

# 000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015 016 017 018 019 020 021 022 023 024 025 026 027 028 029 030 031 032 033 034 035 036 037 038 039 040 041 042 043 044 045 046 047 048 049 050 051 052 053 REASONING VIA TEST-TIME INSTANCE-LEVEL POLICY GRADIENT IN LATENT SPACE

Anonymous authors

Paper under double-blind review

## ABSTRACT

Large Language Models (LLMs) typically reason through explicit, step-by-step natural-language traces. Humans, however, also rely on non-linguistic, unconscious processes, such as the inspirations that emerge during the incubation period. In this work, we introduce LATENTSEEK, a novel framework designed to enhance the reasoning capabilities of LLMs through Test-Time Instance-level Policy Gradient within the model’s latent space—thus complementing explicit natural-language steps. LATENTSEEK employs policy gradient optimization to iteratively refine latent representations, guided solely by a self-generated reward signal. This allows the model to adapt its reasoning trajectory dynamically on a per-instance basis. Empirical evaluations across diverse benchmarks, GSM8K, MATH-500, and AIME2024 as well as multiple LLM families (e.g., LLaMA, Qwen) demonstrate that LATENTSEEK outperforms established baselines, including Chain-of-Thought (CoT), Best-of-N (BoN) and training-based methods. Further analysis indicates that LATENTSEEK is computationally efficient, typically converging within a few optimization iterations for average-level problems. Moreover, the model’s performance improves as the number of latent update iterations increases, highlighting the benefits of exploring within the latent space. These findings highlight LATENTSEEK as a lightweight and effective paradigm for improving the reasoning capabilities of LLMs without changing their parameters.

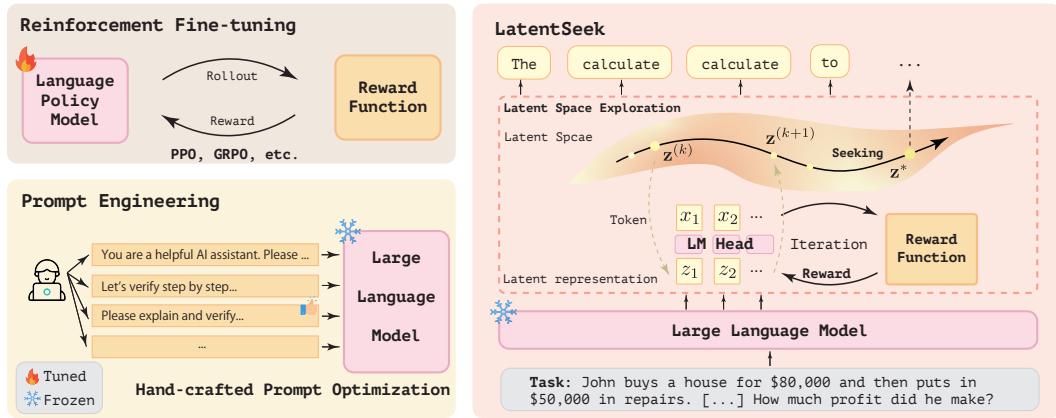


Figure 1: Comparison of LATENTSEEK with RL-based fine-tuning and Prompt Engineering. RL-based fine-tuning methods generally require iterative updates to model parameters guided by reward signals. Prompt engineering approaches depend heavily on manually designed prompts. In contrast, LATENTSEEK performing optimization within the latent space. Of note, the output of LATENTSEEK may be incoherent and semantically ungrounded; please refer to §3.7.

## 1 INTRODUCTION

Large Language Models (LLMs) have demonstrated remarkable reasoning abilities, largely unlocked by techniques such as Chain-of-Thought (CoT) prompting, which guides them to generate step-by-

054 step solutions in natural language (Achiam et al., 2023; Wei et al., 2022). This approach, however,  
 055 presupposes that the optimal reasoning path for a neural network is one that mirrors human linguistic  
 056 expression. However, research in human cognition suggests that effective reasoning is not confined to  
 057 conscious, verbal articulation (Sio and Ormerod, 2009; Ritter and Dijksterhuis, 2014); it also involves  
 058 implicit, non-verbal processes, such as the “incubation” period where inspirations can emerge without  
 059 conscious effort (Sio and Ormerod, 2009; Sklar et al., 2012). These findings suggest that constraining  
 060 LLMs to reason only in natural language may be unnecessarily restrictive.

061 An emerging line of research validates this concept, using specialized fine-tuning to “internalize”  
 062 reasoning steps into sequences of continuous thought vectors (Deng et al., 2024; Hao et al., 2024).  
 063 While these approaches validate the promise of latent reasoning, they rely on costly and complex  
 064 training curricula to adapt the model. As illustrated in Figure 1, this aligns them with the broader  
 065 paradigm of training-based methods such as supervised fine-tuning or reinforcement fine-tuning,  
 066 which require iterative parameter updates that are computationally intensive and permanently alter  
 067 the backbone (Ouyang et al., 2022; DeepSeek-AI, 2025). As a result, they underutilize the model’s  
 068 semantic capacity in its pre-trained **latent space**<sup>1</sup> and, in practice, present inferior performance to  
 069 that of CoT. This raises a critical question: *Can we take the benefits of pre-trained latent space to*  
 070 *perform reasoning at test time, without the need for any parameter updates?*

071 Motivated by these observations, we present the **first** attempt to perform seeking in the latent space by  
 072 introducing **LATENTSEEK**, a framework that significantly enhances instance-level reasoning at test  
 073 time. As shown in Figure 1, LATENTSEEK operates on a frozen model, circumventing the costs of fine-  
 074 tuning. Unlike static **prompt engineering**, LATENTSEEK performs dynamic, iterative optimization  
 075 on instance-specific latent representations that steer the pre-trained model’s reasoning process without  
 076 modifying its parameters. These latent representations act as a planning or control mechanism that  
 077 guides the model toward better reasoning paths for each specific problem instance. We optimize latent  
 078 representations at test time using the policy gradient method (Williams, 1992) to maximize reward  
 079 (§2.3). Specifically, for each reasoning problem, we update the token-wise latent representations  
 080 using guidance from the reward function, treating them as independent variables. In each iteration,  
 081 the updated latent representations are decoded into tokens, which serve as inputs for computing the  
 082 reward. Importantly, the reward function operates in a self-rewarding manner, relying solely on the  
 083 model’s internal capabilities without incorporating any external information. The process continues  
 084 until the reward exceeds a predefined threshold or the maximum number of iterations is reached.

085 Our innovative latent space reasoning method is **simple yet effective**: Notably, LATENTSEEK yields  
 086 average gains of **15.23** points over BoN on GSM8K, **4.72** points over Self-Reflection on MATH-500,  
 087 and **6.67** points over CoT on AIME2024 (Table 1). Furthermore, when using LLaMA3.1-8B-Instruct  
 088 as the backbone, LATENTSEEK surpasses prior arts including SimpleRL-Zoo (Williams, 1992)  
 089 (**+18.1**) and Genius (Xu et al., 2025) (**+12.7**), on the GSM8K and MATH-500 datasets (Table 2).  
 090 Further experiments (§3.4) show that test-time performance improves with the number of update  
 091 iterations, suggesting a **complementary scaling axis: the number of optimization steps in latent**  
 092 **space**. To better characterize the latent space, we conduct idealized experiments using a perfect,  
 093 ground-truth verifier that issues sharp rewards. Across all evaluation settings, this yields an average  
 094 gain of **19.12** points over CoT (Table 3), **highlighting the benefits of pure exploration in the**  
 095 **latent space**. Apart from the above, our case studies (§3.7) reveal that LATENTSEEK often attains  
 096 the correct answer even when its intermediate reasoning is unintelligible to human readers. This  
 097 divergence reveals that the optimal reasoning path for LLMs need not mirror human reasoning, and  
 098 that **optimization in latent space may be a more native and effective paradigm**.

099 Our contributions can be summarized as follows:  
 100

- We introduce LATENTSEEK, an efficient yet effective method that enhances reasoning capabilities through test-time instance-level policy gradient, and demonstrate that it outperforms all baselines.
- We conduct a scaling analysis, revealing that performance at test time improves with an increased number of update iterations, highlighting the potential of test-time scaling in the latent space.
- We conduct a statistical analysis to demonstrate the efficiency of LATENTSEEK and the appropriateness of its output length. Our qualitative analysis further indicates that language models may adopt reasoning strategies that diverge from human intuition.

106 <sup>1</sup>In this work, we take the convention (Hao et al., 2024) that treats the transformers’ output space ahead of the  
 107 final language model (LM) head as latent space (Figure 1), and the vector in the space as latent representation  
 (Figure 1); refer to §2.2 for notations.

108 **2 TEST-TIME INSTANCE-LEVEL POLICY GRADIENT IN LATENT SPACE**  
109110 **2.1 PROBLEM FORMULATION: TEST-TIME INSTANCE-LEVEL REASONING**  
111112 Let  $\mathbf{c}$  be the context prompt of a reasoning problem instance and  $\pi$  a pre-trained auto-regressive  
113 language model (LM) composed of a Transformer backbone  $\pi_{\text{Transformer}}$  (Vaswani et al., 2017) and a  
114 LM head  $\pi_{\text{LM-head}}$ . For a reasoning sequence  $\mathbf{x} = (x_1, x_2, \dots, x_T)$ ,

115 
$$\pi(\mathbf{x} \mid \mathbf{c}) = \prod_{t=1}^T \pi(x_t \mid \mathbf{x}_{<t}, \mathbf{c}), \quad \pi(x_t \mid \mathbf{x}_{<t}, \mathbf{c}) = \pi_{\text{LM-head}}(x_t \mid z_t), \quad (1)$$
  
116  
117

118 where  $\mathbf{x}_{<t}$  denotes the sequence of tokens preceding position  $t$  and  $z_t := \pi_{\text{Transformer}}(\mathbf{x}_{<t}, \mathbf{c})$  is the  
119 latent representation associated with  $x_t$ . Following Hao et al. (2024), we refer to  $z_t$  as lying in the  
120 latent space of  $x_t$ . At test time, the ground truth is unknown, and thus a reward function  $R(\mathbf{x}, \mathbf{c})$   
121 is introduced to evaluate the reasoning token sequence. Concluding the above, the objective for a  
122 reasoning problem is to find an optimal reasoning path:  
123

124 
$$\mathbf{x}^* = \arg \max_{\mathbf{x}} R(\mathbf{x}, \mathbf{c}). \quad (2)$$
  
125

126 Please refer to Section B for examples.  
127128 **2.2 REASONING VIA POLICY GRADIENT IN LATENT SPACE**  
129130 To solve the problem in Equation (2), we reformulate the task as optimizing over a sequence of latent  
131 representations rather than directly searching for tokens. Specifically, we denote a sequence of latent  
132 representations  $\mathbf{z} = (z_1, z_2, \dots, z_N)$ , where  $z_t$  lies in the latent space of  $x_t$  and  $N$  is typically an  
133 integer smaller than or equal to  $T$ . To identify the optimal sequence of latent representations, the  
134 to-be-optimized objective is as follows:  
135

136 
$$\mathbf{z}^* = \arg \max_{\mathbf{z}} \mathbb{E}_{\mathbf{x} \sim \pi(\mathbf{x} \mid \mathbf{z}, \mathbf{c})} [R(\mathbf{x}, \mathbf{c})]. \quad (3)$$
  
137

138 To sample  $\mathbf{x} \sim \pi(\mathbf{x} \mid \mathbf{z}, \mathbf{c})$ , we first decode the latents  $\mathbf{z}$  into their corresponding tokens, then continue  
139 with autoregressive generation, since the complete sequence may extend beyond the latent-initialized  
140 reasoning path. As  $T$  is the complete sequence length, the factorization is

141 
$$\pi(\mathbf{x} \mid \mathbf{z}, \mathbf{c}) = \underbrace{\prod_{t=1}^N \pi_{\text{LM-head}}(x_t \mid z_t)}_{\text{decode from latents}} \underbrace{\prod_{t=N+1}^T \pi(x_t \mid \mathbf{x}_{<t}, \mathbf{c})}_{\text{continue generation}}, \quad (4)$$
  
142  
143  
144  
145

146 where, if the generation stops at  $t = N$ , the second product is empty and equals 1.  
147148 **Test-Time Optimization of Latent Representations.** Given the objective in Equation (3), we  
149 optimize the latent representations using a direct policy gradient approach based on REINFORCE  
150 (Williams, 1992). Assuming the *independence of the latent representations*, the update process is:  
151

152 
$$\mathbf{z} \leftarrow \mathbf{z} + \eta \nabla_{\mathbf{z}} \mathcal{J}(\mathbf{z}), \quad (5)$$

153 where the gradient  $\nabla_{\mathbf{z}} \mathcal{J}(\mathbf{z})$ , of our objective with respect to  $\mathbf{z}$  can be derived as:  
154

155 
$$\nabla_{\mathbf{z}} \mathcal{J}(\mathbf{z}) = \mathbb{E}_{\mathbf{x} \sim \pi(\mathbf{x} \mid \mathbf{z}, \mathbf{c})} [R(\mathbf{x}, \mathbf{c}) \nabla_{\mathbf{z}} \log \pi(\mathbf{x} \mid \mathbf{z}, \mathbf{c})]. \quad (6)$$
  
156

157 Leveraging Equation (4), the gradient of the  $t$ -th latent representation is:  
158

159 
$$[\nabla_{\mathbf{z}} \mathcal{J}(\mathbf{z})]_t = \mathbb{E}_{\mathbf{x} \sim \pi(\mathbf{x} \mid \mathbf{z}, \mathbf{c})} [R(\mathbf{x}, \mathbf{c}) \nabla_{z_t} \log \pi(x_t \mid z_t)], \quad (7)$$

160 where the expectation is approximated using the empirical mean in practical implementation. The  
161 independence assumption is adopted for two main reasons. (1) **Enlarged Exploration Space.** Without  
this assumption, the autoregressive structure would force the optimization to concentrate on the *first*

latent token, since all subsequent tokens are conditioned on it. This collapses the effective search space. By contrast, imposing independence decouples the latent components, yielding a substantially larger exploration space and a more flexible “launch pad” for generating the subsequent reasoning trajectory  $\prod_{t=N+1}^T \pi(x_t \mid x_{<t}, \mathbf{c})$ . (2) **Theoretical Guidance.** Our analysis in Section C draws an analogy to Multi-Prover Interactive Proofs (MIP), and shows that the potential loss introduced by the independence assumption can be mitigated by both the base model’s ability to faithfully generate the follow-up reasoning trajectory and the reward model’s ability to evaluate it. Please refer to Sections C and D for a theoretical justification of this assumption and a detailed derivation of the policy-gradient update.

### 2.3 LATENTSEEK ALGORITHM

---

#### Algorithm 1 LATENTSEEK

---

**Require:** Problem  $\mathbf{c}$ , learning rate  $\eta$ , pre-trained model  $\pi$ , reward threshold  $\tau$ , fraction ratio  $\rho$ , max iterations  $K$

```

 $\mathbf{x}, \mathbf{z} \leftarrow \pi(\mathbf{x} \mid \mathbf{c})$                                  $\triangleright$  Initialize latent representations with CoT: Equation (1).
 $r \leftarrow R(\mathbf{x}, \mathbf{c})$                                  $\triangleright$  Self Reward with Equation (8).
 $T \leftarrow |\mathbf{x}|$ 
 $\mathbf{z} \leftarrow [z_1, z_2, \dots, z_{\rho T}]$                                  $\triangleright$  Keep a  $\rho$  fraction, such as 20%
 $\mathbf{while}$   $k \leq K$  and  $r \leq \tau$   $\mathbf{do}$                                  $\triangleright K$  is typically 10
     $\mathbf{z} \leftarrow \mathbf{z} + \eta \nabla_{\mathbf{z}} \mathcal{J}(\mathbf{z})$                                  $\triangleright$  Update  $\nabla_{\mathbf{z}} \mathcal{J}(\mathbf{z})$ : Equations (6) and (7).
     $\mathbf{x} \sim \pi(\mathbf{x} \mid \mathbf{z}, \mathbf{c})$                                  $\triangleright$  Sample sequence: Equation (4).
     $r \leftarrow R(\mathbf{x}, \mathbf{c})$                                  $\triangleright$  Self Reward with Equation (8).
 $\mathbf{end while}$ 
 $\mathbf{return}$   $\tilde{\mathbf{x}}$ 

```

---

The LATENTSEEK algorithm is summarized in Algorithm 1. At a high level, it performs an instance-level, gradient-guided search over a latent space by iteratively refining latent representations and querying a self-rewarding mechanism. Concretely, at iteration  $k$ , the current latents  $\mathbf{z}$  are decoded to a full sequence  $\mathbf{x}$  through a sampling method. Conditioned on  $\mathbf{x}$  and the instance context  $\mathbf{c}$ , a scalar reward is then obtained from a self-rewarding mechanism:

$$R(\mathbf{x}, \mathbf{c}) \sim \pi(\cdot \mid \mathbf{x}, \mathbf{c}, \text{prompt}_{\text{self-reward}}). \quad (8)$$

This reward signal drives a gradient-guided refinement step that updates the latent representations, thereby biasing subsequent decodes toward higher-reward reasoning paths. The process runs for a small number of iterations (typically less than 3), stopping early if the reward exceeds a threshold.

