundary conditions: $P_0 = \mathbb{P}_{(c,y)\sim\pi(\cdot|q)}(y = y^*)$ corresponds to the baseline expected accuracy before any reasoning steps are generated, and $P_N = \mathbb{P}_{y\sim\pi(\cdot|c,q)}(y = y^*)$ represents the realized correctness probability given the full generated trajectory $c = c_{\leq N}$. Rearranging the equation $P_N - P_0 = \sum_{t=1}^{N} p_{\text{acc}}(c_t)$ directly yields Eq. (9).

Proof for Eq. (10). Similarly, let $L_t = \mathbb{E}_{(c_{>t},y)\sim\pi(\cdot|c_{\leq t},q)}[Len(c_{>t}, y)]$ denote the expected remaining sequence length after generating step t. The step-wise progress for length is defined as $p_{\text{len}}(c_t) = L_{t-1} - L_t - Len(c_t)$. Summing $p_{\text{len}}(c_t)$ from $t = 1$ to N gives:

$$\sum_{t=1}^{N} p_{\text{len}}(c_t) = \sum_{t=1}^{N} \Big(L_{t-1} - L_t - Len(c_t) \Big) = (L_0 - L_N) - \sum_{t=1}^{N} Len(c_t)$$

By examining the boundary conditions, we have $L_0 = \mathbb{E}_{(c,y)\sim\pi(\cdot|q)}[Len(c, y)]$, which is the total expected length of a reasoning trajectory from the initial question q. Meanwhile, $L_N = \mathbb{E}_{y\sim\pi(\cdot|c,q)}[Len(y)]$ is the expected length of the final answer y after the complete reasoning trajectory c has been generated. Substituting L_0 and L_N into the equation and rearranging the terms algebraically gives:

$$\sum_{t=1}^{N} Len(c_t) + L_N = L_0 - \sum_{t=1}^{N} p_{\text{len}}(c_t)$$

which exactly matches Eq. (10). □

Eq. (9) reveals that the final accuracy of solving the problem is the baseline probability plus the sum of all step-wise accuracy increments ($\sum p_{\text{acc}}$). This means that as long as the model ensures $p_{\text{acc}} > 0$ at each step, it continuously strengthens the correct probability, preventing the model from falling into shallow reasoning or underthinking. Similarly, Eq. (10) demonstrates that the total token cost of the current reasoning path equals the baseline expected length minus the accumulated length reductions ($\sum p_{\text{len}}$). In other words, every step with $p_{\text{len}} > 0$ acts as a "shortcut", bypassing redundant tokens and effectively suppressing verbosity and overthinking. Collectively, this property guarantees that guiding the model to generate steps with positive progress ($p_{\text{acc}} > 0$ and $p_{\text{len}} > 0$) is beneficial.

B.2 Progress of Direct CoT and Vanilla CoT

To further understand the concept of *Progress*, we analyze the empirical progress of the specific reasoning trajectories provided in Example 2 (Vanilla CoT) and Example 3 (Direct CoT).

The observation from Example 2 We observe obvious fluctuations in progress where several steps exhibit sharply negative progress values. These negative signals typically correspond to moments of model hesitation, redundant self-correction, or repetitive "double-checking" (e.g., *"Wait, let me think again,"*). Notably, some self-correction steps increase the token consumption but fail to meaningfully improve the probability of generating the correct answer. This empirical observation aligns with recent literature [41], which demonstrates that explicit self-reflection may degrade performance by generating redundant outputs rather than clarifying logic.

Example 2: Empirical Example of Progress in Vanilla CoT

QUESTION:

Let A equal the number of four digit odd numbers. Let B equal the number of four digit multiples of 5. Find $A + B$.

PROGESS IN COT:

*(p = [0.0, -618]) Alright, so I need to figure out A + B, where A is the number of four-digit odd numbers and B is the number of four-digit multiples of 5. Hmm, okay, let's break this down step by step. (p = [+0.1, +1000]) First, let's tackle A, the number of four-digit odd numbers. A four-digit number ranges from 1000 to 9999. So, I need to count how many numbers in this range are odd. An odd number is one that ends with 1, 3, 5, 7, or 9. That gives me five possibilities for the last digit. (p = [-0.2, -108]) Now, to find the total number of four-digit numbers, it's 9000 because from 1000 to 9999, that's 9999 - 1000 + 1 = 9000 numbers. But since we're only considering odd numbers, I need to figure out how many of these 9000 are odd. (p = [+0.2, -1]) Since the last digit is what determines if a number is odd, I can calculate the number of four-digit numbers ending with 1, 3, 5, 7, or 9. Let me think: for the thousands place, it can't be zero, so it ranges from 1 to 9. That's 9 options. For the hundreds place, it can be any digit from 0 to 9, so that's 10 options. The tens place is also any digit from 0 to 9, another 10 options. And the units place, as we established, has 5 options (1,3,5,7,9). (p = [-0.1, +104]) So, the total number of four-digit odd numbers should be 9 (thousands) * 10 (hundreds) * 10 (tens) * 5 (units). Let me compute that: 9 * 10 is 90, 90 * 10 is 900, and 900 * 5 is 4500. So, A = 4500. (p = [+0.1, -1248]) Wait, that seems a lot. Let me check. From 1000 to 9999, half of them should be odd, right? Since every other number is odd, so 9000 / 2 = 4500. Yeah, that makes sense. So, A is indeed 4500. (p = [-0.2, +1835]) Now, moving on to B, the number of four-digit multiples of 5. A multiple of 5 ends with either 0 or 5. So, similar to before, let's figure out how many four-digit numbers end with 0 or 5. (p = [+0.2, -844]) Again, the thousands place is from 1 to 9 (9 options), hundreds place is 0-9 (10 options), tens place is 0-9 (10 options). The units place is 0 or 5, so 2 options. (p = [-0.1, +496]) Therefore, the total number of four-digit multiples of 5 is 9 * 10 * 10 * 2. Let's compute that: 9 * 10 is 90, 90 * 10 is 900, 900 * 2 is 1800. So, B = 1800. (p = [0.0, +228]) Wait, let me think again. Since every fifth number is a multiple of 5, and four-digit numbers go from 1000 to 9999, how many multiples of 5 are there? Well, 9999 divided by 5 is approximately 1999.8, but since we're dealing with four-digit numbers, it's from 1000 to 9999 inclusive. (p = [0.0, -837]) Wait, the number of multiples of 5 in a range can be calculated by ((last - first)/step) + 1. So, ((9999 - 1000)/5) + 1. Let's compute that: (8999)/5 = 1799.8, which isn't an integer. Hmm, that might be a problem. (p = [0.0, +175]) Wait, no, actually, the number of multiples of 5 between 1000 and 9999 inclusive is equal to floor(9999/5) - floor(999/5). Let me compute floor(9999/5): 9999 /5 = 1999.8, so floor is 1999. floor(999/5) is floor(199.8) = 199. So, 1999 - 199 = 1800. So, that confirms B is 1800. (p = [+0.1, +534]) So, A is 4500 and B is 1800. Therefore, A + B is 4500 + 1800 = 6300. (p = [0.0, +1201]) Wait a second, just to make sure I didn't make any mistakes here. Let me think about another approach for B. Since the number must be a multiple of 5, it must end with 0 or 5. So, as I did earlier, thousands place 1-9 (9), hundreds 0-9 (10), tens 0-9 (10), units 0 or 5 (2). So, 91010*2=1800. (p = [0.0, +294]) And A is 4500, which is exactly half of 9000, which is the total number of four-digit numbers. So, that makes sense because every other number is odd, so half are odd. (p = [0.0, -397]) Therefore, adding them together, 4500 + 1800 equals 6300. So, A + B should be 6300. (p = [0.0, +503]) But wait, hold on, just to be thorough, let's double-check the number of four-digit numbers. 9999 - 1000 + 1 is indeed 9000, so that's correct. Half of 9000 is 4500, so A is 4500. For B, as we calculated, 1800. So, 4500 + 1800 is 6300. (p = [0.0, +276]) Yeah, that seems consistent. I think that's the answer. So, A + B is 6300. (p = [0.0, +4]) Final Answer The value of $A + B$ is $\boxed{6300}$.*

The observation from Example 3 Conversely, the Direct CoT trajectory presents a different progress profile. Most of the progress values are positive, indicating a consistently advancement toward the solution. This demonstrates that extracting Direct CoT trajectories can yield a efficient reasoning step set. Interestingly, we also observe a phenomenon at the very first step of the Direct CoT sequence

where the initial token triggers a massive progress gain ($p = [+0.0, +2530.7]$). This observation also aligns with the recent empirical findings of [42]. They observed that the specific leading words of the thought process (e.g., starting with "To" versus "Okay") dictate the verbosity of the entire subsequent output, demonstrating that forcing an efficient initial word substantially compresses the reasoning length. By extracting Direct CoT representations, our method implicitly leverages this powerful initial token to compress the output length.

Example 3: Empirical Example of Progress in Direct CoT

QUESTION:

Let A equal the number of four digit odd numbers. Let B equal the number of four digit multiples of 5. Find $A + B$.

PROGESS IN COT:

(p = [+0.0, 2530.7]) To find $A + B$, where A is the number of four-digit odd numbers and B is the number of four-digit multiples of 5, we proceed as follows: (p = [+0.1, +83.3])1. Calculating A (number of four-digit odd numbers): - A four-digit number ranges from 1000 to 9999. - The units digit must be an odd number (1, 3, 5, 7, 9), giving 5 choices. - The thousands digit can be any digit from 1 to 9, giving 9 choices. - The hundreds and tens digits can each be any digit from 0 to 9, giving 10 choices each. - Therefore, the total number of four-digit odd numbers is:$A = 9 \times 10 \times 10 \times 5 = 4500$ (p = [+0.0, +154]) 2. Calculating B (number of four-digit multiples of 5): - A number is a multiple of 5 if its units digit is 0 or 5, giving 2 choices. - The thousands digit can be any digit from 1 to 9, giving 9 choices. - The hundreds and tens digits can each be any digit from 0 to 9, giving 10 choices each. - Therefore, the total number of four-digit multiples of 5 is:$B = 9 \times 10 \times 10 \times 2 = 1800$ (p = [+0.0, +30]) 3. Adding A and B: - $A + B = 4500 + 1800 = 6300$ (p = [0.0, +115]) Thus, the value of $A + B$ is $\boxed{6300}$.

C More Analysis in Pilot Experiment

C.1 Visualization of Reasoning Steps under Different Models

To verify the consistency of our geometric observation across different models, we extended the latent space visualization analysis described in Section 4 to DeepSeek-R1-Distill-Qwen-7B and Qwen3-4B/14B. As illustrated in Figure 4, 5 and 6, the experimental results across different models reveal a consistent phenomenon: the hidden representations of Direct CoTs naturally form a concentrated core region, whereas those of Vanilla CoTs are more dispersed and encapsulate this region. This reinforces our motivation to define this concentrated region as the proxy for the target set in Eq. (3).

C.2 The Further Analysis of Visualization in Latent Space

We further explored the latent representations of the DeepSeek-R1-Distill-Qwen-1.5B model by examining higher-dimensional projections within the $k = 10$ dimensional space. As illustrated in Figure 7, the concentration of Direct CoT steps relative to the scattered region of Vanilla CoT steps is not limited to the first two components but persists across multiple dimensions. This consistent geometric pattern reinforces our approach of adopting the core region formed by Direct CoT as an empirical proxy for the target set in Eq. (3).

To further understand how overthinking patterns deviate from this core region, we analyzed two specific behaviors typically observed in overthinking: Transition and Reflection steps, following the definitions in SEAL [16]. A Reflection step corresponds to a moment where the model pauses to verify its preceding logic, while a Transition step involves shifting the reasoning flow to explore a different perspective. Our analysis reveals that Direct CoT steps are clearly separable from these behaviors in the latent space. This clear separability illustrates that when hidden states deviate away from the efficient reasoning set, the model is likely engaging in unproductive hesitation or redundant self-correction.

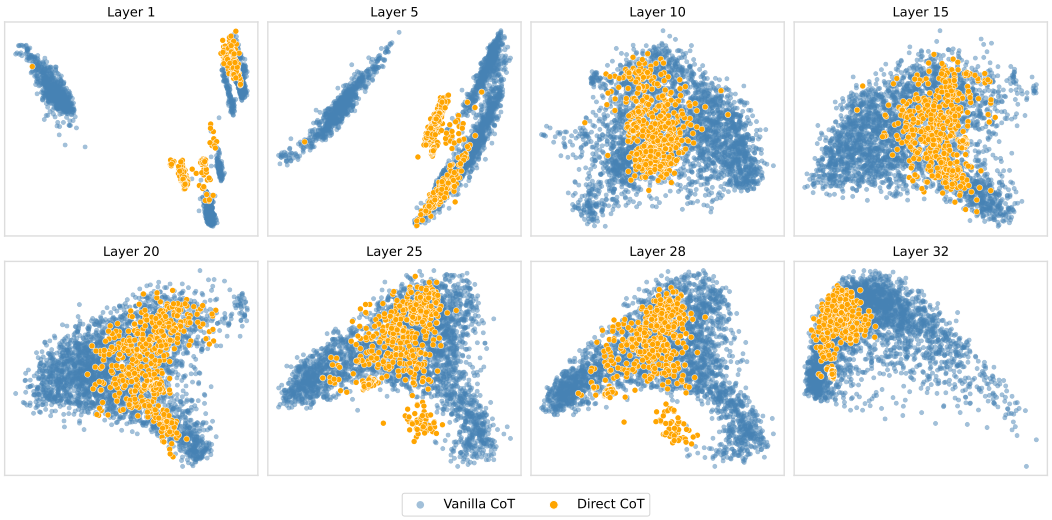


Figure 4: Visualization of different reasoning steps generated by Qwen3-4B. The horizontal and vertical axes correspond to the first and second PCA components, respectively.