**Enhancing Techniques.** To ensure more stable and efficient optimization, we propose the integration of two complementary techniques. First, *CoT Initialization*: The initial latent representation is derived from the CoT reasoning sequence. This approach leverages the reasoning capabilities of CoT to establish an effective starting point for subsequent optimization. Second, *Fractional Sequence Optimization*: Instead of optimizing the entire sequence of latent representations  $\mathbf{z} = [z_1, z_2, \dots, z_T]$ , we propose to optimize only a subsequence  $[z_1, z_2, \dots, z_{\rho T}]$ , where  $\rho \in (0, 1]$  is a hyperparameter. The hyperparameter  $\rho$  must strike a balance between two competing objectives: maintaining adequate representational capacity to support effective exploration, and limiting the number of latent representations being updated. Excessive modification of latent representations—especially when their decoded outputs lack semantic coherence—can compromise the reliability of the reward function.

## 3 EMPIRICAL RESULTS

### 3.1 EXPERIMENTAL SETUP

In this subsection, we list our experimental setup. See Sections F.1 to F.3 and F.5 to F.7 for more details.

**Reward Mechanism** We employ the mathematical reasoning prompts introduced by Lifshitz et al. (2025) to elicit the model’s self-reward computation. The reward prompts remain consistent across all

216 Table 1: Accuracy results. **Bold**: best performance, Underlined: second-best. Numbers highlighted  
 217 in red indicate the performance gap relative to the CoT method. Each rollout baseline uses a rollout  
 218 number comparable to LATENTSEEK’s update iterations for fairness. See Section E.1 for larger-  
 219 number comparisons, Section E.3 for token consumption, Section E.8 for performance trends across  
 220 different fraction ratios, and Figure 3 for results of Qwen3 on AIME2024.  $\dagger$ : requires training.

222	Model	Qwen2 7B	Qwen2.5 1.5B	Qwen2.5 7B	14B	LLaMA3.1 8B	Avg
223	Methods	GSM8K: Prompt 1					
224	CoT	68.01	68.08	88.86	92.03	50.19	73.43
225	BoN	<u>79.76</u>	68.31	89.08	<u>92.27</u>	<u>72.93</u>	<u>80.47</u>
226	Self-Reflection (Shinn et al., 2023)	67.85	68.84	88.48	92.21	52.16	73.91
227	Self-Consistency	73.24	<u>69.07</u>	<u>89.23</u>	92.19	55.34	75.81
228	SFT $\dagger$	65.86	49.20	72.55	82.39	40.33	62.07
229	SFT (Magpie 25K) $\dagger$	76.50	66.48	83.01	90.30	70.81	77.42
230	iCoT $\dagger$ (Deng et al., 2024)	47.54	23.28	41.02	-	47.08	39.73
231	LATENTSEEK (Self)	<b>84.38</b> <sub>+16.37</sub>	<b>70.89</b> <sub>+2.81</sub>	<b>90.14</b> <sub>+1.28</sub>	<b>92.49</b> <sub>+0.46</sub>	<b>78.54</b> <sub>+28.35</sub>	<b>83.29</b> <sub>+9.86</sub>
232	Methods	GSM8K: Prompt 2					
233	CoT	65.20	15.31	66.41	91.81	69.07	61.56
234	BoN	61.33	6.14	<u>74.04</u>	92.27	75.97	61.95
235	Self-Reflection (Shinn et al., 2023)	<u>68.16</u>	<u>27.98</u>	<u>72.63</u>	91.96	<u>76.22</u>	<u>67.39</u>
236	Self-Consistency	64.37	8.79	69.22	<u>92.49</u>	74.30	61.83
237	LATENTSEEK (Self)	<b>80.21</b> <sub>+15.01</sub>	<b>44.20</b> <sub>+28.89</sub>	<b>85.06</b> <sub>+18.65</sub>	<b>92.72</b> <sub>+0.91</sub>	<b>83.70</b> <sub>+14.63</sub>	<b>77.18</b> <sub>+15.62</sub>
238	Methods	MATH-500: Prompt 1					
239	CoT	51.40	54.80	72.80	77.20	47.60	60.76
240	BoN	53.40	47.40	<u>75.40</u>	<u>78.80</u>	<u>51.20</u>	<u>61.24</u>
241	Self-Reflection (Shinn et al., 2023)	53.00	53.00	72.40	75.80	45.40	59.92
242	Self-Consistency	<u>53.60</u>	51.60	73.20	77.40	50.40	61.24
243	SFT (Magpie 25K) $\dagger$	46.60	44.40	55.40	68.20	31.00	49.12
244	LATENTSEEK (Self)	<b>57.40</b> <sub>+6.00</sub>	<b>55.60</b> <sub>+0.80</sub>	<b>75.60</b> <sub>+2.80</sub>	<b>80.00</b> <sub>+2.80</sub>	<b>54.60</b> <sub>+7.00</sub>	<b>64.64</b> <sub>+3.88</sub>
245	Methods	MATH-500: Prompt 2					
246	CoT	37.40	29.40	53.80	68.00	40.40	45.80
247	BoN	41.60	29.40	55.80	64.20	44.40	47.08
248	Self-Reflection (Shinn et al., 2023)	43.00	<u>32.20</u>	55.60	70.00	35.20	<u>47.20</u>
249	Self-Consistency	<u>43.20</u>	17.20	<u>57.00</u>	65.80	<u>45.20</u>	45.68
250	LATENTSEEK (Self)	<b>44.80</b> <sub>+7.40</sub>	<b>34.40</b> <sub>+5.00</sub>	<b>57.60</b> <sub>+3.80</sub>	<b>71.00</b> <sub>+3.00</sub>	<b>47.00</b> <sub>+6.60</sub>	<b>50.96</b> <sub>+5.16</sub>
251	Methods	AIME2024: Prompt1					
252	CoT	0.00	3.33	6.67	10.00	0.00	4.00
253	BoN	0.00	0.00	<u>10.00</u>	<b>16.67</b>	0.00	<u>5.33</u>
254	Self-Reflection (Shinn et al., 2023)	0.00	<b>6.67</b>	6.67	10.00	0.00	4.67
255	Self-Consistency	<b>3.33</b>	0.00	3.33	13.33	<u>6.67</u>	<u>5.33</u>
256	SFT (Magpie 25K) $\dagger$	<b>3.33</b>	0.00	3.33	10.00	3.33	4.00
257	LATENTSEEK (Self)	<b>3.33</b> <sub>+3.33</sub>	<b>6.67</b> <sub>+3.33</sub>	<b>13.33</b> <sub>+6.67</sub>	<b>16.67</b> <sub>+6.67</sub>	<b>10.00</b> <sub>+10.00</sub>	<b>10.00</b> <sub>+6.00</sub>
258	Methods	AIME2024: Prompt2					
259	CoT	0.00	0.00	0.00	3.33	0.00	0.67
260	BoN	<b>3.33</b>	0.00	<u>6.67</u>	<b>10.00</b>	<b>6.67</b>	<u>5.33</u>
261	Self-Reflection (Shinn et al., 2023)	0.00	<b>3.33</b>	0.00	6.67	<b>6.67</b>	3.33
262	Self-Consistency	<b>3.33</b>	0.00	3.33	3.33	0.00	2.00
263	LATENTSEEK (Self)	<b>3.33</b> <sub>+3.33</sub>	<b>3.33</b> <sub>+3.33</sub>	<b>13.33</b> <sub>+13.33</sub>	<b>10.00</b> <sub>+6.67</sub>	<b>6.67</b> <sub>+6.67</sub>	<b>7.33</b> <sub>+6.67</sub>

257 **datasets.** For some models on specific tasks, we further incorporate a format-based reward (DeepSeek-  
 258 AI, 2025) to enhance their instruction-following capability. To further analysis the potential of our  
 259 paradigm, we introduce a Perfect Sharp Reward Model (PSRM), the details of which are discussed in  
 260 §3.3. All generated reward is chosen to be a number between -1 and 0.

261 **Prompt Designation** For robustness consideration, we use two prompts for evaluation, with the  
 262 first requiring a wrapped answer with  $\boxed{\{\cdot\}}$  (Yang et al., 2024; Team, 2024a) (Prompt 1) and the  
 263 second asking to format the answer as a json (Prompt 2).

264 **Backbones** To evaluate the generalizability of our approach, we conduct experiments with pre-  
 265 trained LLMs of different families and sizes: Qwen2-7B-Instruct, Qwen2.5-1.5B, 7B, 14B-Instruct  
 266 (Yang et al., 2024; Team, 2024a), LLaMA3.1-8B-Instruct (Team, 2024b).

267 **Benchmarks** Following Deng et al. (2024); Liu et al. (2025), we focus on mathematical reasoning  
 268 for evaluation. We evaluate on two standard benchmarks for reasoning, GSM8K (Cobbe et al., 2021)  
 269 and MATH-500 (Hendrycks et al., 2021), and a harder dataset AIME2024.

270 Table 2: Accuracy score compared with more baseline methods on GSM8K and MATH-500 datasets  
 271 with Llama3.1-8B as backbone. The best performances are highlighted in bold.  
 272

273 Methods	274   Supervision	275 Train	276 Backbone	277 GSM8K	278 MATH-500	279 Average
274 CoT	275 -	276 <b>X</b>	277 Instruct	278 69.1	279 47.6	280 58.3
274 Few-Shot CoT (Lambert et al., 2025)	275 -	276 <b>X</b>	277 Instruct	278 83.4	279 42.5	280 63.0
274 BoN	275 Self	276 <b>X</b>	277 Instruct	278 76.0	279 51.2	280 63.6
274 Self-Rewarding (Yuan et al., 2025)	275 Self	276 <b>✓</b>	277 Instruct	278 76.0	279 30.2	280 53.1
274 Self-Reflection (Shinn et al., 2023)	275 Self	276 <b>X</b>	277 Instruct	278 76.2	279 45.4	280 60.8
274 Self-Consistency	275 Self	276 <b>X</b>	277 Instruct	278 55.3	279 48.2	280 51.8
274 ScPO (Prasad et al., 2024)	275 Self	276 <b>✓</b>	277 Instruct	278 71.1	279 31.0	280 51.1
274 CoH (Liu et al., 2023a)	275 Self	276 <b>✓</b>	277 Instruct	278 74.4	279 32.3	280 53.3
274 Genius (Xu et al., 2025)	275 Self	276 <b>✓</b>	277 Instruct	278 78.3	279 34.6	280 56.5
274 SPIN (Chen et al., 2024a)	275 Data	276 <b>✓</b>	277 Instruct	278 74.9	279 31.5	280 53.2
274 iCoT (Deng et al., 2024)	275 Data	276 <b>✓</b>	277 Instruct	278 47.1	279 -	280 47.1
274 SFT (Magpie 25K)	275 Data	276 <b>✓</b>	277 Instruct	278 70.8	279 31.0	280 50.9
274 GRPO (GSM8K Train)	275 Data	276 <b>✓</b>	277 Instruct	278 -	279 50.2	280 50.2
274 SimpleRL-Zoo (Zeng et al., 2025)	275 Data	276 <b>✓</b>	277 Base	278 79.2	279 23.0	280 51.1
274 <b>LATENTSEEK</b>	275 Self	276 <b>X</b>	277 Instruct	278 <b>83.7</b>	279 <b>54.6</b>	280 <b>69.2</b>

288 **Baselines** We compare our methods against several established baselines:  
 289

- 290 • **Prompting (Training-Free)**: CoT (Wei et al., 2022) and Few-Shot CoT (Lambert et al., 2025).
- 291 • **Explicit Search (Training-Free)**: Best-of-N (BoN) represents a highly effective search strategy, as  
 292 demonstrated by Liu et al. (2025). Self-Reflection (Shinn et al., 2023) is a search strategy imitating  
 293 humans’ reasoning pattern. Self-Consistency, also named majority vote, is a search mechanism in  
 294 explicit space that stems from a multi-agent system.
- 295 • **Reinforcement Learning**: (1) Self Reward: Self-Rewarding (Yuan et al., 2025), ScPO (Prasad et al.,  
 296 2024), CoH (Liu et al., 2023a), and Genius (Xu et al., 2025). (2) Verifiable Reward: SimpleRL-Zoo  
 297 (Zeng et al., 2025), GRPO (GSM8K train set), and SPIN (Chen et al., 2024a).
- 298 • **Latent Chain-of-Thought**: Although iCoT (Deng et al., 2024) requires an augmented training  
 299 dataset for GSM8K (Cobbe et al., 2021), it remains a canonical example of the latent CoT.
- 300 • **Supervised Fine-Tuning (SFT)**: Following (Xu et al., 2025), we apply SFT on Magpie 25K. For  
 301 GSM8K, we also report SFT performance using its provided training set.

303 **Sampling Method** We use greedy decoding to sample from LATENTSEEK, as it is more computa-  
 304 tionally efficient and more robust. (See Section E.4 for an experimental comparison).

### 306 3.2 STATE-OF-THE-ART TEST-TIME REASONING PERFORMANCE

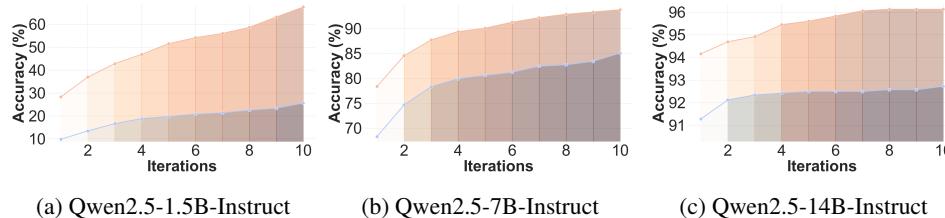
308 The main experimental results are presented in Table 1 and Table 2.

309 **Best Performance on GSM8K (Cobbe et al., 2021) and MATH-500 (Hendrycks et al., 2021).**  
 310 As shown in Table 1, LATENTSEEK surpasses every baseline across all 30 settings and achieves the  
 311 best average performance over all datasets. When averaged across backbones, the improvement over  
 312 BoN is **15.23** points on GSM8K and against *Self-Reflection* is **4.72** points on MATH-500. Compared  
 313 with *Self-Consistency*, the gap is **15.35** on GSM8K and **5.28** on MATH-500. The Qwen2.5 series  
 314 was explicitly trained with Prompt-1-distilled data (Team, 2024a), yielding strong *CoT* performance;  
 315 nevertheless, LATENTSEEK achieves further improvements and attains the highest scores across  
 316 baselines. Using LLaMA3.1 as the backbone, as shown in Table 2, we compare against a broader  
 317 set of baselines. On GSM8K, our method surpasses the *CoT* baseline by **14.6** points and *BoN* by **7.7**  
 318 points; on MATH-500, the improvements are **7.0** and **3.4** points, respectively. It also outperforms the  
 319 self-reward RL method *Genius* (Xu et al., 2025) by **5.4** points on GSM8K and **20.0** points on MATH-  
 320 500. Even compared with verifiable-feedback RL methods such as *SimpleRL-Zoo* (Zeng et al., 2025),  
 321 trained on the base model, LATENTSEEK leads by an average of **18.1** points across the two datasets.

322 **Robustness Across Backbones.** The robustness of our method across backbones can be evaluated  
 323 along two axes: (1) *Model Families*: LATENTSEEK delivers best results across different models.  
 Focusing on 7-8B models and using the same averaging protocol over GSM8K, MATH-500, and

324 Table 3: Accuracy results with Perfect Sharp Reward Model (PSRM). *Self*: the self-reward mechanism.  
 325 The reported score is the average of both prompts.  
 326

Reward	Qwen2 7B	1.5B	Qwen2.5 7B	14B	LLaMA3.1 8B	Avg
<i>GSM8K</i>						
LATENTSEEK- <i>Self</i>	82.3	57.5	87.6	92.6	81.1	80.2
LATENTSEEK- <i>Perfect Sharp Reward Model</i>	92.8 <sub>+10.5</sub>	76.5 <sub>+19.0</sub>	93.9 <sub>+6.3</sub>	96.0 <sub>+3.4</sub>	91.7 <sub>+10.6</sub>	90.2 <sub>+10.0</sub>
<i>MATH-500</i>						
LATENTSEEK- <i>Self</i>	51.1	45.0	66.6	75.5	50.8	57.8
LATENTSEEK- <i>Perfect Sharp Reward Model</i>	71.8 <sub>+20.7</sub>	66.7 <sub>+21.7</sub>	81.8 <sub>+15.2</sub>	86.6 <sub>+11.1</sub>	70.5 <sub>+19.7</sub>	75.5 <sub>+17.7</sub>
<i>AIME2024</i>						
LATENTSEEK- <i>Self</i>	3.3	5.0	13.3	13.3	8.3	8.6
LATENTSEEK- <i>Perfect Sharp Reward Model</i>	8.3 <sub>+5.0</sub>	6.7 <sub>+1.7</sub>	15.0 <sub>+1.7</sub>	25.0 <sub>+11.7</sub>	10.0 <sub>+1.7</sub>	13.0 <sub>+4.4</sub>



344 Figure 2: GSM8K(Cobbe et al., 2021) Prompt 2 Accuracy changes with respect to the increasing  
 345 number of iterations. Orange: Perfect Sharp Reward Model. Blue: Self Reward Model.  
 346

347  
 348 both prompts, LATENTSEEK surpasses *BoN* by **7.67** points on Qwen2, exceeds *Self-Reflection* by  
 349 **4.82** on Qwen2.5, and improves over *Self-Consistency* by **9.65** on LLaMA3.1. (2) *Model Size*: At  
 350 1.5B parameters, LATENTSEEK surpasses *BoN* on GSM8K by **28.89** points and at 14B parameters,  
 351 the margin on MATH-500 attains **6.80** points. These results provide robust evidence of our method’s  
 352 robustness across diverse model families and scale.