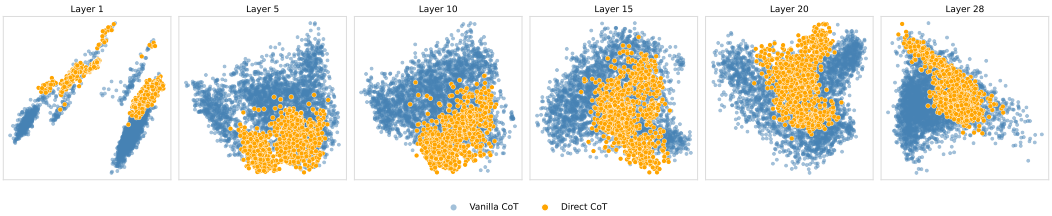


Figure 5: Visualization of different reasoning steps generated by DeepSeek-R1-Distill-Qwen-7B. The horizontal and vertical axes correspond to the first and second PCA components, respectively.

D Additional Experimental Results

D.1 Cross-Domain and Cross-Difficulty Transferability

To investigate the transferability of our method, we conduct experiments on DeepSeek-R1-Distill-Qwen-1.5B. Specifically, we construct efficient reasoning sets by sampling Direct CoTs and Vanilla CoTs from various datasets, encompassing different mathematical difficulty levels (GSM8K, MATH500, AMC 2023, AIME 2025) as well as a distinct scientific domain (GPQA). For GSM8K and MATH500, we randomly select 300 questions and generate one Direct CoT and one Vanilla CoT per question. For GPQA, we generate one Direct CoT and one Vanilla CoT for each question in the dataset. For AMC 2023 and AIME 2025, due to the limited number of available questions, we oversample by generating 10 CoT per question to reach a comparable scale.

As shown in Table 3, applying efficient reasoning sets constructing from different datasets consistently outperforms the Vanilla baseline in both accuracy and token reduction across all evaluation benchmarks. Remarkably, even when the efficient reasoning set is constructed from easy math (GSM8K) or scientific questions (GPQA), it still effectively guides the model to solve highly complex competition-level math (AIME 2025) and coding tasks (LiveCodeBench). This transferability demonstrates that the efficient reasoning set $\mathcal{H}$ captures a general, domain-agnostic geometric property of efficient reasoning in the latent space.

Furthermore, we observe that the efficient reasoning set constructed from the MATH500 dataset yields a highly effective and balanced performance across both in-domain and out-of-domain evaluations. Consequently, we adopt the MATH500-derived set as our default experimental setting throughout the main text.

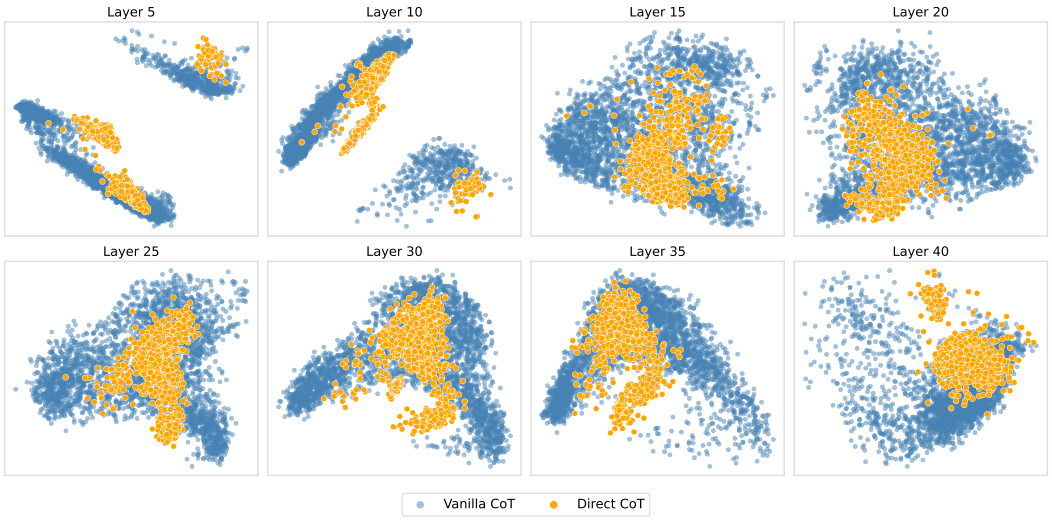


Figure 6: Visualization of different reasoning steps generated by Qwen3-14B. The horizontal and vertical axes correspond to the first and second PCA components, respectively.

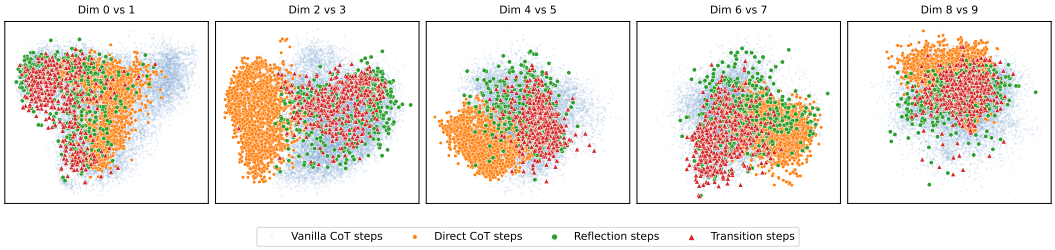


Figure 7: Visualization of latent representations across different PCA dimensions. The results show that Direct CoT steps remain concentrated in a core region across various dimension pairs (e.g., Dim 0 vs 1, Dim 2 vs 3), while overthinking behaviors such as Transition and Reflection steps fall into the scattered region.

D.2 Analysis of Semantic Change

To further analyze the semantic changes of the model's output after our method's intervention, we conduct a statistical semantic analysis over all generated reasoning trajectories on the MATH500 dataset. Following the vocabularies introduced in SEAL [16], we explicitly quantify the occurrence of Transition and Reflection vocabularies by reporting their average word count, term frequency (TF), and term frequency-inverse document frequency (TF-IDF) scores.. For readability, the TF

Table 3: Cross-Domain and Cross-Difficulty Transferability on DeepSeek-R1-Distill-Qwen-1.5B.

Methods	GSM8K		MATH500		AMC2023		AIME2025		LiveCodeBench		GPQA-Diamond	
	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.	Acc.	Tok.
Vanilla	81.5	1489	82.1	4476	64.4	7708	17.3	11811	32.1	10013	33.3	8241
Ours-MATH500	83.1	853	83.2	2949	75.2	4821	26.7	8502	32.7	7656	36.4	5651
Ours-GSM8K	82.3	707	82.3	2980	76.7	4825	23.3	8097	32.8	7575	35.8	5738
Ours-AMC23	82.6	782	82.9	2951	72.5	4609	22.2	8649	32.5	7594	35.3	5590
Ours-AIME2025	82.7	901	82.0	3294	71.6	4939	30.0	8727	33.4	8393	34.9	6149
Ours-GPQA	81.6	843	82.4	3141	70.0	5377	25.5	9145	32.2	8411	34.8	5827

and TF-IDF scores are scaled by a factor of $1,000$. These metrics are evaluated alongside the final reasoning Accuracy and overall Token consumption.