353  
 354 **Challenging Problems — AIME2024.** AIME2024 results appear in Table 1. Our method consis-  
 355 tently outperforms all baselines, achieving an average gain of **6.34** points over *CoT* across model  
 356 families and prompt configurations. The largest improvement occurs on AIME2024 with Prompt  
 357 2, where our approach exceeds *CoT* by **13.33** points and outperforms *BoN* by **6.67** points using  
 358 Qwen2.5-7B-Instruct. Averaged over backbones, LATENTSEEK also surpasses *BoN* by **4.67** points,  
 359 *Self-Reflection* by **5.33** points, and *Self-Consistency* by **4.67** points, indicating strong effectiveness in  
 360 more complex settings. For the performance of a stronger model (Qwen3-4B-Instruct-2507 (Yang  
 361 et al., 2025)) on AIME2024, please refer to Section E.2.

### 362 3.3 IDEAL EXPERIMENT: PERFECT SHARP REWARD MODEL

363  
 364 To inspect the value of exploration in latent space, we evaluate a Perfect Sharp Reward Model (PSRM)  
 365 that returns 0 only when the final answer exactly matches the ground truth and  $-1$  otherwise. This  
 366 yields an all-or-nothing signal: until the correct answer is reached, every step receives identical  
 367 feedback. Results (Table 3) show that, despite this extreme sharpness, the PSRM still delivers  
 368 substantial improvements, outperforming the self-reward variant by an average of 10.67 points across  
 369 all evaluated settings. Conceptually, this setup reduces optimization to maximizing the success rate  
 370 under a near-constant reward landscape; the search carries almost no directional information and is  
 371 therefore close to blind exploration. This demonstrate that **exploration in latent space alone can**  
 372 **yield meaningful performance improvements, even under sharp feedback.**

373  
 374 **Small Model and Large Iterations.** Following Liu et al. (2025), we analyze the small Qwen2.5-  
 375 1.5B-Instruct (Team, 2024a) under a large iteration budget to probe the limits of latent-space  
 376 exploration. As shown in Table 4, Qwen2.5-1.5B-Instruct yields a **14-point** gain over GPT-4o on  
 377 AIME2024. On MATH-500, it achieves a high accuracy score, trailing  $\text{\textcircled{1}}$ -preview by only  
 378 **2.7** points. These results demonstrate that **even a 1.5B-parameter model’s latent space is highly**  
 379 **expressive and that LATENTSEEK can be an effective mechanism for eliciting this expressivity.**

Table 4: Performance of Extreme Scaling on MATH-500 (Hendrycks et al., 2021) and AIME2024. Setting the maximum update iteration to 256.  $K$ : average number of outputs or iterations.

Model	Size	Type	MATH-500		AIME2024	
			Acc	K	Acc	K
GPT-4o	-	CoT	74.6	-	9.3	-
o1-preview	-	CoT	85.5	-	44.6	-
Qwen2.5-1.5B-Inst.	1.5B	TTS ( <a href="#">Liu et al., 2025</a> )	81.8	256.0	20.0	256.0
<b>LATENTSEEK (PSRM)</b> (Qwen2.5-1.5B-Inst.)	1.5B	Latent Search	82.8	61.8	23.3	211.8

### 3.4 TEST-TIME SCALING: SCALING TREND ANALYSIS OF LATENTSEEK

In addition to increasing the number of generated tokens at test time (Muennighoff et al., 2025; Snell et al., 2025), we propose an alternative approach to test-time scaling: increasing the number of LATENTSEEK iterations. The relationship between model performance on GSM8K (Cobbe et al., 2021) and the number of iterations is illustrated in Figure 2. As shown, an ideal reward model demonstrates strong performance and yields a consistently improving trend with more iterations. In contrast, the self-reward method exhibits rapid initial gains followed by a plateau, a behavior we attribute to the reward model’s limited accuracy. Notably, as the number of iterations increases further, performance eventually surpasses this plateau and continues to improve. The above shows that **test-time scaling remains attainable even under a sharply peaked reward**. It further suggests that, given an appropriate reward model, **searching through the latent space offers a promising new direction for test-time scaling**.

### 3.5 COMPARISON WITH UNGUIDED SEARCH

We compare LATENTSEEK with a single-step stochastic explorative search in latent space on the GSM8K (Cobbe et al., 2021), to assess the effectiveness of our gradient-guided search. In this baseline, instead of performing iterative gradient ascent, a Gaussian noise is added once to the initial latent representations,

$$\tilde{z} \leftarrow z + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma^2 I)$$

, and evaluate three noise levels  $\sigma^2 = 0.5, 0.75$ , and  $1.0$ . As shown in Table 5, LATENTSEEK outperforms the stochastic exploration baselines by an average of **13.66 points**, indicating that the gains arise from effective gradient guidance rather than random perturbations.

### 3.6 ANALYSIS OF SELF-RWARD MECHANISM

To understand the role of the self-reward mechanism, we evaluate the accuracy of the self-reward mechanism on the GSM8K test set, in which the generation uses the prompt 2 (the json prompt). Specifically, we collect *all explored sequences*, including intermediate reasoning steps as well as final generated answers, and evaluate the accuracy. We binarized our dense  $[-1, 0]$  rewards into 0 and 1.

Table 5: Comparison with Unguided Search on GSM8K. *SE*: Stochastic Exploration.

Model	Qwen2 7B	Qwen2.5 7B	Qwen2.5 14B	LLaMA3.1 8B	Avg
<i>GSM8K: Prompt 1</i>					
SE ( $\sigma^2 = 0.50$ )	66.19	64.22	88.60	92.33	49.73
SE ( $\sigma^2 = 0.75$ )	65.96	63.53	88.60	92.03	49.73
SE ( $\sigma^2 = 1.00$ )	65.13	63.76	89.08	91.88	50.57
<b>LATENTSEEK</b>	<b>84.38</b>	<b>70.89</b>	<b>90.14</b>	<b>92.49</b>	<b>78.54</b>
<i>GSM8K: Prompt 2</i>					
SE ( $\sigma^2 = 0.50$ )	64.32	16.68	65.60	92.04	67.85
SE ( $\sigma^2 = 0.75$ )	64.77	15.69	64.92	91.89	67.02
SE ( $\sigma^2 = 1.00$ )	63.84	16.07	66.34	92.12	66.72
<b>LATENTSEEK</b>	<b>80.21</b>	<b>44.20</b>	<b>85.06</b>	<b>92.72</b>	<b>83.70</b>
<b>77.18</b>					

Table 6: Self-Reward Mechanism Percentage Accuracy on **CSM8K**

Backbone	Qwen2		Qwen2.5		LLaMA3.1		Avg
	7B	1.5B	7B	14B	8B		
Acc	85.78	77.15	80.17	89.83	80.46		82.68

432  $\leq -0.2$ ) and compared this against the ground-truth correctness. This yields the results in Table 6.  
 433 The table demonstrates that in approximately 80% of the cases, the self-reward mechanism is reliable.  
 434 In practice, the ability to judge an answer cooperates with the ability to generate one, and together  
 435 they lead to substantially improved overall performance.

436 Furthermore, we replace our self-  
 437 reward mechanism with a constant  
 438 negative reward baseline to isolate the  
 439 effect of reward guidance, assigning a  
 440 **constant reward of -1 at every step.**  
 441 The experiment is performed on the  
 442 same condition as Table 6. A nega-  
 443 tive reward is used because any non-  
 444 negative reward would simply reinforce the current tokens under argmax decoding and hinder  
 445 exploration. As reported in Table 7, this ablation causes **an average performance drop of 14.21**  
 446 **points**(eg., 85.06% to 64.52% for Qwen2.5-7B-Instruct), demonstrating that **our self-reward guid-**  
 447 **ance is essential for effective optimization.**

### 448 3.7 QUALITATIVE ANALYSIS

449 An analysis of the LATENTSEEK’s  
 450 generated responses offers insight  
 451 into its reasoning processes, with a  
 452 representative example detailed in Ta-  
 453 ble 8. These responses often contain  
 454 nonsensical or obscure tokens—such  
 455 as “andLet”, “on80ward)”, and  
 456 “BothThe”—as well as syntactically  
 457 absurd phrases like “Let find this”  
 458 and “denote the common”. This phe-  
 459 nomenon indicates that the reasoning  
 460 processes of pre-trained language  
 461 models can diverge significantly from  
 462 human cognition, even when they  
 463 produce a correct final output. Conse-  
 464 quently, this suggests that **optimal reasoning paths for**  
 465 **language models may not align with human strategies, and thus may be more effectively sought**  
 466 **within the latent space**, which is native to LLM and therefore also a natural choice. A more detailed  
 467 qualitative analysis and additional cases are presented in Section I.

### 468 3.8 ALGORITHMIC STATISTICS

469 Table 9 reports two key statistical fea-  
 470 tures: (1) the ratio of final answer  
 471 tokens to the original CoT tokens,  
 472 and (2) the average number of update  
 473 iterations across the entire dataset.  
 474 The ratio of the final answer’s to-  
 475 ken count to that of the original CoT  
 476 rarely exceeds 1.1, confirming that  
 477 performance gains are not achieved  
 478 through verbose generation. More-  
 479 over, the method converges rapidly,  
 480 requiring an average of fewer than  
 481 two update iterations for average-level  
 482 questions. This highlights the compu-  
 483 tational efficiency and rapid conver-  
 484 gence behavior of our method.

485 To further demonstrate the value of shifting computation from training to test time, we compare  
 Floating Point Operations (FLOPs) against Genius (Xu et al., 2025), a strong RL-based self-rewarding

Table 7: Comparison with Constant Reward baseline

Backbone	Qwen2 7B	1.5B	Qwen2.5 7B	14B	LLaMA3.1 8B	Avg
LATENTSEEK (Constant)	60.19	29.49	64.52	90.67	69.98	62.97
LATENTSEEK (Self)	<b>80.21</b>	<b>44.20</b>	<b>85.06</b>	<b>92.72</b>	<b>83.70</b>	<b>77.18</b>
$\Delta$	+20.02	+14.71	+20.54	+2.05	+13.72	+14.21

Table 8: Case Study. Latent optimized tokens are in blue. Please refer to Section I for more cases.

<b>QUESTION:</b> Two trains leave San Rafael at the same time [...]
What's the distance covered by each train in the two days?
<b>ANSWER:</b> 230
<b>LATENTSEEK:</b> Let find this,, let'll more understand it down step two andLet 1: BothThe the trains are same route west both first time on80ward) on same and and can consider they the travel travel same distance of So's denote the common xd' they both 80 miles on x have write up an following: [...] [...] Since both trains travel the same distance in each direction, the distance covered by each train is 230 miles. The final answer is: \boxed{230}

467  
 468 Consequently, this suggests that **optimal reasoning paths for**  
 469 **language models may not align with human strategies, and thus may be more effectively sought**  
 470 **within the latent space**, which is native to LLM and therefore also a natural choice. A more detailed  
 471 qualitative analysis and additional cases are presented in Section I.

Table 9: Algorithmic Statistics: (1) The ratio of answer length to CoT length. (2) Average update iterations.

Model	Qwen2 7B	1.5B	Qwen2.5 7B	14B	LLaMA3.1 8B	Avg
<i>GSM8K: Prompt 1</i>						
Answer / CoT	0.97	0.99	1.00	1.00	0.94	0.98
Avg.# Iter	0.94	0.75	0.24	0.05	1.66	0.72
<i>GSM8K: Prompt 2</i>						
Answer / CoT	1.06	3.80	1.08	1.00	1.02	1.59
Avg.# Iter	0.55	4.59	1.52	0.14	0.58	1.48
<i>MATH-500: Prompt 1</i>						
Answer / CoT	0.92	0.97	0.99	0.99	0.92	0.96
Avg.# Iter	2.09	1.93	1.10	0.62	2.35	1.62
<i>MATH-500: Prompt 2</i>						
Answer / CoT	1.01	0.94	0.98	0.97	0.90	0.96
Avg.# Iter	0.72	2.19	0.98	0.49	1.65	1.21

baseline. As shown in Table 2, LATENTSEEK outperforms Genius by 12.7% on GSM8K using LLaMA3.1-8B as the backbone. For this model, a single forward pass costs approximately  $2.29 \times 10^{11}$  FLOPs. Crucially, Genius requires full backpropagation through the entire model to update parameters, whereas LATENTSEEK backpropagates only through the language model, incurring about  $1.05 \times 10^9$  FLOPs. Following our calculations, LATENTSEEK remains more efficient for processing up to approximately  $1.94 \times 10^5$  inference instances when compared to the total cost of the Genius framework. This threshold notably exceeds the 100k examples in Genius’s training set, underscoring the significant practical efficiency of our approach. Please refer to Section G for detailed calculation.

## 4 RELATED WORK

**Reasoning in Language Models.** Recent advances in reasoning capabilities of large language models (Brown et al., 2020; OpenAI, 2023) have been enhanced through prompting techniques. CoT prompting (Wei et al., 2022; Kojima et al., 2022; Zhou et al., 2022) encourages models to generate intermediate reasoning steps. Unlike these static approaches, our method dynamically optimizes the reasoning process for each problem instance. Compute-optimal scaling (Snell et al., 2025; Misaki et al., 2025) adaptively adjusts inference strategies based on task complexity. Latent reasoning methods (Hao et al., 2024; Shen et al., 2025; Cheng and Van Durme, 2024; Deng et al., 2024; Chen et al., 2024b) replace explicit text-based reasoning with continuous representations. The broader field of learning to reason includes techniques like process supervision (Uesato et al., 2022) and self-critique (Huang et al., 2022).

**Reinforcement Learning for Language Models.** The integration of Reinforcement Learning and LLM starts from the realm of Human Feedback (RLHF) (Ouyang et al., 2022; Bai et al., 2022; Rafailov et al., 2023), which commonly employs algorithms such as Proximal Policy Optimization (PPO) (Schulman et al., 2017). Recent advancements (Liu et al., 2023b; Guo et al., 2025) specifically address reasoning tasks with RL techniques. Innovations in reward modeling have explored human preference-based architectures (Schulman et al., 2017; Hazra et al., 2024), automated language model-driven design (Kwon et al., 2023), and multi-agent verification frameworks (Lifshitz et al., 2025).

**Controllable Generation and Test-Time Optimization.** Various approaches have been developed for controlling language model outputs, including conditioning on control codes (Keskar et al., 2019), gradient-based steering (Dathathri et al., 2019), and prompt optimization (Qin et al., 2023). At test time, techniques (Sun et al., 2023; Zhang et al., 2025; Wang et al., 2022) improve outputs through sampling and selection. Some Test-Time Training(TTT) framewroks (Sun et al., 2020; 2024; Hardt and Sun, 2023) integrate self-supervised objectives for online model updates. Our work differs from test-time planning approaches (Hao et al., 2023) by optimizing in a continuous latent space rather than performing a discrete search.

**Prompt Tuning and Soft Prompt.** Prompt Tuning adapts language models by prepending trainable vectors to inputs or hidden states (Lester et al., 2021; Liu et al., 2024a; Li and Liang, 2021; Liu et al., 2021). However, both require labeled data and full backpropagation, incurring high computational cost. In contrast, our method leverages latent-space manipulation without training data or model updates, enabling efficient, flexible adaptation.

## 5 CONCLUSION

In conclusion, the LATENTSEEK framework introduces a novel and efficient approach to enhancing reasoning capabilities in LLMs by leveraging TTIA in the latent space. By optimizing latent representations through policy gradient, LATENTSEEK circumvents the need for parameter updates, offering an alternative to methods that require substantial retraining or reinforcement learning. Empirical results across multiple reasoning benchmarks consistently demonstrate the superior performance of LATENTSEEK compared to existing baselines, such as CoT, BoN, and reinforcement learning-based techniques. Furthermore, the framework proves to be computationally efficient, with rapid convergence for average-level problems. This work also demonstrates a new possible avenue for test-time scaling in the latent space. Ultimately, LATENTSEEK represents a significant step forward in advancing LLMs in the realm of TTIA reasoning. Please refer to Section A for discussions and future works.

540  
541  
**ETHICS STATEMENT**

542 This work focuses on improving the reasoning capabilities of large language models (LLMs). The  
 543 research is methodological in nature and does not involve human subjects, sensitive data, or de-  
 544 ployment in real-world applications. Our contributions are confined to improving core algorithmic  
 545 aspects of LLM reasoning and do not introduce new data that could raise concerns regarding privacy,  
 546 bias, or misuse. While we recognize that LLMs can have broader societal impacts, particularly  
 547 when used in downstream applications, our work does not directly engage with these deployment  
 548 scenarios. We also note that enhanced reasoning capabilities may indirectly influence downstream  
 549 model behavior. However, the improvements described in this paper are academic-purpose and do not  
 550 facilitate manipulation, deception, or unethical use of LLMs. Overall, we believe that our research  
 551 poses no direct ethical or societal risks and is aligned with the responsible development of trustworthy  
 552 AI systems.

553  
554 **REPRODUCIBILITY STATEMENT**  
555

556 The findings presented in this paper are supported by a detailed disclosure of our methodology,  
 557 designed to enable full reproducibility. The core methodology and algorithmic formulation of our  
 558 approach are presented in §2. Our full experimental protocol, which covers the datasets, evaluation  
 559 benchmarks, and baselines, is detailed in §3. All requisite implementation details for replication,  
 560 including model backbones and hyperparameters, are documented in Section F. Taken together, the  
 561 paper and its appendices provide a complete blueprint for reproducing our work. For the statement on  
 562 our use of LLMs, please see Section K. The code is attached to the submission.

563  
564 **REFERENCES**

- 565 Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo  
 566 Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint*  
 567 *arXiv:2303.08774*, 2023.
- 568 Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al.  
 569 Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information*  
 570 *Processing Systems (NeurIPS)*, 35:24824–24837, 2022.
- 571 Ut Na Sio and Thomas C Ormerod. Does incubation enhance problem solving? a meta-analytic review. *Psychol*  
 572 *Bull*, 135(1):94–120, January 2009.
- 573 Simone M Ritter and Ap Dijksterhuis. Creativity-the unconscious foundations of the incubation period. *Front*  
 574 *Hum Neurosci*, 8:215, April 2014.
- 575 Asael Y. Sklar, Nir Levy, Ariel Goldstein, Roi Mandel, Anat Maril, and Ran R. Hassin. Reading and do-  
 576 ing arithmetic nonconsciously. *Proceedings of the National Academy of Sciences*, 109(48):19614–19619,  
 577 2012. doi: 10.1073/pnas.1211645109. URL <https://www.pnas.org/doi/abs/10.1073/pnas.1211645109>.
- 578 Yuntian Deng, Yejin Choi, and Stuart Shieber. From explicit cot to implicit cot: Learning to internalize cot step  
 579 by step. *CoRR*, 2024.
- 580 Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training  
 581 large language models to reason in a continuous latent space. *CoRR*, 2024.
- 582 Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang,  
 583 Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with  
 584 human feedback. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:27730–27744, 2022.
- 585 DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL  
 586 <https://arxiv.org/abs/2501.12948>.
- 587 Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning.  
 588 *Machine learning*, 8:229–256, 1992.
- 589 Fangzhi Xu, Hang Yan, Chang Ma, Haiteng Zhao, Qiushi Sun, Kanzhi Cheng, Junxian He, Jun Liu, and Zhiyong  
 590 Wu. Genius: A generalizable and purely unsupervised self-training framework for advanced reasoning. *CoRR*,  
 591 2025.

- 594 Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser,  
 595 and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach,  
 596 R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*,  
 597 volume 30. Curran Associates, Inc., 2017. URL [https://proceedings.neurips.cc/paper\\_files/paper/2017/file/3f5ee243547dee91fdb053c1c4a845aa-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fdb053c1c4a845aa-Paper.pdf).
- 599 Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language  
 600 agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–  
 601 8652, 2023.
- 602 Shalev Lifshitz, Sheila A McIlraith, and Yilun Du. Multi-agent verification: Scaling test-time compute with  
 603 multiple verifiers. *CoRR*, 2025.
- 604
- 605 An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li,  
 606 Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang,  
 607 Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai  
 608 Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng,  
 609 Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu,  
 610 Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren,  
 611 Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and  
 612 Zhihao Fan. Qwen2 technical report. *CoRR*, 2024.
- 613 Qwen Team. Qwen2.5: A party of foundation models, September 2024a. URL <https://qwenlm.github.io/blog/qwen2.5/>.
- 614
- 615 Llama Team. The llama 3 herd of models, 2024b. URL <https://arxiv.org/abs/2407.21783>.
- 616 Runze Liu, Junqi Gao, Jian Zhao, Kaiyan Zhang, Xiu Li, Biqing Qi, Wanli Ouyang, and Bowen Zhou. Can 1b  
 617 llm surpass 405b llm? rethinking compute-optimal test-time scaling. *CoRR*, 2025.
- 618
- 619 Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert,  
 620 Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to  
 621 solve math word problems. *CoRR*, 2021.
- 622 Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and  
 623 Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. In *Advances in Neural  
 624 Information Processing Systems (NeurIPS)*, 2021.
- 625
- 626 Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester  
 627 James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D.  
 628 Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, Noah A. Smith,  
 629 Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tulu 3: Pushing frontiers in open language model  
 630 post-training, 2025. URL <https://arxiv.org/abs/2411.15124>.
- 631 Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason  
 632 Weston. Self-rewarding language models. *CoRR*, 2025.
- 633 Archiki Prasad, Weizhe Yuan, Richard Yuanzhe Pang, Jing Xu, Maryam Fazel-Zarandi, Mohit Bansal, Sainbayar  
 634 Sukhbaatar, Jason Weston, and Jane Yu. Self-consistency preference optimization, 2024. URL <https://arxiv.org/abs/2411.04109>.
- 635
- 636 Hao Liu, Carmelo Sferrazza, and Pieter Abbeel. Chain of hindsight aligns language models with feedback. In  
 637 *International Conference on Learning Representations (ICLR)*, 2023a.
- 638
- 639 Weihao Zeng, Yuzhen Huang, Qian Liu, Wei Liu, Keqing He, Zejun Ma, and Junxian He. Simplerl-zoo:  
 640 Investigating and taming zero reinforcement learning for open base models in the wild, 2025. URL <https://arxiv.org/abs/2503.18892>.
- 641
- 642 Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. Self-play fine-tuning converts weak  
 643 language models to strong language models, 2024a.
- 644
- 645 An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengan  
 646 Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.
- 647 Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer,  
 648 Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. *CoRR*, 2025.

- 648 Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally  
 649 can be more effective than scaling parameters for reasoning. In *International Conference on Learning*  
 650 *Representations (ICLR)*, 2025.
- 651 Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind  
 652 Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In  
 653 *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 1877–1901, 2020.
- 654 OpenAI. Gpt-4 technical report. *CoRR*, 2023.
- 655 Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models  
 656 are zero-shot reasoners. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.
- 657 Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Olivier  
 658 Bousquet, Quoc Le, and Ed Chi. Least-to-most prompting enables complex reasoning in large language  
 659 models. In *International Conference on Learning Representations (ICLR)*, 2022.
- 660 Kou Misaki, Yuichi Inoue, Yuki Imajuku, So Kuroki, Taishi Nakamura, and Takuya Akiba. Wider or deeper?  
 661 scaling llm inference-time compute with adaptive branching tree search. *CoRR*, 2025.
- 662 Zhenyi Shen, Hanqi Yan, Linhai Zhang, Zhanghao Hu, Yali Du, and Yulan He. Codi: Compressing chain-of-  
 663 thought into continuous space via self-distillation. *CoRR*, 2025.
- 664 Jeffrey Cheng and Benjamin Van Durme. Compressed chain of thought: Efficient reasoning through dense  
 665 representations. *CoRR*, 2024.
- 666 Haolin Chen, Yihao Feng, Zuxin Liu, Weiran Yao, Akshara Prabhakar, Shelby Heinecke, Ricky Ho, Phil Mui,  
 667 Silvio Savarese, Caiming Xiong, and Huan Wang. Language models are hidden reasoners: Unlocking latent  
 668 reasoning capabilities via self-rewarding, 2024b. URL <https://arxiv.org/abs/2411.04282>.
- 669 Jonathan Uesato, Nate Kushman, Ramana Kumar, Michele Catasta, Johan Legrand, Jelena Luketina, Andrew  
 670 Lampinen, Aja Brownsmith, Zoya Bylinskii, Victoria Ellison, et al. Solving math word problems with  
 671 process-based and outcome-based feedback. *CoRR*, 2022.
- 672 Jiayuan Huang, Jierui Kwon, Kevin Cohen, and Nanyun Peng. Language models as inductive reasoners. In  
 673 *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics*  
 674 (*Volume 1: Long Papers*), 2022.
- 675 Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen,  
 676 Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback.  
 677 *CoRR*, 2022.
- 678 Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D Manning, and Chelsea Finn. Direct  
 679 preference optimization: Your language model is secretly a reward model. *Advances in Neural Information*  
 680 *Processing Systems (NeurIPS)*, 36, 2023.
- 681 John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization  
 682 algorithms. *CoRR*, 2017.
- 683 Tianle Liu, Tianyi Zhou, Xiaofei Lin, Percy Liang, and Tao Xiao. Statistical rejection sampling improves  
 684 preference optimization. *International Conference on Learning Representations (ICLR)*, 2023b.
- 685 Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma,  
 686 Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning.  
 687 *CoRR*, 2025.
- 688 Rishi Hazra, Alkis Sykounas, Andreas Persson, Amy Loutfi, and Pedro Zuidberg Dos Martires. Revolve:  
 689 Reward evolution with large language models using human feedback. In *International Conference on Learning*  
 690 *Representations (ICLR)*, 2024.
- 691 Minae Kwon, Sang Michael Xie, Kalesha Bullard, and Dorsa Sadigh. Reward design with language models. In  
 692 *International Conference on Learning Representations (ICLR)*, 2023.
- 693 Nitish Shirish Keskar, Bryan McCann, Lav R Varshney, Caiming Xiong, and Richard Socher. Ctrl: A conditional  
 694 transformer language model for controllable generation. *CoRR*, 2019.
- 695 Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and  
 696 Rosanne Liu. Plug and play language models: A simple approach to controlled text generation. In *International*  
 697 *Conference on Learning Representations (ICLR)*, 2019.

- 702 Yao Qin, Asli Celikyilmaz, Wenjing Li, Jung H Oh, Victoria Lin, Semih Agrawal, Yang Zhou, Singaram Kumar,  
 703 Jing Shen, Mitesh M Khapra, et al. Pomp: Unsupervised controllable generation by optimizing prompts via  
 704 reinforcement learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2023.
- 705 David Sun, Jasmine Jiang, Terry Nguyen, James Tsai, Yizhe Xia, Kyunghyun Cho, Heng Ji, Hugo Larochelle,  
 706 Xiang Yuan, David Simchi-Levi, et al. Recitation-augmented language models. In *International Conference  
 707 on Learning Representations (ICLR)*, 2023.
- 708 Zhaowei Zhang, Fengshuo Bai, Qizhi Chen, Chengdong Ma, Mingzhi Wang, Haoran Sun, Zilong Zheng, and  
 709 Yaodong Yang. Amulet: Realignment during test time for personalized preference adaptation of llms. In *The  
 710 Thirteenth International Conference on Learning Representations*, 2025.
- 711 Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and  
 712 Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *CoRR*, 2022.
- 713 Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei Efros, and Moritz Hardt. Test-time training with  
 714 self-supervision for generalization under distribution shifts. In *International Conference on Machine Learning  
 715 (ICML)*, 2020.
- 716 Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong  
 717 Wang, Sanmi Koyejo, et al. Learning to (learn at test time): Rnns with expressive hidden states. *CoRR*, 2024.
- 718 Moritz Hardt and Yu Sun. Test-time training on nearest neighbors for large language models. In *International  
 719 Conference on Learning Representations (ICLR)*, 2023.
- 720 Shibo Hao, Yi Wang, and Bin Xiao. Reasoning with language model is planning with world model. In *Annual  
 721 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.
- 722 Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Annual  
 723 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2021.
- 724 Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. Gpt understands, too.  
 725 *AI Open*, 5:208–215, 2024a.
- 726 Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Annual Meeting  
 727 of the Association for Computational Linguistics (ACL)*, 2021.
- 728 Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Lam Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. P-tuning v2:  
 729 Prompt tuning can be comparable to fine-tuning universally across scales and tasks. In *Annual Meeting of the  
 730 Association for Computational Linguistics (ACL)*, 2021.
- 731 Salil Vadhan and Qian Zhang. Lecture 31: Multiprover interactive proofs and probabilistically checkable proofs.  
 732 In *Scribe Notes of CS221, Harvard School*, 2002.
- 733 S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press,  
 734 2006. ISBN 978-0-521-42426-4. URL <https://theory.cs.princeton.edu/complexity/book.pdf>.
- 735 Michael Ben-Or, Shafi Goldwasser, Joe Kilian, and Avi Wigderson. Multi-prover interactive proofs: how to  
 736 remove intractability assumptions. In *Annual ACM Symposium on Theory of Computing*, STOC '88, page  
 737 113–131, New York, NY, USA, 1988. Association for Computing Machinery. ISBN 0897912640. doi:  
 738 10.1145/62212.62223. URL <https://doi.org/10.1145/62212.62223>.
- 739 L. Babai, L. Fortnow, and C. Lund. Nondeterministic exponential time has two-prover interactive protocols. In  
 740 *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, pages 16–25 vol.1, 1990.  
 741 doi: 10.1109/FSCS.1990.89520.
- 742 Chris Yuhao Liu, Liang Zeng, Jiacai Liu, Rui Yan, Jujie He, Chaojie Wang, Shuicheng Yan, Yang Liu, and Yahui  
 743 Zhou. Skywork-reward: Bag of tricks for reward modeling in llms. *CoRR*, 2024b.
- 744 Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego  
 745 de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud,  
 746 Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and  
 747 William El Sayed. Mistral 7b, 2023. URL <https://arxiv.org/abs/2310.06825>.
- 748 Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang,  
 749 YK Li, Y Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models.  
 750 *CoRR*, 2024.

- 756 Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyuan Luo, Zhangchi Feng, and Yongqiang Ma.  
757 Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Annual Meeting of the Association*  
758 *for Computational Linguistics (ACL)*, 2024.
- 759 Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John  
760 Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning*  
761 *Representations (ICLR)*, 2023.
- 762
- 763
- 764
- 765
- 766
- 767
- 768
- 769
- 770
- 771
- 772
- 773
- 774
- 775
- 776
- 777
- 778
- 779
- 780
- 781
- 782
- 783
- 784
- 785
- 786
- 787
- 788
- 789
- 790
- 791
- 792
- 793
- 794
- 795
- 796
- 797
- 798
- 799
- 800
- 801
- 802
- 803
- 804
- 805
- 806
- 807
- 808
- 809

# 810 Appendices

## 811 Contents

812	<b>A Discussion and future works</b>	<b>18</b>
813		
814		
815		
816	<b>B Methods of Test-Time Instance-Level Reasoning</b>	<b>18</b>
817		
818		
819		
820	<b>C Theoretical Analysis</b>	<b>18</b>
821		
822	C.1 Preliminaries: Multiple Prover Interactive Proofs and NEXP . . . . .	18
823	C.2 Theoretical Analysis: Independent Updating . . . . .	19
824	C.3 Proofs of Theorem C.10 and Theorem C.11 . . . . .	21
825		
826		
827	<b>D Derivation of Policy Gradient</b>	<b>22</b>
828		
829	<b>E Additional Experimental Results</b>	<b>23</b>
830		
831	E.1 LATENTSEEK vs BoN (N=10) . . . . .	23
832	E.2 Qwen3 AIME Performance . . . . .	23
833	E.3 Token Efficiency Comparison with BoN Baseline . . . . .	24
834	E.4 Greedy Decoding vs Sampling . . . . .	24
835	E.5 Limitations of Outcome Reward Model . . . . .	25
836	E.6 Trends of Reward . . . . .	25
837	E.7 Comparison of Optimizing Initial-Stage Latent Tokens and Optimizing Middle-Stage Latent Tokens . . . . .	25
838		
839	E.8 Performance vs Fraction Ratio . . . . .	26
840		
841		
842		
843	<b>F Experimental Details</b>	<b>26</b>
844		
845	F.1 Prompt Designation . . . . .	27
846	F.2 Backbone . . . . .	27
847	F.3 Baselines . . . . .	27
848	F.4 Hyperparameter Tuning . . . . .	28
849	F.5 GSM8K . . . . .	29
850	F.6 MATH-500 . . . . .	29
851	F.7 AIME2024 . . . . .	31
852		
853		
854		
855	<b>G Detailed FLOPs Calculation</b>	<b>31</b>
856		
857	G.1 Total FLOPs for Genius . . . . .	35
858	G.2 Total FLOPs for LATENTSEEK . . . . .	35
859	G.3 Efficiency Threshold Analysis . . . . .	35
860	G.4 Comparison with iCoT . . . . .	35
861		
862		
863	<b>H Latent Space Trajectories</b>	<b>36</b>

864	<b>I Qualitative Analysis and Case Studies</b>	<b>39</b>
865		
866		
867	<b>J Computational Resources</b>	<b>42</b>
868		
869	<b>K The Use of Large Language Models (LLMs)</b>	<b>42</b>
870		
871		
872		
873		
874		
875		
876		
877		
878		
879		
880		
881		
882		
883		
884		
885		
886		
887		
888		
889		
890		
891		
892		
893		
894		
895		
896		
897		
898		
899		
900		
901		
902		
903		
904		
905		
906		
907		
908		
909		
910		
911		
912		
913		
914		
915		
916		
917		

918 A DISCUSSION AND FUTURE WORKS  
919920 While our work demonstrates the effectiveness of latent space optimization, we offer some discussion  
921 for inspiring future research.  
922923 **Reward Models.** A primary limitation of our current approach is its reliance on a self-rewarding  
924 mechanism. While this proves effective, the optimization process is inherently constrained by the  
925 base model’s own evaluation capabilities and potential biases, lacking a truly objective external  
926 signal. The ideal scenario would involve a principled Outcome Reward Model (ORM) that assesses  
927 the final answer and provides unambiguous guidance for the search. However, as detailed in our  
928 experiments in Section E.5, we found that current publicly available ORMs are not yet sufficiently  
929 robust or generalizable for this task; their noisy signals proved less effective for guiding the latent  
930 space optimization than the more consistent self-reward mechanism. **This highlights a critical area**  
931 **for future work: the development of more powerful verifiers capable of reliably scoring complex**  
932 **reasoning tasks.**933 **Latent Optimization.** We adopt standard policy-gradient methods in our implementation, leaving  
934 the exploration of more advanced reinforcement learning algorithms—such as Proximal Policy  
935 Optimization (PPO)—to future work. In addition, pursuing latent-space-specific optimization may  
936 be a worth exploring direction.937 **Large Base Model.** Our experiments are conducted on models up to 14B-parameter scale, con-  
938 strained by available computational resources. Scaling the approach to larger base models remains an  
939 important avenue for future investigation.  
940941 B METHODS OF TEST-TIME INSTANCE-LEVEL REASONING  
942943 We list the formulations of two classical test-time instance-level reasoning methods:  
944