As detailed in Table 4, we observe a consistent reduction in the frequency of these specific reasoning steps across all evaluated models. This indicates that our method effectively limits the model's tendency to engage in extensive self-verification (Reflection) and perspective shifts (Transition). Notably, this reduction directly correlates with lower overall token consumption without compromising reasoning accuracy.

Table 4: Semantic statistics of Transition and Reflection vocabularies.

Method	Reflection			Transition			Performance	
	Word Count	TF	TF–IDF	Word Count	TF	TF–IDF	Acc.	Tok.
DeepSeek-R1-Distill-Qwen-1.5B								
Baseline	28.5	9.3	13.0	8.0	2.5	4.7	82.1	4476
Ours	6.0	4.2	6.6	1.7	0.7	1.5	83.2	2949
DeepSeek-R1-Distill-Qwen-7B								
Baseline	18.3	7.6	10.3	4.9	2.1	3.7	91.6	3647
Ours	6.1	4.3	6.4	1.9	1.0	1.7	92.2	2599
Qwen3-4B								
Baseline	32.2	9.0	11.5	11.5	4.0	5.5	95.2	4587
Ours	15.3	6.7	8.9	3.5	1.5	2.3	95.8	3550
Qwen3-14B								
Baseline	22.3	7.8	12.5	7.3	2.8	4.9	96.2	4559
Ours	3.6	2.0	3.4	1.0	0.6	1.1	96.8	3510

D.3 Creativity Analysis

While our method is specifically developed to improve reasoning capacity, a natural question arises: could this method introduce unintended side effects on other fundamental capabilities? To address this concern, we extend our analysis to investigate the method's influence on other general model abilities. Specifically, we focus on exploring whether it negatively impacts creativity or the naturalness of linguistic expressions. To conduct this evaluation, we assess the models using the Creative Writing v3 benchmark [43], which is also adopted in recent studies [8, 27]. The detailed results of this assessment are presented in Table 5.

In our evaluation, we utilize officially recommended Claude-Sonnet-4.0 as the judge model. Specifically, we focus on two primary metrics: (1) **Rubric Score**, which provides a comprehensive evaluation of overall writing quality across various dimensions; (2) **Fine-grained Ability Scores**, which measure performance across fifteen distinct creative writing skills. These competencies include *coherence* (logical consistency and clear structure), *creativity* (originality and avoidance of templates), *descriptive imagery* (vivid and sensory language), *pacing* (appropriate narrative flow), *elegant prose* (stylistic fluency), *instruction following* (strict adherence to prompts), *consistent voice and tone* (stable narrative perspective), *strong dialogue* (natural and character-driven conversations), *sentence flow* (smooth transitions), *show-don't-tell* (conveying meaning through actions rather than direct exposition), *avoidance of amateurish prose* (steering clear of clichéd patterns), *emotional depth* (nuanced feelings), *avoidance of positivity bias* (rejecting forced optimism), *avoidance of purple prose* (restraint from overly ornate vocabulary), and *believable characters* (psychological realism).

As demonstrated in the results, applying our method does not compromise the semantic richness of the outputs. Instead, the models largely preserve, and in several cases even enhance, their performance regarding creative expression and natural language fluency.

D.4 Pass@k Performance Analysis

To further analyze whether our method affects the model's exploration capability, we conduct a Pass@k evaluation. Pass@k measures the probability that at least one out of k independently

Table 5: Creative-writing performance on Creative Writing v3 benchmark. Ability abbreviations: Coh = Coherent; Crt = Creativity; Img = Descriptive Imagery; Pac = Pacing; Ele = Elegant Prose; Inst = Instruction Following; Voi = Consistent Voice & Tone; Dia = Strong Dialogue; Flo = Sentence Flow; SDT = Show-Don'tTell; Ama = Avoids Amateurish Prose; Emo = Emotional Depth; Pos = Avoids Positivity Bias; Pur = Avoids Purple Prose; Ch = Believable Characters.

Method	Score	Ability														
	Rubric	Coh	Crt	Img	Pac	Ele	Inst	Voi	Dia	Flo	SDT	Ama	Emo	Pos	Pur	Ch
DeepSeek-R1-Distill-Qwen-1.5B																
Baseline	13.80	0.55	1.20	1.09	1.20	0.30	0.39	0.84	1.33	0.48	1.38	0.91	0.19	**16.88**	**14.83**	0.28
Ours	**14.16**	**0.75**	**1.53**	**1.27**	**1.42**	**0.45**	**0.50**	**0.93**	**1.37**	**0.60**	**1.61**	**1.33**	**0.31**	14.60	13.78	**0.52**
DeepSeek-R1-Distill-Qwen-7B																
Baseline	21.40	3.60	3.20	3.92	4.98	2.30	**2.11**	3.89	3.36	3.01	3.83	2.78	**1.73**	**12.48**	**11.72**	2.84
Ours	**22.56**	**4.19**	**3.99**	**4.07**	**5.81**	**2.40**	2.10	**4.21**	**4.26**	**3.31**	**4.24**	**3.57**	1.62	12.22	11.62	**3.15**
Qwen3-4B																
Baseline	36.83	10.98	**4.71**	**7.89**	**10.56**	5.89	5.35	10.02	5.02	8.29	**4.73**	**4.93**	3.79	**13.56**	11.61	7.88
Ours	**37.44**	**12.05**	4.64	7.75	8.85	**6.12**	**5.56**	**10.62**	**5.18**	**8.62**	4.45	4.92	**4.11**	13.36	**12.25**	**8.22**
Qwen3-14B																
Baseline	59.06	16.41	**6.32**	12.45	**15.45**	10.74	11.74	15.21	8.90	13.56	**8.44**	8.67	8.30	**15.83**	**14.37**	13.74
Ours	**59.89**	**16.73**	6.18	**12.47**	14.91	**10.86**	**13.37**	**15.43**	**9.79**	**13.60**	8.40	**9.09**	**8.79**	14.86	14.07	**13.96**

generated reasoning trajectories yields the correct answer. It is widely used to assess the exploration capability of a model's generation distribution [44]. Specifically, we evaluate our method on two hard mathematical benchmarks, AMC2023 and AIME2025. We report the Pass@1, Pass@4, Pass@8, and Pass@16 scores in Table 6. The results show that our method consistently maintains, and even improves, the Pass@k performance across various model scales. For example, on the AIME 2025 dataset using the DeepSeek-R1-Distill-Qwen-7B model, our method improves Pass@1 from 26.9% to 39.0% and Pass@16 from 66.7% to 70.0%. The results demonstrate that projecting hidden states into the efficient reasoning set $\mathcal{H}$ does not collapse the model's output diversity nor harm its ability to navigate complex problem spaces. Instead, by pruning unproductive deviations, the model concentrates its exploration capacity on highly promising reasoning trajectories, achieving a better balance between reasoning diversity and generation efficiency.