- 945 • Prompt Engineering: Given the problem instance prompt
- $c$
- , the reward function is simply the
- 
- 946 language modeling distribution:

947 
$$\mathbf{x}^* = \arg \max_{\mathbf{x}} \pi(\mathbf{x} \mid \mathbf{c}) \quad (9)$$
  
948

- 949 • Best-of-N (BoN): Given
- $N$
- i.i.d sequences
- $\mathbf{x}_{(1)}, \mathbf{x}_{(2)}, \dots, \mathbf{x}_{(N)} \sim \pi(\cdot \mid \mathbf{c})$
- ,
- 
- 950

951 
$$\mathbf{x}^* = \arg \max_{\mathbf{x} \in \{\mathbf{x}_{(1)}, \mathbf{x}_{(2)}, \dots, \mathbf{x}_{(N)}\}} R(\mathbf{x}, c) \quad (10)$$
  
952

953 C THEORETICAL ANALYSIS  
954956 This section offers a theoretical framework for LATENTSEEK, with a main focus on justifying  
957 the expressiveness of the independence among latent variables, which may be a good start for  
958 understanding and inspecting our algorithm.  
959960 C.1 PRELIMINARIES: MULTIPLE PROVER INTERACTIVE PROOFS AND NEXP  
961962 This section introduces the concept of Multi-prover Interactive Proofs (MIP) and NEXP for comple-  
963 ness. A comprehensive exposition of MIP is provided in [Vadhan and Zhang \(2002\)](#), while [Arora and](#)  
964 [Barak \(2006\)](#) offers a detailed introduction to theoretical computer science, encompassing concepts  
965 such as language and Turing Machine.966 In the Multi-Prover Interactive Proof (MIP) model, provers may communicate with one another prior  
967 to the initiation of the proof process. Once the proof process begins, however, such communication is  
968 prohibited, and each prover interacts with the verifier in a fully private manner.  
969970 **Definition C.1** (Multiple Prover Interaction). Let  $P_1, P_2, \dots, P_k : \{0, 1\}^* \rightarrow \{0, 1\}^*$  be func-  
971 tions. A  $n$ -round interactions between the verifier  $V$  and the provers  $P_1, P_2, \dots, P_k$ , denoted by  
972  $\langle (P_1, P_2, P_3, \dots, P_k), V \rangle (x)$  is the sequence of the following strings  $q_{11}, q_{21}, \dots, q_{k1}, a_{11}, a_{21},$

972  $\dots, a_{k1}, q_{12}, \dots, q_{k2}, a_{12}, \dots, a_{k2}, \dots, q_{1n}, q_{2n}, \dots, q_{kn}, \dots, a_{1n}, \dots, a_{kn}, v$  defined as follows:  
973  $q_{11}, q_{21}, \dots, q_{k1} = V(x)$   
974  $a_{11} = P_1(x, q_{11})$   
975  $a_{21} = P_2(x, q_{21})$   
976  $\dots$   
977  $a_{k1} = P_k(x, q_{k1})$   
978  $q_{12}, q_{22}, \dots, q_{k2} = V(x, q_{11}, a_{11}, \dots, q_{k1}, a_{k1})$   
979  $\dots$   
980  $a_{kn} = P_k(x, q_{k1}, a_{k1}, \dots, q_{kn})$   
981  $v = V(x, q_{11}, a_{11}, \dots, a_{kn}, q_{kn})$

985 We denote  $\langle (P_1, P_2, P_3, \dots, P_k), V \rangle_V(x)$  to be last output  $v$ .

986 **Definition C.2** ( $k$ -MIP [Vadhan and Zhang \(2002\)](#)). A language  $L$  is in  $k$ -MIP if there is a Turing  
987 machine verifier  $V$  such that on inputs  $x, a_{11}, \dots, a_{ij}$ ,  $V$  runs in time polynomial in  $|x|$  and such  
988 that:

- 989 • Efficiency: The number and length of all messages exchanged is at most polynomial in the  
990 common input  $x$ .
- 992 • Completeness:  $x \in L \Rightarrow \exists P_1, P_2, \dots, P_k, \Pr[\langle (P_1, P_2, \dots, P_k), V \rangle_V(x) = 1] \geq \frac{2}{3}$
- 994 • Soundness:  $x \notin L \Rightarrow \forall P_1, P_2, P_3, \dots, P_k, \Pr[\langle (P_1, P_2, \dots, P_k), V \rangle_V(x) = 1] \leq \frac{1}{3}$

995 **Definition C.3** (MIP [Vadhan and Zhang \(2002\)](#)).  $\text{MIP} = \bigcup_{k \in \mathbb{N}} k\text{-MIP}$

996 Next, we'll introduce NEXP.

998 **Definition C.4** (NTIME [Arora and Barak \(2006\)](#)). For every function  $T : \mathbb{N} \rightarrow \mathbb{N}$  and  $L \subseteq \{0, 1\}^*$ ,  
999 we say that  $L \in \text{NTIME}(T(n))$  if there is a constant  $c > 0$  and a  $cT(n)$ -time non-deterministic  
1000 Turing Machine  $M$  such that for every  $x \in \{0, 1\}^*$ ,  $x \in L \Leftrightarrow M(x) = 1$ .

1001 **Definition C.5** (NP).  $\text{NP} = \bigcup_{k \in \mathbb{N}} \text{NTIME}(n^k)$

1002 **Definition C.6** (NEXP).  $\text{NEXP} = \bigcup_{k \in \mathbb{N}} \text{NTIME}(2^{n^k})$

## 1004 C.2 THEORETICAL ANALYSIS: INDEPENDENT UPDATING

1006 Mentioned in Section 2, the latent representations are independent, which results in the independence  
1007 of the token update process, which at first glance may constrain the model's expressive capacity.  
1008 In this subsection, we focus on decision problems [Arora and Barak \(2006\)](#). Despite discrepancies  
1009 between theoretical frameworks and practical challenges, valuable insights can be gained. Specifically,  
1010 it is possible to demonstrate that the model's expressivity remains theoretically comparable in spite  
1011 of the updating independence.

1012 Our method is strongly related to a TCS complexity class, Multi Prover Interactive Proofs [Ben-Or  
1013 et al. \(1988\)](#) or simply MIP. We initially outline the notations for multiple prover interactions and  
1014 subsequently define the associated complexity class for our latent thought framework, which we  
1015 later demonstrate is equivalent to MIP and the complexity class NEXP (solved in exponential time  
1016 by a non-deterministic Turing Machine). Please refer to Section C.1 for preliminaries. We list the  
1017 relationship as follows:

- 1018 • MIP: A verifier interacts with multiple non-communicating provers to decide membership  
1019 in a language.
- 1020 • LATENTSEEK: the optimized tokens, as they are updated independently, act like the **non-  
1021 communicated** provers who jointly determine the outcome. **After decoding the latent vectors,  
1022 where each prover proposes its own output, the subsequent autoregressive generation is  
1023 regarded as part of the verifier function  $V$ .** Regarding the CoT initialization, recall that the  
1024 MIP framework permits communication among provers *before* the proving process begins.  
1025 Accordingly, initializing the latent vectors with a CoT prompt can be interpreted as this  
1026 pre-proof communication stage.

- 1026 • MIP-Bounded (**LATENTSEEK**): Different from MIP, each prover in LatentSeek can only  
 1027 generate 1 token, whose number of bits is bounded. Furthermore, a polynomial proportion  
 1028 of tokens (such as,  $\rho = 0.2$ ) is allowed to be optimized, leading to a polynomial number of  
 1029 provers.

1030 **Definition C.7** (Multiple Prover Interaction). Let  $P_1, P_2, \dots, P_k : \{0, 1\}^* \rightarrow \{0, 1\}^*$  be functions.  
 1031 A  $n$ -round interactions between the verifier  $V$  and the provers  $P_1, P_2, \dots, P_k$ , denoted by  
 1032  $\langle (P_1, P_2, P_3, \dots, P_k), V \rangle(x)$  is the sequence of the following strings  $q_{11}, q_{21}, \dots, q_{k1}, a_{11}, a_{21},$   
 1033  $\dots, a_{k1}, q_{12}, \dots, q_{k2}, a_{12}, \dots, a_{k2}, \dots, q_{1n}, q_{2n}, \dots, q_{kn}, \dots, a_{1n}, \dots, a_{kn}, v$  defined as follows:

$$\begin{aligned}
 q_{11}, q_{21}, \dots, q_{k1} &= V(x) \\
 a_{11} &= P_1(x, q_{11}) \\
 a_{21} &= P_2(x, q_{21}) \\
 &\dots \\
 a_{k1} &= P_k(x, q_{k1}) \\
 q_{12}, q_{22}, \dots, q_{k2} &= V(x, q_{11}, a_{11}, \dots, q_{k1}, a_{k1}) \\
 &\dots \\
 a_{kn} &= P_k(x, q_{k1}, a_{k1}, \dots, q_{kn}) \\
 v &= V(x, q_{11}, a_{11}, \dots, a_{kn}, q_{kn})
 \end{aligned}$$

1045 We denote  $\langle (P_1, P_2, P_3, \dots, P_k), V \rangle_V(x)$  to be last output  $v$ .  
 1046

1047 Different from MIP, which allows each prover to output a polynomial-length string, in our method,  
 1048 each prover can only output a bounded length of string (a token). We name the corresponding  
 1049 complexity class of our method as MIP-Bounded.

1050 **Definition C.8** (MIP-Bounded). A language  $L$  is in MIP-Bounded if there is a Turing machine verifier  
 1051  $V$ , and a polynomial function  $\text{poly}(\cdot)$  such that on inputs  $x, a_{ij}, q_{ij}$ ,  $V$  runs in time polynomial in  $|x|$   
 1052 and such that:

- 1053 • Bounded:  $\forall i$ , output of  $P_i$  is bounded, its output denoted as  $|a_{ij}|$  satisfies that  $|a_{ij}| \leq C$ ,  
 1054 where  $C$  is a constant greater than 1.
- 1055 • Completeness:  $x \in L \Rightarrow \exists P_1, P_2, \dots, P_{\text{poly}(|x|)}, \Pr[\langle (P_1, P_2, \dots, P_{\text{poly}(|x|)}), V \rangle_V(x) = 1] \geq \frac{2}{3}$
- 1056 • Soundness:  $x \notin L \Rightarrow \forall P_1, P_2, \dots, P_{\text{poly}(|x|)}, \Pr[\langle (P_1, P_2, \dots, P_{\text{poly}(|x|)}), V \rangle_V(x) = 1] \leq \frac{1}{3}$

1061 *Remark C.9.* The constant  $C$  in the definition can be any constant. It won't affect the definition as  
 1062 long as it's a constant.

1063 **Theorem C.10.** *MIP-Bounded = MIP*

1064 Theorem C.10 establishes the efficacy of our framework. By building on the classical result of [Babai et al. \(1990\)](#), we derive Theorem C.11 that clearly illustrates the expressive power of our approach.  
 1065 Refer to Section C.3 for detailed proofs.

1066 **Corollary C.11.** *NP ⊂ NEXP = MIP-Bounded*

1069 The corollary indicates that if the verifier and the provers satisfy some condition, the LatentSeek  
 1070 framework is strong enough, and the independence assumption can be amended. Intuitively speaking,  
 1071 the independence between the provers can be controlled by the verifier, such as the verifier can ask  
 1072 for dependence between the generated tokens.

1073 *There is a limitation in this theoretical analysis, which is why we present it in the appendix: it should  
 1074 be viewed only as an indication or heuristic guideline rather than a rigorous validation.* In particular,  
 1075 we do not account for the change through gradient updates. As a result, there is no guarantee that  
 1076 every token becomes reachable in each step, especially when each prover relies solely on the gradients  
 1077 from the language-model head for guidance.

1078 **All in all, the theorem indicates that with a “strong” base model and a “well-aligned” reward  
 1079 model, the deficit introduced by the independence assumption can be effectively mitigated.**

1080 C.3 PROOFS OF THEOREM C.10 AND THEOREM C.11  
10811082 **Theorem C.12.** *MIP-Bounded = MIP*  
10831084 *Proof.* By the classical results of  $\text{MIP} = 2\text{-MIP} = \text{NEXP}$  Babai et al. (1990), it's sufficient to consider  
1085 only two provers in the MIP class.  
10861087 We will first prove that  $\text{MIP} \subseteq \text{MIP-Bounded}$ .  
10881089 For all  $L \in \text{MIP}$ ,  $\forall x$ , in the  $j$ -th turn, the proofs offered by the two provers are denoted as  $a_{1j}, a_{2j}$ , the  
1090 verifier as  $V$ . According to the definition of MIP,  $\exists$  polynomial function  $\rho(\cdot)$  such that  $|a_{ij}| \leq \rho(|x|)$ , we  
1091 set  $\text{poly}(\cdot) = \frac{2\rho(\cdot)}{C}$ , which is also polynomial. We design the verifier  $V' = V$ .  
1092

- 1093 1. If
- $x \in L$
- ,
- $\exists \text{poly}(|x|)$
- and provers
- $P'_1, P'_2, \dots, P'_{\text{poly}(|x|)}$
- such that the accept rate is larger
- 
- 1094 than
- $\frac{2}{3}$
- in the following manner:
- 
- 1095

1096 The first  $\frac{|a_{1j}|}{C}$  provers output  $a_{1j}$  with each outputting  $C$  bits, and the continuing  $\frac{|a_{2j}|}{C}$   
1097 provers output  $a_{2j}$  with each outputting  $C$  bits. As  $\text{poly}(|x|) = \frac{2\rho(|x|)}{C} \geq \frac{|a_{1j}|}{C} + \frac{|a_{2j}|}{C}$ , we  
1098 have enough provers to do this operation. Therefore, the outputs of  $P'_1, P'_2, \dots, P'_{\text{poly}'(|x|)}$  is  
1099 the same as the original two provers, as the original two provers satisfies that the accepts  
1100 probability is greater than  $\frac{2}{3}$ , we have  
1101

1102 
$$\Pr[\langle (P'_1, P'_2, \dots, P'_{\text{poly}(|x|)}), V' \rangle_{V'}(x) = 1] \geq \frac{2}{3}$$
  
1103

- 1104 2. If
- $x \notin L$
- ,
- $\forall P'_1, P'_2, \dots, P'_{\text{poly}'(|x|)}$
- , we denote their output at
- $j$
- th turn as
- 
- 1105
- $a'_{1j}, a'_{2j}, a'_{3j}, \dots, a'_{\text{poly}'(|x|),j}$
- . Let the first prover output the concat output of odd provers,
- 
- 1106 i.e.
- $a_{1j} = (a'_{1j}, a'_{2j}, \dots, a'_{\lfloor \frac{\text{poly}'(|x|)}{2} \rfloor, j})$
- , the second output the concat output of even provers,
- 
- 1107 i.e.
- $a_{2j} = (a'_{\lceil \frac{\text{poly}'(|x|)}{2} \rceil, j}, \dots, a'_{\text{poly}'(|x|), j})$
- . As no two provers can make the verifier accept at a
- 
- 1108 probability larger than
- $\frac{1}{3}$
- , thus
- 
- 1109

1110 
$$\Pr[\langle (P'_1, P'_2, \dots, P'_{\text{poly}(|x|)}), V' \rangle_{V'}(x) = 1] \leq \frac{1}{3}$$
  
1111

1112 Therefore,  $L \in \text{MIP-Bounded}$ , and thus  $\text{MIP} \subseteq \text{MIP-Bounded}$ .  
11131114 Next, we are going to prove  $\text{MIP-Bounded} \subseteq \text{MIP}$ .  
11151116  $\forall L \in \text{MIP-Bounded}$ ,  $\forall x$ , in the  $j$ -th turn, the proofs offer by the bounded provers are denoted  
1117 as:  $a'_{1j}, a'_{2j}, a'_{3j}, \dots, a'_{\text{poly}'(|x|),j}$ , the verifier as  $V'$ . We design  $V$  as follows: for each concat string  
1118  $s = (x, q_{11}, a_{11}, \dots)$ ,  $V$  first truncates the first prover's answer to length of  $\text{LIM}_1 := C \cdot \lfloor \frac{\text{poly}'(|x|)}{2} \rfloor$   
1119 bits and the second prover's answer to length of  $\text{LIM}_2 := C \cdot (\text{poly}'(|x|) - \lceil \frac{\text{poly}'(|x|)}{2} \rceil + 1)$  bits. The  
1120 resulted string is denoted as  $s' := (x, q_{11}, a_{11:0:\text{LIM}_1}, q_{21}, a_{21:0:\text{LIM}_2} \dots)$ , and output  $V'(s')$ .  
1121

- 1122 1. If
- $x \in L$
- , similar as above, we let the first prover output the concat output of odd provers,
- 
- 1123 i.e.
- $a_{1j} = (a'_{1j}, a'_{2j}, \dots, a'_{\lfloor \frac{\text{poly}'(|x|)}{2} \rfloor, j})$
- , the second output the concat output of even provers,
- 
- 1124 i.e.
- $a_{2j} = (a'_{\lceil \frac{\text{poly}'(|x|)}{2} \rceil, j}, \dots, a'_{\text{poly}'(|x|), j})$
- . As
- $|a_{1j}| \leq \text{LIM}_1$
- , it won't be truncated and
- 
- 1125
- $|a_{2j}| \leq \text{LIM}_2$
- , it won't be truncated, thus
- $V$
- acts exactly as
- $V'$
- , and therefore
- 
- 1126

1127 
$$\Pr[\langle (P_1, P_2), V \rangle_V(x) = 1] \geq \frac{2}{3}$$
  
1128

- 1129 2. (Prove by contradiction) If
- $x \notin L$
- , suppose
- $\exists P_1, P_2$
- such that:
- 
- 1130

1131 
$$\Pr[\langle (P_1, P_2), V \rangle_V(x) = 1] > \frac{1}{3}$$
  
1132

1134 As  $V$  first truncate answer, therefore, for  $j$ -turns, we can truncate  $a_{1j}$  to  $a_{1j;0:\text{LIM}_1}$  and  
 1135  $a_{2j}$  to  $a_{2j;0:\text{LIM}_2}$ , and follow the division process depicted in the first case of proving  
 1136 MIP  $\subseteq$  MIP-Bounded to gain  $P'_1, P'_2, P'_3, \dots, P'_{\text{poly}(|x|)}$  provers. Thus:

$$\begin{aligned} 1138 \quad \Pr[\langle (P_1, P_2), V \rangle_V(x) = 1] &= \Pr[\langle (P'_1, P'_2, \dots, P'_{\text{poly}(|x|)}), V \rangle_V(x) = 1] \\ 1139 &= \Pr[\langle (P'_1, P'_2, \dots, P'_{\text{poly}(|x|)})', V' \rangle_{V'}(x) = 1] \end{aligned}$$

1140 As a result,

1141  $\Pr[\langle (P'_1, P'_2, \dots, P'_{\text{poly}(|x|)}), V' \rangle_{V'}(x) = 1] > \frac{1}{3}$   
 1142 , this contradicts to  $\Pr[\langle (P'_1, P'_2, \dots, P'_{\text{poly}(|x|)}), V' \rangle_{V'}(x) = 1] \leq \frac{1}{3}$ . Therefore,  $\forall P_1, P_2$ ,  
 1143 we have:

$$1144 \quad \Pr[\langle (P_1, P_2), V \rangle_V(x) = 1] \leq \frac{1}{3}$$

1145 Hence, we have  $L \in \text{MIP}$ , which means that MIP-Bounded  $\subseteq$  MIP.

1146 Concluding the above, we have MIP-Bounded = MIP.  $\square$

1147 **Theorem C.13** (Babai et al. (1990)).  $\text{MIP} = 2\text{-MIP} = \text{NEXP}$

1148 *Remark C.14.* NEXP is the complexity class that a non-deterministic Turing Machine can solve in  
 1149 exponential time, which means that  $\text{NP} \subset \text{NEXP}$ .

1150 Based on this theorem, we can easily derive Theorem C.11.

1151 **Corollary C.15.**  $\text{NP} \subset \text{NEXP} = \text{MIP-Bounded}$

1152

## D DERIVATION OF POLICY GRADIENT

1153 Our target is to derive Equation (7):

$$1154 \quad [\nabla_{\mathbf{z}} \mathcal{J}(\mathbf{z})]_t = \mathbb{E}_{\mathbf{x} \sim \pi(\mathbf{x} | \mathbf{z}, \mathbf{c})} [R(\mathbf{x}, \mathbf{c}) \nabla_{z_t} \log \pi(x_t | z_t)]$$

1155 Starting from the objective:

$$1156 \quad \mathcal{J}(\mathbf{z}) := \mathbb{E}_{\mathbf{x} \sim \pi(\mathbf{x} | \mathbf{z}, \mathbf{c})} [R(\mathbf{x}, \mathbf{c})]$$

1157 By taking gradient of  $\mathbf{z}$ , we have:

$$1158 \quad \nabla_{\mathbf{z}} \mathcal{J}(\mathbf{z}) = \nabla_{\mathbf{z}} \int_{\mathbf{x}} R(\mathbf{x}, \mathbf{c}) \pi(\mathbf{x} | \mathbf{z}, \mathbf{c}) d\mathbf{x} = \int_{\mathbf{x}} R(\mathbf{x}, \mathbf{c}) \nabla_{\mathbf{z}} \pi(\mathbf{x} | \mathbf{z}, \mathbf{c}) d\mathbf{x}$$

1159 Leveraging  $\nabla_{\mathbf{z}} \log \pi(\mathbf{x} | \mathbf{z}, \mathbf{c}) = \frac{1}{\pi(\mathbf{x} | \mathbf{z}, \mathbf{c})} \nabla_{\mathbf{z}} \pi(\mathbf{x} | \mathbf{z}, \mathbf{c})$ :

$$1160 \quad \nabla_{\mathbf{z}} \mathcal{J}(\mathbf{z}) = \int_{\mathbf{x}} R(\mathbf{x}, \mathbf{c}) \pi(\mathbf{x} | \mathbf{z}, \mathbf{c}) \nabla_{\mathbf{z}} \log \pi(\mathbf{x} | \mathbf{z}, \mathbf{c}) d\mathbf{x}$$

1161 According to  $\pi(\mathbf{x} | \mathbf{z}, \mathbf{c}) = \prod_{t=1}^N \pi(x_t | z_t) \prod_{t=N+1}^T \pi(x_t | x_{<t}, \mathbf{c})$ , we have:

$$1162 \quad \nabla_{\mathbf{z}} \mathcal{J}(\mathbf{z}) = \int_{\mathbf{x}} R(\mathbf{x}, \mathbf{c}) \pi(\mathbf{x} | \mathbf{z}, \mathbf{c}) \nabla_{\mathbf{z}} \left( \sum_{t=1}^N \log \pi(x_t | z_t) + \sum_{t=N+1}^T \log \pi(x_t | x_{<t}) \right) d\mathbf{x}$$

1163 As the second term doesn't have  $\mathbf{z}$ , thus we have:

$$1164 \quad \nabla_{\mathbf{z}} \mathcal{J}(\mathbf{z}) = \int_{\mathbf{x}} R(\mathbf{x}, \mathbf{c}) \pi(\mathbf{x} | \mathbf{z}, \mathbf{c}) \nabla_{\mathbf{z}} \left( \sum_{t=1}^N \log \pi(x_t | z_t) \right) d\mathbf{x} = \mathbb{E}_{\mathbf{x} \sim \pi(\mathbf{x} | \mathbf{z}, \mathbf{c})} [R(\mathbf{x}, \mathbf{c}) \nabla_{\mathbf{z}} \left( \sum_{t=1}^N \log \pi(x_t | z_t) \right)]$$

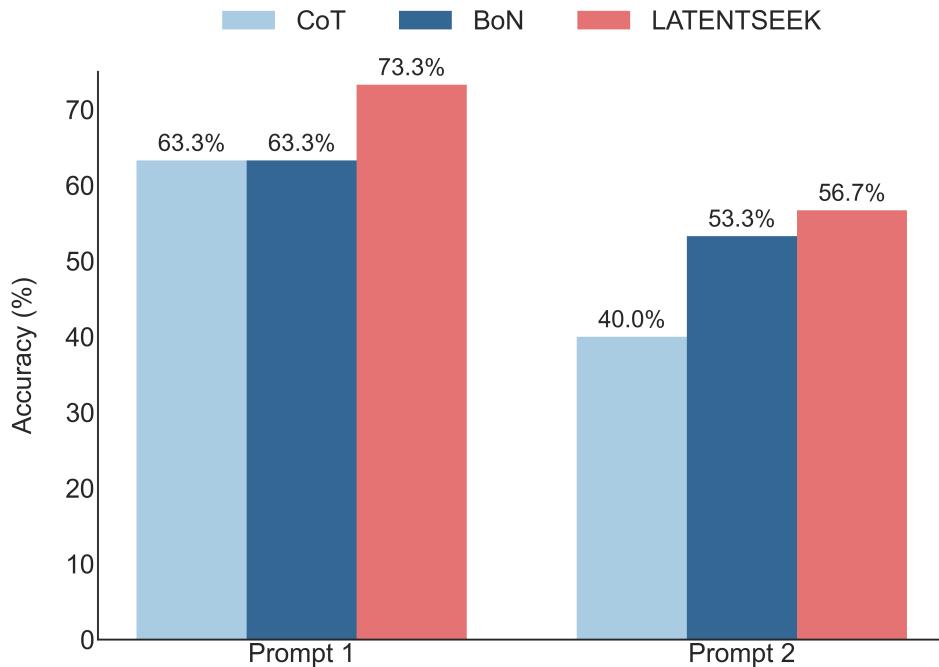
1165 Therefore, the  $t$ -th term is:

$$1166 \quad [\nabla_{\mathbf{z}} \mathcal{J}(\mathbf{z})]_t = \mathbb{E}_{\mathbf{x} \sim \pi(\mathbf{x} | \mathbf{z}, \mathbf{c})} [R(\mathbf{x}, \mathbf{c}) \nabla_{z_t} \log \pi(x_t | z_t)],$$

1167 which is exactly what we want.

1188 E ADDITIONAL EXPERIMENTAL RESULTS  
11891190 E.1 LATENTSEEK VS BoN (N=10)  
11911192 In Table 1, we show that Best-of-N (BoN) is outperformed by LATENTSEEK, even though BoN **con-**  
1193 **sumes a comparable compute consumption (See Figure 4)**. One might also ask how LATENTSEEK  
1194 compares to BoN when  $N = 10$ ; in this setting, BoN requires at least five times more sequence-level  
1195 computation than LATENTSEEK. As reported in Table 10, LATENTSEEK remains superior on all  
1196 datasets and all backbone models.1197 Table 10: Additional Comparison Between LATENTSEEK and BoN (N=10) with Prompt 2.  
1198

1199 Methods \ Model	1200 Qwen2	1201 7B	1.5B	1202 Qwen2.5	1203 7B	14B	1204 LLaMA3.1	1205 8B	1206 Avg
<i>GSM8K</i>									
BoN	72.10	7.73		80.06		92.19	81.58		66.73
LATENTSEEK	<b>80.21</b> <sub>+8.11</sub>	<b>44.20</b> <sub>+36.47</sub>		<b>85.06</b> <sub>+5.00</sub>		<b>92.72</b> <sub>+0.53</sub>	<b>83.70</b> <sub>+2.12</sub>		<b>77.18</b> <sub>+10.45</sub>
<i>MATH-500</i>									
BoN	42.40	28.80		56.40		63.20	45.60		47.28
LATENTSEEK	<b>44.80</b> <sub>+2.40</sub>	<b>32.20</b> <sub>+3.40</sub>		<b>57.60</b> <sub>+1.20</sub>		<b>71.00</b> <sub>+7.80</sub>	<b>47.00</b> <sub>+1.40</sub>		<b>50.52</b> <sub>+3.24</sub>
<i>AIME2024</i>									
BoN	3.33	0.00		6.67		10.00	3.33		4.66
LATENTSEEK	<b>3.33</b> <sub>+0.00</sub>	<b>3.33</b> <sub>+3.33</sub>		<b>13.33</b> <sub>+6.66</sub>		<b>10.00</b> <sub>+0.00</sub>	<b>6.67</b> <sub>+3.33</sub>		<b>7.33</b> <sub>+2.67</sub>

1211 E.2 QWEN3 AIME PERFORMANCE  
12121236 Figure 3: Performance comparison of the Qwen3-4B-Instruct model on the AIME2024 dataset. The  
1237 chart illustrates the accuracy of LATENTSEEK against the Chain-of-Thought (CoT) and Best-of-N  
1238 (BoN) baselines across two distinct prompt formats.  
12391240 This section provides a detailed study on the performance of the Qwen3-4B-Instruct-2507 model  
1241 on the challenging AIME2024 dataset. Figure 3 illustrates the accuracy of LATENTSEEK compared  
against the Chain-of-Thought (CoT) and Best-of-N (BoN) baselines under two prompt formats.

As shown in the figure, under Prompt 1, while the BoN strategy fails to improve upon the CoT baseline (both at 63.3% accuracy score), LATENTSEEK delivers a significant **10 points uplift** to 73.3%. This advantage holds under Prompt 2, where our method’s accuracy of 56.7% surpasses CoT by **16.7 points** and BoN by **3.4 points**. Taken together, these findings demonstrate that LATENTSEEK **consistently improves upon strong baselines** and is valuable in scenarios where conventional search methods are insufficient.

### E.3 TOKEN EFFICIENCY COMPARISON WITH BON BASELINE

This section provides a quantitative analysis of the token efficiency of LATENTSEEK, comparing it against the Best-of-N (BoN) baseline reported in Table 1. The analysis measures the average number of tokens consumed per problem instance under the JSON prompt format (Prompt 2) on the GSM8K and MATH-500 datasets.

As shown in Figure 4, LATENTSEEK achieves higher token efficiency than BoN in all of cases across different model families and sizes, while attaining superior reasoning accuracy as reported in Table 1. These results demonstrates that LATENTSEEK can enhance performance without incurring additional token overhead for all models, highlighting the efficiency of latent space optimization.

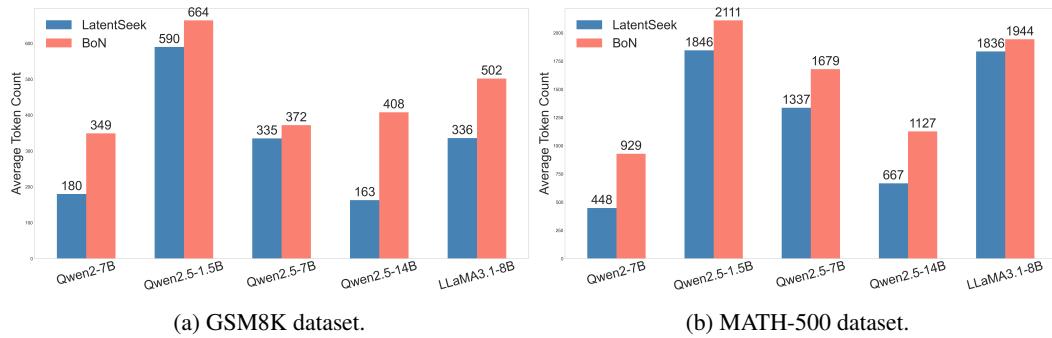


Figure 4: Comparison of average tokens consumed per problem between LATENTSEEK and the Best-of-N (BoN) baseline when using Prompt 2. The token consumption is comparable for the two methods, with BoN consuming slightly more tokens in total. For experimental results where BoN consumes extremely more calculation, please refer to Table 10.

### E.4 GREEDY DECODING VS SAMPLING

Our experiments show that greedy decoding closely approximates categorical sampling while being a better choice for reducing stochastic error. As detailed in Table 11, greedy decoding performs slightly better than the sampling method. We hypothesize that this is because sampling can amplify the approximation error when replacing an expected mean with an empirical one. Therefore, we use greedy decoding for our main experiments.

Table 11: LATENTSEEK: Greedy Decoding vs Sampling (Temperature 0.7) on the MATH-500 dataset.

Methods	Model	Qwen2		Qwen2.5		LLaMA3.1	
		7B	1.5B	7B	14B	8B	
<i>Prompt 1</i>							
Sampling		56.80	50.00	74.00	74.00	49.60	
Greedy Decoding		57.40	55.60	75.60	80.00	54.60	
<i>Prompt 2</i>							
Sampling		44.40	22.40	55.80	68.40	45.40	
Greedy Decoding		44.80	34.40	57.60	71.00	47.00	

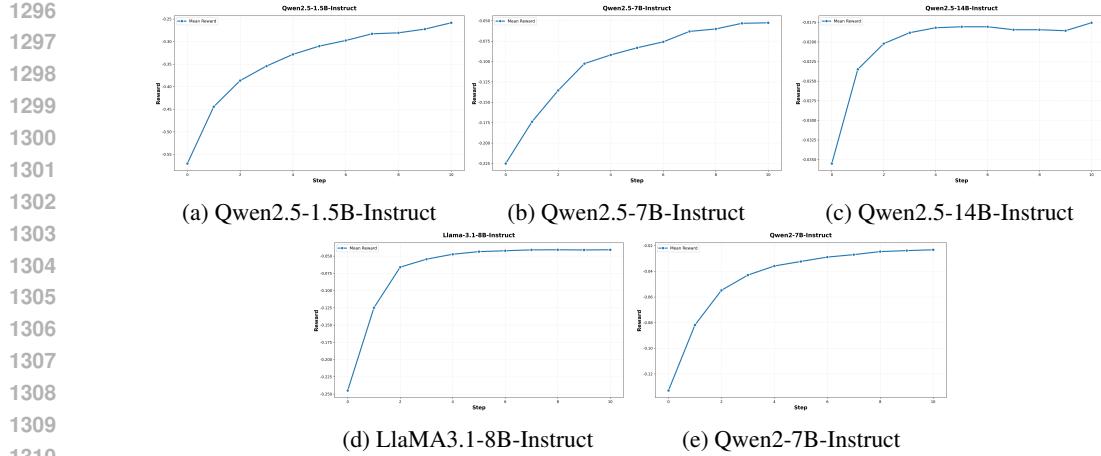


Figure 5: Trends of reward on GSM8K dataset. Generation uses the prompt 2 (the Json prompt).