Table 6: Pass@k Performance on AMC 2023 and AIME 2025.

Methods	AMC 2023				AIME 2025			
	Pass@1	Pass@4	Pass@8	Pass@16	Pass@1	Pass@4	Pass@8	Pass@16
DeepSeek-R1-Distill-Qwen-1.5B								
Vanilla	64.4	85.6	92.5	95.0	17.3	33.3	40.0	40.0
Ours	**75.2**	**91.2**	**96.2**	**97.5**	**26.7**	**36.7**	**40.0**	**40.0**
DeepSeek-R1-Distill-Qwen-7B								
Vanilla	86.3	94.3	95.0	95.0	26.9	58.3	63.3	66.7
Ours	**91.1**	**96.3**	**97.5**	**100.0**	**39.0**	**60.0**	**66.7**	**70.0**
Qwen3-4B								
Vanilla	95.9	100.0	100.0	100.0	60.2	70.0	81.7	86.7
Ours	**97.5**	**100.0**	**100.0**	**100.0**	**67.1**	**78.3**	**83.3**	**86.7**
Qwen3-14B								
Vanilla	96.9	100.0	100.0	100.0	67.1	76.7	83.3	83.3
Ours	**98.3**	**100.0**	**100.0**	**100.0**	**70.4**	**76.7**	**83.3**	**83.3**

D.5 Statistical Reliability and Significance Testing

For small-sized benchmark datasets such as AMC2023 and AIME2025, we report the mean accuracy (%), average token consumption per problem, and corresponding standard deviations to establish statistical reliability. Across all evaluated models, our approach yields statistically significant accuracy improvements alongside substantial reductions in token consumption.

Table 7: **Statistical reliability evaluation on AMC23 and AIME2025.** We report mean accuracy, mean token count and corresponding standard deviations across datasets and model sizes.

Model	Method	AMC2023		AIME2025	
		Acc	Tokens	Acc	Tokens
DeepSeek-1.5B	Vanilla	64.4 ± 5.4	7708 ± 703	17.3 ± 3.7	11811 ± 625
	Ours	75.2 ± 3.5	4821 ± 304	26.7 ± 4.2	8502 ± 390
DeepSeek-7B	Vanilla	86.3 ± 4.0	5846 ± 358	26.9 ± 3.9	11209 ± 780
	Ours	91.1 ± 2.7	4083 ± 261	39.0 ± 4.0	9042 ± 631
Qwen-4B	Vanilla	95.9 ± 1.7	7588 ± 475	60.2 ± 3.7	16835 ± 944
	Ours	97.5 ± 1.3	5898 ± 187	67.1 ± 5.2	14809 ± 525
Qwen3-14B	Vanilla	96.9 ± 1.5	7277 ± 336	67.1 ± 3.7	16019 ± 888
	Ours	98.3 ± 1.3	5751 ± 296	70.4 ± 3.8	13144 ± 603

E Details on Solving Quadratic Programming

In this section, we provide a comprehensive discussion on how we formulate and efficiently solve the projection problem (7).

E.1 The Definition of Quadratic Programming

A standard Quadratic Programming (QP) problem aims to minimize a quadratic objective function subject to linear constraints. The general formulation is given by:

$$\min_{u \in \mathbb{R}^k} \frac{1}{2} u^\top P u + q^\top u \quad \text{s.t.} \quad Gu \leq h \tag{11}$$

where $P \in \mathbb{R}^{k \times k}$ is a symmetric positive-definite matrix, $q \in \mathbb{R}^k$ is a linear coefficient vector, $G \in \mathbb{R}^{M \times k}$ represents the constraint matrix, and $h \in \mathbb{R}^M$ is the constraint boundary vector.

In our method, the projection operation seeks the closest point z_t^* within the efficient reasoning set $\mathcal{H}$ by minimizing its Euclidean distance to the original hidden state z_t:

$$z_t^* = \arg\min_{u \in \mathbb{R}^k} \frac{1}{2} \|z_t - u\|_2^2 \quad \text{s.t.} \quad w_m^\top u + b'_m \leq 0, \forall\, m \in \{1, \ldots, M\} \tag{12}$$

By expanding the objective function $\frac{1}{2}\|z_t - u\|_2^2 = \frac{1}{2} u^\top u - z_t^\top u + \frac{1}{2} z_t^\top z_t$, and omitting the constant term $\frac{1}{2} z_t^\top z_t$ which does not affect the optimization, we can exactly map our problem to the standard QP format:

- The quadratic matrix P is the identity matrix $I \in \mathbb{R}^{k \times k}$.
- The linear coefficient vector q is $-z_t$.
- The constraint matrix $G \in \mathbb{R}^{M \times k}$ is composed of the normal vectors $w_m^\top$.
- The constraint boundary vector $h \in \mathbb{R}^M$ is composed of the elements $-b'_m$.

Crucially, in our method, the constraint boundaries defining $\mathcal{H}$ (i.e., G and h) are pre-computed and fixed during inference. However, the linear term $q = -z_t$ varies dynamically at each reasoning step. Because the optimization problem is parameterized by the dynamically changing input vector z_t, it is also classified as a Multi-Parametric Quadratic Programming (mp-QP) problem.

E.2 Online Numerical Solving Approach

For a given input state z_t, the mp-QP is instantiated into a specific standard QP instance. Typically, solving such instances relies on iterative numerical optimization algorithms, such as the active-set method [45] and first-order method [46]. These numerical methods are highly robust and have been integrated into well-established solvers.

For online solving, we utilize `CvxpyLayer`, a differentiable convex optimization solver, to compute the optimal solution z_t^* during the model's forward pass. While numerically reliable, this online

approach suffers from a major bottleneck: numerical solvers typically execute on the CPU. Consequently, for every intervention, the inference engine must transfer the hidden state z_t from the GPU to the CPU, wait for the solver to converge, and transfer z_t^* back. This frequent CPU-GPU communication introduces additional latency for high-throughput LLM serving.

E.3 Explicit Acceleration via Offline mp-QP

To eliminate the CPU-GPU communication bottleneck, we exploit a fundamental structural property of mp-QP: the optimal solution z_t^* can be analytically expressed as a continuous piecewise affine function of the input parameter z_t [21].

Specifically, the continuous k-dimensional latent space can be partitioned into a finite set of non-overlapping critical regions $\{\mathcal{CR}_1, \ldots, \mathcal{CR}_N\}$. Within each specific region $\mathcal{CR}_i$, the optimal solution z_t^* is strictly given by a closed-form affine mapping:

$$z_t^* = F_i z_t + g_i, \quad \text{if } z_t \in \mathcal{CR}_i \tag{13}$$

where $F_i \in \mathbb{R}^{k \times k}$ and $g_i \in \mathbb{R}^k$ are pre-computable matrices. Each critical region $\mathcal{CR}_i$ is a convex polyhedral set characterized by linear inequalities, i.e., $\mathcal{CR}_i = \{z \in \mathbb{R}^k \mid H_i z \leq K_i\}$.