### E.5 LIMITATIONS OF OUTCOME REWARD MODEL

The performance is described in Liu et al. (2024b), with results presented in Table 12. Although this model achieves an average score of 64.79—representing an improvement of 3.23 points over the Chain-of-Thought (CoT) method—it remains significantly inferior to the performance attained using a self-rewarding mechanism, which reaches 77.18. These results highlight the limitations of current outcome-based reward models in scenarios lacking ground-truth data, underscoring the need for the development of more robust and effective outcome reward models.

Table 12: Math Reward Model Accuracy results on GSM8k Cobbe et al. (2021), MATH-500 Hendrycks et al. (2021), and AIME2024 datasets. The best performances are highlighted in bold, and the second-best performances are underlined.

Methods \ Model	Qwen2 7B	Qwen2.5 1.5B	Qwen2.5 7B	Qwen2.5 14B	LLaMA3.1 8B	Avg
<i>GSM8k: Prompt 2</i>						
CoT	65.20	15.31	66.41	91.81	69.07	61.56
LATENTSEEK (Math)	61.18	<u>28.28</u>	66.64	91.05	<u>76.80</u>	<u>64.79</u>
LATENTSEEK (Self)	<b>80.21</b>	<b>44.20</b>	<b>85.06</b>	<b>92.72</b>	<b>83.70</b>	<b>77.18</b>

### E.6 TRENDS OF REWARD

We plot the reward trends in Figure 5, which empirically illustrates how the update dynamics guide the model from a low-reward region toward a higher-reward region. The reward curves are obtained by running LATENTSEEK on the GSM8K dataset, with generation constrained by a JSON-formatted prompt.

### E.7 COMPARISON OF OPTIMIZING INITIAL-STAGE LATENT TOKENS AND OPTIMIZING MIDDLE-STAGE LATENT TOKENS

One may wonder how performance changes when we instead optimize **middle-stage** tokens. Formally, this corresponds to the factorization

$$\pi(\mathbf{x} \mid \mathbf{z}, \mathbf{c}) = \prod_{t=1}^{N_1} \pi(x_t \mid \mathbf{x}_{<t}, \mathbf{c}) \prod_{t=N_1+1}^N \pi_{\text{LM-head}}(x_t \mid z_t) \prod_{t=N+1}^T \pi(x_t \mid \mathbf{x}_{<t}, \mathbf{c}).$$

Intuitively, because the prefix tokens  $x_1$  to  $x_{N_1}$  do not depend on the optimized middle-stage tokens  $\mathbf{x}_{N_1+1:N}$ , the model may be unable to fully exploit the improved latent representations, potentially

1350      Table 13: Comparison with Optimizing the Middle-Stage Tokens on GSM8K. The best performances  
 1351      are highlighted in bold.  
 1352

Model	Qwen2 7B	Qwen2.5 1.5B	Qwen2.5 7B	14B	LLaMA3.1 8B	Avg
<i>GSM8K: Prompt 1</i>						
CoT	68.01	68.08	88.86	92.03	50.19	73.43
LATENTSEEK (middle)	74.68	69.07	89.31	92.49	65.21	78.15
<b>LATENTSEEK</b>	<b>84.38</b>	<b>70.89</b>	<b>90.14</b>	<b>92.49</b>	<b>78.54</b>	<b>83.29</b>
<i>GSM8K: Prompt 2</i>						
CoT	65.20	15.31	66.41	91.81	69.07	61.56
LATENTSEEK (middle)	70.73	22.44	72.93	92.49	77.26	67.17
<b>LATENTSEEK</b>	<b>80.21</b>	<b>44.20</b>	<b>85.06</b>	<b>92.72</b>	<b>83.70</b>	<b>77.18</b>

1363  
 1364      causing performance degradation. To test this hypothesis, we conduct an additional experiment where  
 1365      we optimize tokens starting from (40%) into the sequence. The results, shown in Table 13, confirm  
 1366      our intuition: although optimizing middle-stage tokens yields a 5.16-point improvement over CoT  
 1367      on average, it still underperforms the version that optimizes the **initial-stage** tokens by 7.58 points.  
 1368      This gap supports our hypothesis that prefix independence limits the effectiveness of optimizing  
 1369      middle-stage tokens.  
 1370

## 1371      E.8 PERFORMANCE VS FRACTION RATIO

1372  
 1373      Figure 6 illustrates the performance trends of various instruction-tuned language models across  
 1374      different fraction ratios, which range from 0.1 to 0.8. Here, we also experimented with another model:  
 1375      *Mistral-7B-Instruct* (Jiang et al., 2023). Performance is reported as a percentage and plotted on the  
 1376      y-axis, while the fraction ratio is shown on the x-axis.

- 1377      • *Qwen2.5-14B-Instruct* exhibits a relatively stable performance curve. It shows a slight  
 1378      increase in performance from 0.1 to 0.6, peaking around the 0.6 fraction ratio, followed by a  
 1379      minor decrease at 0.8. Overall, this model maintains consistently high performance above  
 1380      91% across all fraction ratios.
- 1381      • *Qwen2.5-7B-Instruct* demonstrates a mild upward trend from 0.1 to 0.2, followed by a  
 1382      steady decrease as the fraction ratio increases beyond 0.2, indicating reduced effectiveness  
 1383      at higher ratios.
- 1384      • *Qwen2-7B-Instruct* shows a consistent downward trend across the full range of fraction  
 1385      ratios, suggesting that its performance deteriorates steadily with increasing input fraction.
- 1386      • *LLaMA3.1-8B-Instruct* remains relatively stable at first but shows a slight decreasing trend  
 1387      overall, with performance gently declining from 0.2 onwards.
- 1388      • *Qwen2.5-1.5B-Instruct* follows a sharp non-monotonic trend. It increases markedly from 0.1  
 1389      to peak at 0.2, then drops rapidly as the fraction ratio increases further, reaching its lowest  
 1390      performance at 0.8.
- 1391      • *Mistral-7B-Instruct* exhibits a subtle peak at 0.2, followed by a gradual and modest decrease  
 1392      in performance as the fraction ratio increases, maintaining relatively stable performance  
 1393      throughout.

1394  
 1395      In summary, the *Qwen2.5-14B-Instruct* model shows a slightly rising then declining trend but remains  
 1396      highly stable and strong overall. Smaller models, especially *Qwen2.5-1.5B-Instruct*, are more sensitive  
 1397      to increases in fraction ratio, with noticeable performance degradation at higher values. Mid-sized  
 1398      models like *Mistral-7B-Instruct* and *LLaMA3.1-8B-Instruct* exhibit relatively mild downward trends,  
 1399      indicating moderate robustness.

## 1400      F EXPERIMENTAL DETAILS

1401  
 1402      In this section, we provide a detailed description of the experimental setup.  
 1403

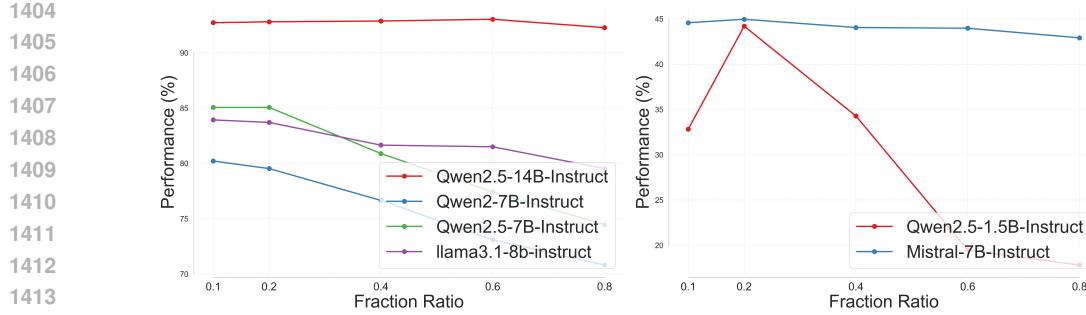


Figure 6: Performance vs Fraction Ratio

### F.1 PROMPT DESIGNATION

The corresponding prompts are shown in the following tables.

**Prompt.** The prompt of CoT is as follows.

- Prompt 1: Table 15 and Table 16.
- Prompt 2: Table 14.

**Reward Prompt.** The prompts are listed in Tables 23 to 26.

### F.2 BACKBONE

We list all model backbone as follows:

- Qwen2-7B-Instruct [Yang et al. \(2024\)](#): A powerful model designed for instruction-based tasks, leveraging the 7B parameter version of Qwen2. (accessible at <https://huggingface.co/Qwen/Qwen2-7B-Instruct>)
- Qwen2.5-1.5B-Instruct [Team \(2024a\)](#): A compact yet efficient model designed for task-specific instructions, based on the 1.5B-parameter Qwen2.5. (accessible at <https://huggingface.co/Qwen/Qwen2.5-1.5B-Instruct>)
- Qwen2.5-7B-Instruct [Team \(2024a\)](#): A middle-tier model based on 7B-parameter Qwen2.5, optimized for handling various instructions. (accessible at <https://huggingface.co/Qwen/Qwen2.5-7B-Instruct>)
- Qwen2.5-14B-Instruct [Team \(2024a\)](#): A robust, large-scale model built on the 14B-parameter Qwen2.5, excelling in complex instruction-based tasks. (accessible at <https://huggingface.co/Qwen/Qwen2.5-14B-Instruct>)
- LLaMA3.1-8B-Instruct [Team \(2024b\)](#): LLaMA's 8B parameter version designed for better instruction-following capabilities. (accessible at <https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct>)

### F.3 BASELINES

We describe all baselines in experiments as follows:

- Chain-of-Thought (CoT): CoT refers to a structured, sequential approach to problem-solving, wherein complex tasks are decomposed into intermediate steps, each explicitly articulated to facilitate logical progression toward a solution. In our experiments, CoT follows the prompt listed in Section F.1.
- Few-Shot CoT: Few-Shot Chain-of-Thought (CoT) prompting is a technique in natural language processing that enhances model performance on complex reasoning tasks by providing a small number of illustrative examples with step-by-step reasoning, enabling the model to generalize and produce structured, logical outputs. In our experiment, we adopt the result from [Lambert et al. \(2025\)](#).

1458 Table 14: Prompt 1 for GSM8k [Cobbe et al. \(2021\)](#), MATH-500 [Hendrycks et al. \(2021\)](#) and  
 1459 AIME2024.

1460

```
1461 {"role": "system", "content": "Please reason step by step, and  

  1462 put your final answer within \boxed{\{ \} . } ",  

  1463 \{"role": "user", "content": "\{q\}\\"}
```

1464

1465

- 1466 • Best-of- $N$  (BoN): We draw  $N$  independent candidates from the model, score each with a  
 1467 predefined objective, and select the candidate with the highest reward. In our experiments,  
 1468 we set  $N = 3$  so that the total computation is comparable to—while exceeding—that of  
 1469 LATENTSEEK. The BoN reward is computed under the same self-reward mechanism as that  
 1470 of LATENTSEEK.
- 1471 • Self-Consistency: We draw  $N$  independent candidates from the model, and select the  
 1472 candidate that appears with the highest frequency. In our experiments, we set  $N = 3$  so that  
 1473 the total computation is comparable to—while exceeding—that of LATENTSEEK and align  
 1474 with that of BoN.
- 1475 • Self-Rewarding [Yuan et al. \(2025\)](#): This leverages intrinsic feedback mechanisms to iter-  
 1476 atively enhance model performance without reliance on external reward signals.
- 1477 • ScPO [Prasad et al. \(2024\)](#): A framework for self-consistency preference optimization,  
 1478 rigorously formalizing the alignment of decision-making processes with logically coherent  
 1479 and preference-driven outcomes in complex systems.
- 1480 • CoH [Liu et al. \(2023a\)](#): A framework which systematically aligns language models with  
 1481 human feedback through a structured, iterative process, enhancing their performance in  
 1482 complex reasoning tasks.
- 1483 • Genius [Xu et al. \(2025\)](#): A purely unsupervised self-training framework designed to enhance  
 1484 advanced reasoning capabilities in artificial intelligence systems, offering generalizable  
 1485 performance across diverse tasks without reliance on labeled data.
- 1486 • SimpleRL-Zoo [Zeng et al. \(2025\)](#): A reinforcement learning framework designed to enhance  
 1487 sample efficiency and performance stability in complex decision-making environments  
 1488 through simplified algorithmic structures and adaptive exploration strategies.
- 1489 • GRPO [Shao et al. \(2024\)](#): Group Relative Policy Optimization (GRPO), introduced  
 1490 in the DeepSeekMath framework, is a novel reinforcement learning algorithm that  
 1491 enhances mathematical reasoning in large language models by optimizing policy up-  
 1492 dates through group-based reward comparisons, significantly reducing memory con-  
 1493 sumption compared to traditional Proximal Policy Optimization (PPO). In our ex-  
 1494 periment, we adopt the result from [https://www.perplexity.ai/hub/blog/](https://www.perplexity.ai/hub/blog/rl-training-for-math-reasoning?utm_source=chatgpt.com)  
 1495 [rl-training-for-math-reasoning?utm\\_source=chatgpt.com](#).
- 1496 • SPIN [Chen et al. \(2024a\)](#): A self-play fine-tuning methodology that significantly enhances  
 1497 the performance of weaker language models, transforming them into robust and highly  
 1498 capable systems competitive with stronger counterparts.
- 1499 • iCoT [Deng et al. \(2024\)](#): a transition from explicit Chain-of-Thought (CoT) reasoning to  
 1500 implicit CoT internalization, proposing a step-by-step learning framework to enhance logical  
 1501 reasoning capabilities in artificial intelligence systems.
- 1502 • Supervised Fine-Tuning (SFT) was conducted using the LLaMA-Factory framework [Zheng](#)  
 1503 [et al. \(2024\)](#). All models were trained with a learning rate of  $1 \times 10^{-5}$ , employing a cosine  
 1504 learning rate scheduler, a warmup ratio of 0.1, and the bfloat16 (bf16) data type.

1507

1508

#### F.4 HYPERPARAMETER TUNING

1509

1510 To determine the best choices for  $\rho$  and the learning-rate hyperparameters, we perform a grid search  
 1511 on a held-out subset of the training data. For GSM8K, for example, we randomly sample 500 training  
 instances and use this subset exclusively for hyperparameter tuning.

1512 Table 15: Prompt 2 for GSM8k [Cobbe et al. \(2021\)](#).  
1513

---

```

1514 \{"role": "system", "content": "You are a precise math question
1515 solver. Solve this math problem." \},
1516 \{"role": "user", "content": "QUESTION: {q} Let's think step by
1517 step. Please provide your thought process and your final answer
1518 separately and response in json format containing the keys
1519 thought process and final answer. For example your response
1520 should be \{"thought process": 'your thought process', 'final
1521 answer': 'your final answer'\}. Note that the final answer
1522 should be pure numbers, not the calculation formulas, and
1523 without any units or explanation!!!"\}

```

---

1524 Table 16: Prompt 2 for MATH-500 [Hendrycks et al. \(2021\)](#) and AIME2024.  
1525

---

```

1526 \{"role": "system", "content": "You are a precise math question
1527 solver. Solve this math problem." \},
1528 \{"role": "user", "content": "QUESTION: {q} Let's think step by
1529 step. Please provide your thought process and your final answer
1530 separately and response in json format containing the keys
1531 thought process and final answer. For example your response
1532 should be \{"thought process": 'your thought process', 'final
1533 answer': 'your final answer'\}." \}

```

---

## 1534 F.5 GSM8K

1535 We provide details for GSM8K experiments as follows:

1536 **Dataset.** The GSM8K dataset [Cobbe et al. \(2021\)](#), a comprehensive collection of mathematical reasoning problems, serves as a benchmark for evaluating the problem-solving capabilities of language models. Developed by OpenAI and accessible via the Hugging Face repository at <https://huggingface.co/datasets/openai/gsm8k>, GSM8K comprises 8,500 meticulously curated, high-quality math problems that span a diverse range of topics, including arithmetic, algebra, and word problems. These problems are specifically designed to assess a model's ability to perform multi-step reasoning, interpret natural language descriptions of mathematical scenarios, and derive accurate solutions. The dataset is partitioned into a training set of 7,473 examples and a test set of 1,319 examples, enabling robust model training and evaluation. For the purposes of this study, we exclusively utilize the test set to evaluate model performance, ensuring a standardized and unbiased assessment of mathematical reasoning proficiency.

1537 **Experimental Details.** For all backbones and both prompts, we use greedy decoding for inference. For the hyperparameters of LATENTSEEK (Self) and LATENTSEEK (PSRM), please refer to Table 17 and Table 18, respectively. The mathematical reasoning prompts we employed in the self-reward mechanism evaluate answers across four dimensions: correctness of the final answer, accuracy of problem comprehension, correctness of numerical calculations, and provision of a clear answer, weighted at 1:1:2:2, with the final score normalized to the range [-1, 0]. The detailed specifications of these four evaluation prompts are provided in Tables 23 to 26. The prompt structure is consistently applied across all backbones and both prompts. Notably, for Llama-3.1-8B-Instruct, Qwen2.5-1.5B-Instruct and Qwen2-7B-Instruct in LATENTSEEK (Self) experiment using Prompt 1, a new format reward following the methodology of DeepSeek-R1 is incorporated, with weights of 3, 2, and 2 assigned to format-based criteria, respectively.

1538 **Scaling.** We offer more scaling figures in Figure 7.