We leverage the `PDAQP` solver to perform this partitioning and matrix calculation offline [31]. During online deployment, solving the complex optimization problem is decomposed into two pure GPU tensor operations: (1) Region Querying, which identifies the index i such that $z_t \in \mathcal{CR}_i$; and (2) Affine Transformation, which directly applies $F_i z_t + g_i$ to obtain the solution. This computation operates entirely on the GPU, completely bypassing the CPU and ensuring high-throughput reasoning.

E.4 Computational Overhead Analysis

The explicit acceleration above heavily relies on the scale of the problem. Because our method intervenes in a highly compressed latent space, the QP involves only a very low dimension of 10 to 16 decision variables and exactly $M = 8$ linear constraints. This prevents the number of critical regions from combinatorial explosion, making offline calculation and storage extremely lightweight.

Experimental Design. To evaluate the computational overhead presented in Table 8, we collected 1,000 instances of the parameter z_t generated during the actual inference process across different models. We compared the online QP solving method (denoted as *Online*) with our accelerated approach (denoted as *Explicit*). Specifically, the online method directly invokes the numerical solver CvxpyLayer for each QP instance during the model's forward pass. The explicit method relies on offline mp-QP precomputation executed using the PDAQP solver [31], followed by online region querying implemented via PyTorch operators. The *Additional Memory* metric quantifies the GPU memory footprint required to store the pre-computed piecewise linear expressions for the explicit method.

Results and Conclusions. As demonstrated in Table 8, the explicit acceleration yields dramatic efficiency gains. The online solving method (CvxpyLayer) requires between 1.7 ms to 5.3 ms per operation across different models. In contrast, our explicit GPU-accelerated approach reduces this latency to a mere 0.3 ms to 0.5 ms, achieving approximately a $10\times$ speedup. Furthermore, storing the offline-computed piecewise linear expressions (i.e., matrices F_i and g_i) introduces an exceptionally small GPU memory footprint, ranging from just 0.41 MB for the 1.5B model to 1.80 MB for the 14B model. This negligible memory cost ensures that our acceleration framework can be seamlessly integrated into environments with resource constraints.

F Details on Experiment Settings

In this section, we provide a comprehensive overview of our experimental setup, including the mathematical notations used throughout the paper, the specific prompt templates for generating reasoning trajectories, and the detailed hyperparameter settings and decoding configurations for the models evaluated.

Table 8: Comparison of Average Solving Time. We detail the scale of the QP problem and compare the latency of the online solver with the explicit acceleration. The additional memory denotes the GPU footprint required to store the precomputed piecewise linear mappings for the explicit method.

Model	Variables	Constraints	Online (ms)	Explicit (ms)	Additional Memory (MB)
R1-1.5B	10	8	1.7	0.3	0.41
R1-7B	12	8	2.3	0.3	0.93
Qwen3-4B	12	8	2.9	0.3	0.56
Qwen3-14B	16	8	5.3	0.5	1.80

F.1 Notation Summary

To facilitate reading and ensure mathematical clarity, we summarize the key notations used in our problem formulation, latent space analysis, and method in Table 9.

Table 9: Mathematical notations and their descriptions used in our method.

Symbol	Description
X_{dir}, X_{van}	The original representations of Direct CoTs and Vanilla CoTs.
Z_{dir}, Z_{van}	The representations of Direct CoTs and Vanilla CoTs after PCA projection.
$\mathcal{H}$	The efficient reasoning.
U	The PCA projection matrix.
k	The dimension of the latent space after the PCA projection.
M	The number of clusters defining the efficient reasoning set $\mathcal{H}$.
α	The boundary shift proportion, determining the inclusivity of the set $\mathcal{H}$
λ	The intervention strength controlling the magnitude of the correction.

F.2 Prompt Structure

We first prompt the model using the ground-truth answer along with conciseness instructions. During generation, we discard the intermediate `<think>` block and keep only the finalized deduction generated after the `</think>` token. Note that for some problems exceeding the model's reasoning capability, the model may fail to produce a valid deduction and hits the maximum response length limit. We filter out these incomplete generations.

To extract hidden representations, we concatenate this finalized deduction with the initial question prompt (i.e. Vanilla CoT prompt) to construct a synthetic sequence [Vanilla CoT Prompt]+`<think>`+[Direct CoT]. Then we perform a forward pass on this synthetic sequence and extract the hidden representations at step delimiters (e.g., `\n\n`). The prompt templates used in our method are provided below.

The Prompt of Vanilla CoT

Answer the following questions. You should think step-by-step and put your final answer within \\boxed{}.

Question: {QUESTION}

The Prompt for Generating Direct CoT

Given a question and its answer, provide a detailed derivation.

Requirements

1. DO NOT use headers like "Step-by-Step" or other subtitles(e.g. "Step 1").

2. ONLY provide the sequential logic and the necessary math steps in a cohesive flow.

3. Ensure the derivation is compact and direct.

4. The result must conclude with the final answer in \\boxed{}.

Task

Question: {QUESTION}

Answer: {ANSWER}

F.3 Hyperparameter and Decoding Settings.

Decoding Settings. Following the officially recommended configurations for each model, we utilize a top-p of 0.95, a temperature of 0.6. The maximum sequence length is set to 16,384 tokens for the DeepSeek-R1-Distill-Qwen-1.5B/7B models, and 32,768 tokens for the Qwen3-4B/14B model.

Hyperparameter Settings. Based on the extensive ablation studies detailed in Appendix G, we use a fixed intervention strength $\lambda = 1.0$, cluster number $M = 8$, and boundary proportion $\alpha = 90\%$ across all models. The steering layer and PCA dimension k are configured as follows: DeepSeek-R1-Distill-Qwen-1.5B (layer 20, $k = 10$), DeepSeek-R1-Distill-Qwen-7B (layer 20, $k = 12$), Qwen3-4B (layer 25, $k = 12$), and Qwen3-14B (layer 30, $k = 16$).

G Details on Ablation Study

In this section, we present the comprehensive results of tuning key hyperparameters across three models: DeepSeek-R1-Distill-Qwen-1.5B, DeepSeek-R1-Distill-Qwen-7B, and Qwen3-4B on the MATH500 benchmark.

G.1 Impact of Intervention Strength λ

We evaluate the effect of the intervention strength λ, which controls the magnitude of the state correction. As shown in Figure 8, increasing λ up to 1.0 yields a substantial reduction in token consumption without compromising the accuracy. However, exceeding this threshold ($\lambda > 1.0$) introduces a clear trade-off: overly aggressive interventions provide only marginal token savings while degrading reasoning accuracy, likely due to latent state over-correction. Finally, $\lambda = 1.0$ offers a favorable balance across different models, and thus we adopt it as our default setting.

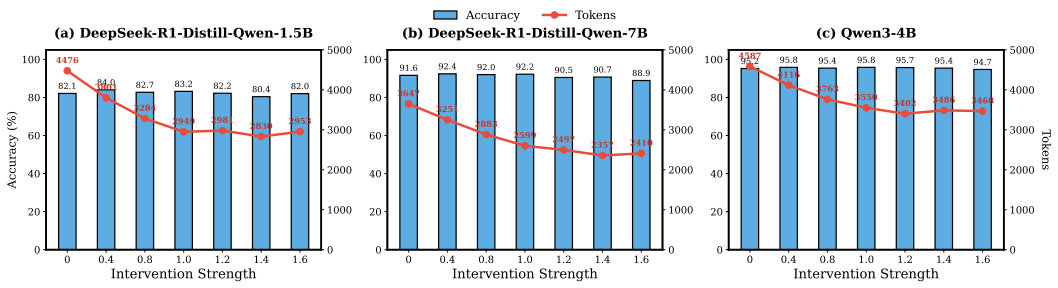


Figure 8: The impact of intervention strength λ on Pass@1 accuracy and average token count across three models on the MATH500 dataset.

G.2 Impact of Steering Layer

Figure 9 illustrates the sensitivity of the performance to the chosen intervention layer. Intervening at middle-to-deep layers generally yields better performance, as these layers usually capture more abstract reasoning features [47]. Based on the empirical results, we set the steering layer to 20 for both the DeepSeek-R1-Distill-Qwen-1.5B and 7B models, and to 25 for the Qwen-4B model.

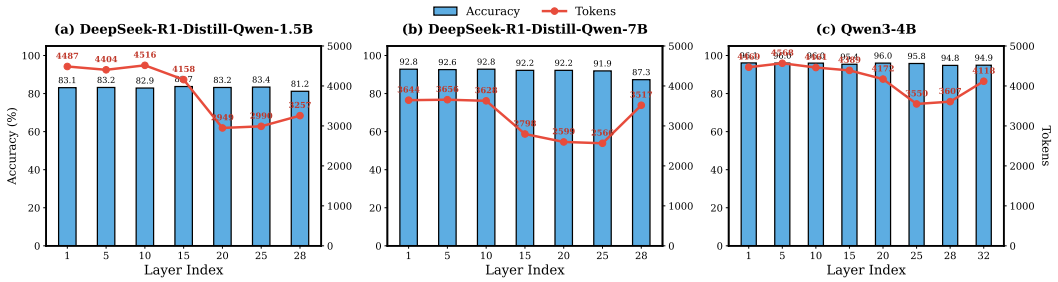


Figure 9: The impact of the steering layer index on Pass@1 accuracy and average token count across three models on the MATH500 dataset.

G.3 Impact of PCA Dimension k

As shown in Figure 10, we investigate the effect of the latent space dimension k on the reasoning performance. A very low dimension may lead to severe information loss, making it difficult to construct a efficient reasoning set. Conversely, a high dimension introduces unnecessary noise and complicates the projection process without bringing notable performance gains. Since the performance remains relatively stable within the middle range, we select $k = 10$ for the DeepSeek-R1-Distill-Qwen-1.5B model, and $k = 12$ for both the DeepSeek-R1-Distill-Qwen-7B and Qwen3-4B models.

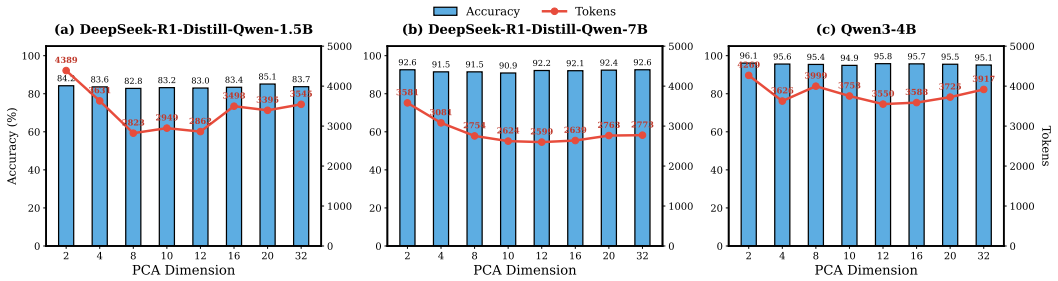


Figure 10: The impact of the PCA dimension k on Pass@1 accuracy and average token count across three models on the MATH500 dataset.

G.4 Impact of Boundary Proportion α

Figure 11 demonstrates the influence of the boundary shift proportion α. When α is small, the constructed set $\mathcal{H}$ is highly restricted. This leads to relatively aggressive projections that can inevitably discard useful semantic variations, resulting in a slightly weaker accuracy. On the other hand, when $\alpha = 100\%$, the token reduction becomes less pronounced. This is likely because a fully inclusive proxy set may also encompass some "overthinking" behaviors such as redundant double-checking, making the intervention conservative. Therefore, we select $\alpha = 90\%$ as a default setting across all models.

G.5 Impact of Cluster Number M

Figure 12 illustrates the effect of M, which defines the number of half-space constraints used to form the set $\mathcal{H}$. Empirically, a small M yields limited efficiency gains, which may be due to the resultant

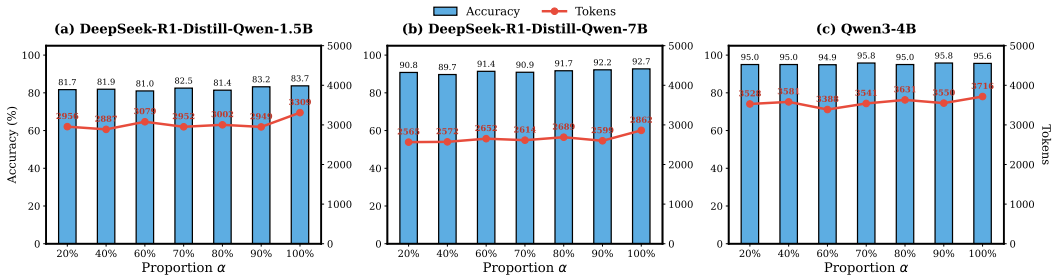


Figure 11: The impact of the boundary shift proportion α on Pass@1 accuracy and average token count across three models on the MATH500 dataset.

set forming an overly loose boundary. As M increases, the performance improves, likely because the boundary becomes tighter and more refined. However, a larger M introduces additional linear constraints into the QP problem, inevitably increasing the computational latency during inference. To balance reasoning efficiency and computational overhead, we set $M = 8$ across all models.

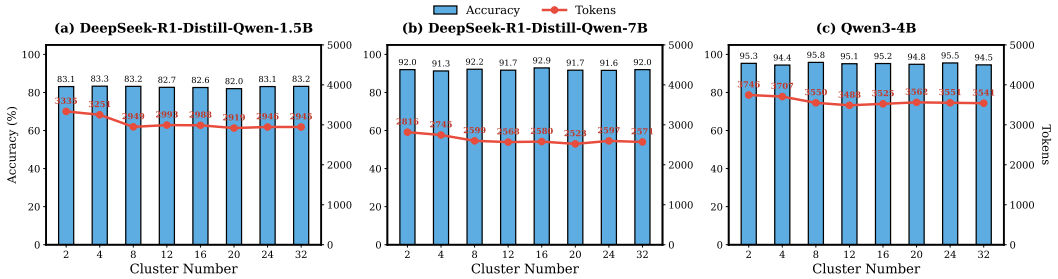


Figure 12: The impact of the cluster number M (number of linear constraints) on Pass@1 accuracy and average token count across three models on the MATH500 dataset

H Limitations

Several aspects of our study warrant future investigation. First, our current empirical evaluations are constrained by computational resources to reasoning models with up to 14B parameters. It remains an open question whether the geometric concentration of efficient reasoning steps observed in this work persists in much larger models (e.g., >70B parameters), which may exhibit more complex latent space. Second, while our method successfully improve the reasoning performance on general reasoning tasks such as mathematics and coding, its efficacy on highly specialized professional domains, such as complex legal argumentation or medical diagnosis, requires further evaluation.

I Broader Impacts

Our work focuses on achieving efficient reasoning and presents several potential positive societal impacts. Primarily, by reducing redundant token generation, our method drastically lowers the computational cost and energy consumption associated with deploying advanced AI systems. This efficiency gain can contribute to a reduced carbon footprint for AI operations. Furthermore, by alleviating the hardware requirements for high-performance reasoning, our approach makes powerful reasoning models more accessible to researchers, educators, and organizations with limited computational resources. As our research focuses on a generic optimization for inference efficiency, it serves as foundational research and does not introduce any direct paths to specific negative applications beyond the inherent capabilities of the underlying base models.

J Case Study

Case 1

Prompt: Jamaar loves fresh fruit and is headed to the store with $10 he earned mowing lawns. Including tax, peaches and pears are $.5 each, apples are $.75 each, kiwis are $1, and plums cost $.25 each. If he has already purchased 3 peaches, 4 pears, 2 kiwis, and 5 apples, how many plums can he buy? **Gold: 3**

✗ Vanilla Output (673 Tokens)

Alright, so I've got this problem here about Jamaar buying fruit with money he earned. Let me try to figure this out step by step. Sometimes math problems can be a bit overwhelming, but breaking them down usually helps.
Let me list out the information given:

- **Total money Jamaar has:** $10
- **Cost of each fruit:** Peaches: $0.5 each; Pears: $0.75 each; Kiwis: $1 each; Plums: $0.25 each
- **Fruits already purchased:** 3 peaches, 4 pears, 2 kiwis, 5 apples

I need to calculate how much money he has spent on the fruits he already bought and then subtract that from his total to see how much he has left. After that, I can figure out how many plums he can buy...
1. **Peaches:** He bought 3 peaches, each costing $0.5. So, 3 × $0.5 = $1.50
2. **Pears:** He bought 4 pears, each costing $0.75. So, 4 × $0.75 = $3.00
3. **Kiwis:** He bought 2 kiwis, each costing $1. So, 2 × $1 = $2.00
4. **Apples:** He bought 5 apples, each costing $0.25. So, 5 × $0.25 = $1.25
Now, let me add up all these amounts to find the total money spent so far.
Total spent = $1.50 + $3.00 + $2.00 + $1.25
First, $1.50 + $3.00 = $4.50
Then, $4.50 + $2.00 = $6.50
Next, $6.50 + $1.25 = $7.75
So, he's spent a total of $7.75 on the fruits he already purchased.
Now, let's subtract this from his total money:
Money left = $10.00 - $7.75 = $2.25
Now, each plum costs $0.25, so I need to find out how many plums he can buy with $2.25.
Number of plums = $2.25 / $0.25
Let me think, $0.25 goes into $1.00 four times, right? So, $0.25 goes into $2.25 how many times?
Well, 0.25 × 9 = 2.25 because 9 × 0.25 is 2.25.
Wait, let me double-check that.
0.25 × 10 = 2.50, which is more than 2.25, so it must be 9. Yes, that makes sense. So, 9 plums.
Wait a second, let me just make sure I didn't make a mistake in the initial calculations.
Total spent on peaches: 3 × 0.5 = 1.50. Correct.
Pears: 4 × 0.75 = 3.00. Correct.
Kiwis: 2 × 1 = 2.00. Correct.
Apples: 5 × 0.25 = 1.25. Correct.
Total money: 10.00 - 7.75 = 2.25. Correct.
Number of plums: 2.25 / 0.25 = 9. Correct.
So, I think that's right. Therefore, Jamaar can buy 9 plums with the remaining money. `</think>`

Jamaar has $10 and has already spent $7.75 on peaches, pears, kiwis, and apples. With the remaining $2.25, he can buy 9 plums.

Answer: Jamaar can buy $\boxed{9}$ plums.

✓ Ours Output (221 Tokens)

First, calculate the total amount of money Jamaar has spent on peaches, pears, kiwis, and apples.
Peaches: 3 peaches × $0.50 = $1.50
Pears: 4 pears × $0.50 = $2.00
Kiwis: 2 kiwis × $1.00 = $2.00
Apples: 5 apples × $0.75 = $3.75
Add these amounts to find the total expenditure:
$1.50 + $2.00 + $2.00 + $3.75 = $9.25
Subtract this from the total money to find the remaining amount:
$10.00 - $9.25 = $0.75
Finally, divide the remaining money by the cost per plum to determine how many plums Jamaar can buy:
$0.75 ÷ $0.25 = 3 plums.
`</think>`

****Solution:****
Jamaar has a total of $10 to spend. He has already purchased various fruits, and we need to determine how many plums he can buy with the remaining money.
1. Calculate the total cost of the fruits already purchased:

- **Peaches:** 3 × $0.50 = $1.50
- **Pears:** 4 × $0.50 = $2.00
- **Kiwis:** 2 × $1.00 = $2.00
- **Apples:** 5 × $0.75 = $3.75

2. Sum up the costs:

$$\$1.50 + \$2.00 + \$2.00 + \$3.75 = \$9.25$$

3. Determine the remaining money:

$$\$10.00 - \$9.25 = \$0.75$$

4. Calculate the number of plums Jamaar can buy:
Each plum costs $0.25.

$$\frac{\$0.75}{\$0.25} = 3$$

Final Answer:

$$\boxed{3}$$

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The claims are supported by Section 6.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The analysis of limitations is provided in Appendix H.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: We provide it in Appendix B.1

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We disclose all information needed for reproduction in the Section 6 and Appendix F.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We will provide data, code, and instructions for the final paper.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: We provide experimental details in Appendix F.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: For smaller datasets (AMC 2023, AIME 2024, and AIME 2025), we perform 16 independent sampling rounds per instance, with the standard deviations reported in Appendix D.5.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The information on the computer resources is provided in Section 6.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Our research conformed with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discuss potential positive societal impacts in Appendix I

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: This paper focuses on inference processes and does not include model training, so these risks are not applicable.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We properly credit data, code, and models that we used in this paper.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The paper does not release new assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: We only use LLM for evaluation.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.