## 1539 F.6 MATH-500

1540 We provide details for MATH-500 experiments as follows:

1541 **Dataset.** The MATH-500 dataset, a curated subset of the MATH benchmark, serves as a robust resource for evaluating the mathematical reasoning capabilities of machine learning mod-

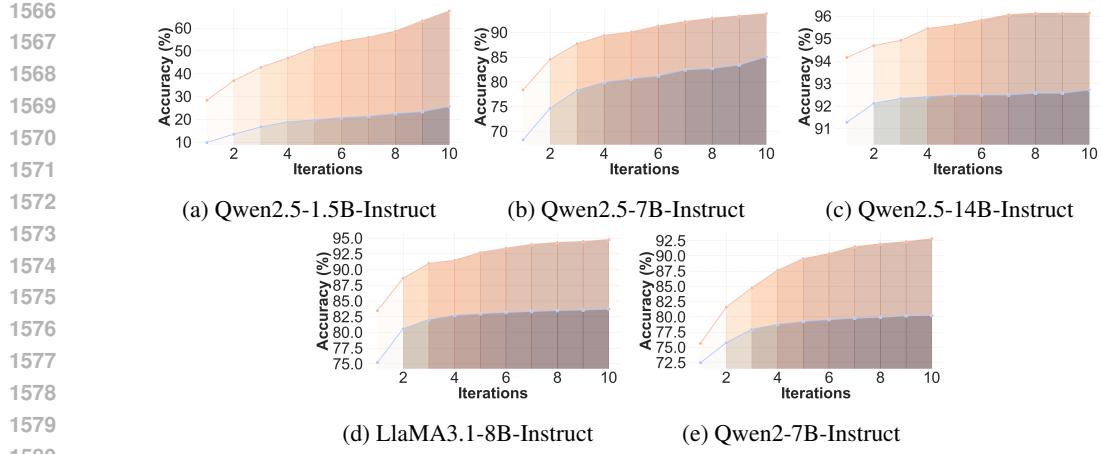


Figure 7: GSM8k [Cobbe et al. \(2021\)](#) Prompt 2 Accuracy changes with respect to the increasing number of iterations. **Orange**: Perfect Sharp Reward Model. **Blue**: Self Reward Model.

Table 17: LATENTSEEK (Self) Hyperparameters on GSM8K. lr: learning rate.  $\rho$ : fraction ratio

methods	model	max len	prompt idx	#GPU	lr	optimizer	$\rho$	dtype	Max Step
LATENTSEEK (Self)	Qwen2-7B-Instruct	1024	1	1 A100	0.03	Adam	0.2	bf16	10
LATENTSEEK (Self)	Qwen2-7B-Instruct	1024	2	1 L40	0.03	Adam	0.2	bf16	10
LATENTSEEK (Self)	Qwen2.5-1.5B-Instruct	1024	1	1 3090	0.03	Adam	0.05	bf16	10
LATENTSEEK (Self)	Qwen2.5-1.5B-Instruct	1024	2	1 3090	10	Adam	0.2	bf16	10
LATENTSEEK (Self)	Qwen2.5-7B-Instruct	1024	1	1 L40	0.05	Adam	0.2	bf16	10
LATENTSEEK (Self)	Qwen2.5-7B-Instruct	1024	2	1 L40	0.05	Adam	0.2	bf16	10
LATENTSEEK (Self)	Qwen2.5-14B-Instruct	1024	1	1 L40	0.03	Adam	0.1	bf16	10
LATENTSEEK (Self)	Qwen2.5-14B-Instruct	1024	2	1 L40	0.03	Adam	0.1	bf16	10
LATENTSEEK (Self)	Llama-3.1-8B-Instruct	1024	1	1 L40	0.03	Adam	0.2	bf16	10
LATENTSEEK (Self)	Llama-3.1-8B-Instruct	1024	2	1 L40	0.03	Adam	0.2	bf16	10

Table 18: LATENTSEEK (PSRM) Hyperparameters on GSM8K. lr: learning rate.  $\rho$ : fraction ratio

methods	model	max len	prompt idx	#GPU	lr	optimizer	$\rho$	dtype	Max Step
LATENTSEEK (PSRM)	Qwen2-7B-Instruct	1024	1	1 A100	0.03	Adam	0.2	bf16	10
LATENTSEEK (PSRM)	Qwen2-7B-Instruct	1024	2	1 4090	0.03	Adam	0.2	bf16	10
LATENTSEEK (PSRM)	Qwen2.5-1.5B-Instruct	1024	1	1 3090	0.03	Adam	0.2	bf16	10
LATENTSEEK (PSRM)	Qwen2.5-1.5B-Instruct	1024	2	1 3090	0.03	Adam	0.2	bf16	10
LATENTSEEK (PSRM)	Qwen2.5-7B-Instruct	1024	1	1 A100	0.05	Adam	0.2	bf16	10
LATENTSEEK (PSRM)	Qwen2.5-7B-Instruct	1024	2	1 L40	0.05	Adam	0.2	bf16	10
LATENTSEEK (PSRM)	Qwen2.5-14B-Instruct	1024	1	1 A100	0.03	Adam	0.1	bf16	10
LATENTSEEK (PSRM)	Qwen2.5-14B-Instruct	1024	2	1 A100	0.03	Adam	0.1	bf16	10
LATENTSEEK (PSRM)	Llama-3.1-8B-Instruct	1024	1	1 L40	0.03	Adam	0.2	bf16	10
LATENTSEEK (PSRM)	Llama-3.1-8B-Instruct	1024	2	1 L40	0.03	Adam	0.2	bf16	10

els. Sourced from the HuggingFace repository at <https://huggingface.co/datasets/HuggingFaceH4/MATH-500>, it comprises 500 problems derived from the original MATH benchmark developed by OpenAI [\(Lightman et al., 2023\)](#). Encompassing a diverse array of mathematical topics and varying difficulty levels, MATH-500 provides a comprehensive and challenging testbed for assessing model performance in mathematical problem-solving.

**Experimental Details.** For all backbones and both prompts, we use greedy decoding for inference. For the hyperparameters of LATENTSEEK (Self) and LATENTSEEK (PSRM), please refer to Table 19 and Table 20, respectively. The mathematical reasoning prompts we employed in the self-reward mechanism evaluate answers across four dimensions: correctness of the final answer, accuracy of problem comprehension, correctness of numerical calculations, and provision of a clear answer, weighted at 1:1:2:2, with the final score normalized to the range [-1, 0]. The prompt structure is consistently applied across all backbones and both prompts. Notably, for Qwen2-7B-Instruct and Llama-3.1-8B-Instruct in LATENTSEEK (Self) experiment using prompt 1, the weight ratios are adjusted to 1:1:1:2. Additionally, for Qwen2-7B-Instruct, Qwen2.5-1.5B-Instruct, Qwen2.5-7B-Instruct and Llama-3.1-8B-Instruct in LATENTSEEK (Self) experiment using Prompt 1, a new format

1620 Table 19: LATENTSEEK (Self) Hyperparameters on MATH-500. lr: learning rate.  $\rho$ : fraction ratio  
1621

1622 methods	1623 model	1624 max len	1625 prompt idx	1626 #GPU	1627 lr	1628 optimizer	1629 $\rho$	1630 dtype	1631 Max Step
LATENTSEEK (Self)	Qwen2-7B-Instruct	1024	1	1 3090	0.03	Adam	0.1	bf16	10
LATENTSEEK (Self)	Qwen2-7B-Instruct	1024	2	1 3090	0.03	Adam	0.1	bf16	10
LATENTSEEK (Self)	Qwen2.5-1.5B-Instruct	1024	1	1 A100	0.03	Adam	0.1	bf16	10
LATENTSEEK (Self)	Qwen2.5-1.5B-Instruct	1024	2	1 3090	0.03	Adam	0.05	bf16	10
LATENTSEEK (Self)	Qwen2.5-7B-Instruct	1024	1	1 3090	0.03	Adam	0.1	bf16	10
LATENTSEEK (Self)	Qwen2.5-7B-Instruct	1024	2	1 3090	0.03	Adam	0.1	bf16	10
LATENTSEEK (Self)	Qwen2.5-14B-Instruct	1024	1	1 A100	0.03	Adam	0.1	bf16	10
LATENTSEEK (Self)	Qwen2.5-14B-Instruct	1024	2	1 A100	0.05	Adam	0.1	bf16	10
LATENTSEEK (Self)	Llama-3.1-8B-Instruct	1024	1	1 A100	0.03	Adam	0.1	bf16	10
LATENTSEEK (Self)	Llama-3.1-8B-Instruct	1024	2	1 A100	0.03	Adam	0.1	bf16	10

1632 Table 20: LATENTSEEK (PSRM) Hyperparameters on MATH-500. lr: learning rate.  $\rho$ : fraction ratio  
1633

1634 methods	1635 model	1636 max len	1637 prompt idx	1638 #GPU	1639 lr	1640 optimizer	1641 $\rho$	1642 dtype	1643 Max Step
LATENTSEEK (PSRM)	Qwen2-7B-Instruct	1024	1	1 3090	0.03	Adam	0.1	bf16	10
LATENTSEEK (PSRM)	Qwen2-7B-Instruct	1024	2	1 3090	0.03	Adam	0.1	bf16	10
LATENTSEEK (PSRM)	Qwen2.5-1.5B-Instruct	1024	1	1 A100	0.03	Adam	0.1	bf16	10
LATENTSEEK (PSRM)	Qwen2.5-1.5B-Instruct	1024	2	1 A100	0.03	Adam	0.1	bf16	10
LATENTSEEK (PSRM)	Qwen2.5-7B-Instruct	1024	1	1 3090	0.03	Adam	0.1	bf16	10
LATENTSEEK (PSRM)	Qwen2.5-7B-Instruct	1024	2	1 3090	0.03	Adam	0.1	bf16	10
LATENTSEEK (PSRM)	Qwen2.5-14B-Instruct	1024	1	1 A100	0.03	Adam	0.1	bf16	10
LATENTSEEK (PSRM)	Qwen2.5-14B-Instruct	1024	2	1 A100	0.03	Adam	0.1	bf16	10
LATENTSEEK (PSRM)	Llama-3.1-8B-Instruct	1024	1	1 A100	0.03	Adam	0.1	bf16	10
LATENTSEEK (PSRM)	Llama-3.1-8B-Instruct	1024	2	1 A100	0.03	Adam	0.1	bf16	10

1644 reward following the methodology of DeepSeek-R1 is incorporated, with weight of 2 assigned to  
1645 format-based criteria.

## 1647 F.7 AIME2024

1649 We provide details for AIME2024 experiments as follows:

1651 **Dataset.** The American Invitational Mathematics Examination (AIME) is a prestigious com-  
1652 petition designed to challenge high-achieving high school students with complex mathematical  
1653 problems, requiring advanced problem-solving and reasoning skills. The AIME2024 dataset, as  
1654 introduced in this context, serves as a valuable resource for evaluating the capabilities of lan-  
1655 guage models in tackling such sophisticated mathematical tasks. Sourced from the Huggingface  
1656 repository Maxwell-Jia/AIME\_2024 (accessible at [https://huggingface.co/datasets/](https://huggingface.co/datasets/Maxwell-Jia/AIME_2024)  
1657 [Maxwell-Jia/AIME\\_2024](https://huggingface.co/datasets/Maxwell-Jia/AIME_2024)), the AIME2024 dataset comprises 30 meticulously curated prob-  
1658 lems. Although modest in quantity, each problem is deliberately designed to reflect the style, rigor, and  
1659 difficulty of the AIME, thereby providing a robust benchmark for assessing advanced mathematical  
1660 reasoning in computational models.

1661 **Experimental Details.** For all backbones and both prompts, we use greedy decoding for inference.  
1662 For the hyperparameters of LATENTSEEK (Self) and LATENTSEEK (PSRM), please refer to Table 21  
1663 and Table 22, respectively. The mathematical reasoning prompts we employed in the self-reward  
1664 mechanism evaluate answers across four dimensions: correctness of the final answer, accuracy of  
1665 problem comprehension, correctness of numerical calculations, and provision of a clear answer,  
1666 weighted at 1:1:2:2, with the final score normalized to the range [-1, 0]. The prompt structure is  
1667 consistently applied across all backbones and both prompts.

## 1669 G DETAILED FLOPs CALCULATION

1672 This section provides a detailed breakdown of the Floating Point Operations (FLOPs) required for  
1673 the Genius baseline and our proposed LATENTSEEK method, using LLaMA3.1-8B-Instruct as the  
backbone model.

1674 Table 21: LATENTSEEK (Self) Hyperparameters on AIME2024. lr: learning rate.  $\rho$ : fraction ratio  
1675

methods	model	max len	prompt idx	#GPU	lr	optimizer	$\rho$	dtype	Max Step
LATENTSEEK (Self)	Qwen2-7B-Instruct	1024	1	1 4090	0.03	Adam	0.1	bf16	10
LATENTSEEK (Self)	Qwen2-7B-Instruct	1024	2	1 4090	0.03	Adam	0.1	bf16	10
LATENTSEEK (Self)	Qwen2.5-1.5B-Instruct	1024	1	1 3090	0.03	Adam	0.1	bf16	10
LATENTSEEK (Self)	Qwen2.5-1.5B-Instruct	1024	2	1 3090	10	Adam	0.3	bf16	10
LATENTSEEK (Self)	Qwen2.5-7B-Instruct	1024	1	1 4090	0.05	Adam	0.2	bf16	10
LATENTSEEK (Self)	Qwen2.5-7B-Instruct	1024	2	1 4090	0.05	Adam	0.2	bf16	10
LATENTSEEK (Self)	Qwen2.5-14B-Instruct	1024	1	1 A100	0.03	Adam	0.1	bf16	10
LATENTSEEK (Self)	Qwen2.5-14B-Instruct	1024	2	1 A100	0.03	Adam	0.1	bf16	10
LATENTSEEK (Self)	Llama-3.1-8B-Instruct	1024	1	1 4090	0.03	Adam	0.1	bf16	10
LATENTSEEK (Self)	Llama-3.1-8B-Instruct	1024	2	1 A100	0.03	Adam	0.2	bf16	10

1684  
1685 Table 22: LATENTSEEK (PSRM) Hyperparameters on AIME2024. lr: learning rate.  $\rho$ : fraction ratio  
1686

methods	model	max len	prompt idx	#GPU	lr	optimizer	$\rho$	dtype	Max Step
LATENTSEEK (PSRM)	Qwen2-7B-Instruct	1024	1	1 3090	0.03	Adam	0.1	bf16	10
LATENTSEEK (PSRM)	Qwen2-7B-Instruct	1024	2	1 3090	0.03	Adam	0.1	bf16	10
LATENTSEEK (PSRM)	Qwen2.5-1.5B-Instruct	1024	1	1 A100	0.03	Adam	0.1	bf16	10
LATENTSEEK (PSRM)	Qwen2.5-1.5B-Instruct	1024	2	1 A100	0.03	Adam	0.1	bf16	10
LATENTSEEK (PSRM)	Qwen2.5-7B-Instruct	1024	1	1 3090	0.03	Adam	0.1	bf16	10
LATENTSEEK (PSRM)	Qwen2.5-7B-Instruct	1024	2	1 4090	0.05	Adam	0.2	bf16	10
LATENTSEEK (PSRM)	Qwen2.5-14B-Instruct	1024	1	1 A100	0.03	Adam	0.1	bf16	10
LATENTSEEK (PSRM)	Qwen2.5-14B-Instruct	1024	2	1 A100	0.03	Adam	0.1	bf16	10
LATENTSEEK (PSRM)	Llama-3.1-8B-Instruct	1024	1	1 4090	0.03	Adam	0.1	bf16	10
LATENTSEEK (PSRM)	Llama-3.1-8B-Instruct	1024	2	1 A100	0.03	Adam	0.1	bf16	10

1687  
1688 Table 23: Prompt for answer correctness check

```

1689     f"{{math_prefix}}"
1700     "INSTRUCTIONS:"
1701     "Your task is to determine whether the provided answer is
1702     correct."
1703     "Think through the verification process carefully and
1704     logically."
1705     "IMPORTANT RULES:"
1706     "1. Do NOT analyze the steps or methods used to arrive at the
1707     answer."
1708     "2. Only evaluate the final answer's correctness."
1709     "3. Your response must strictly follow the required format:"
1710     f"- If the answer is correct, respond with:
1711     '{VERA_ANSWER_SYMBOL}True'."
1712     f"- If the answer is incorrect, respond with:
1713     '{VERA_ANSWER_SYMBOL}False'."

```

1714 Table 24: Prompt for calculation check

```

1715     f"{{math_prefix}}"
1716     "INSTRUCTIONS:"
1717     "1. EXTRACT CALCULATION EXPRESSIONS: Extract all the
1718     mathematical calculations from the PROPOSED SOLUTION."
1719     "2. INDEPENDENT RECOMPUTATION: Break down the calculations
1720     step-by-step and recompute them."
1721     f"3. VERIFY: Compare your recompuation with the
1722     PROPOSED SOLUTION. If any discrepancy is found, output
1723     '{VERA_ANSWER_SYMBOL}False'. If all steps are correct, output
1724     '{VERA_ANSWER_SYMBOL}True'."
1725     "NOTE: You ONLY need to check calculations (like 1 + 1 = 2, 2 *
1726     3 = 6, etc). Ignore standalone numbers (like 1, 2, 3, etc) that
1727     are not part of a computation."

```

Table 25: Prompt for understanding check

---

```

1728
1729
1730 f"{{math_prefix}}"
1731 "INSTRUCTIONS:"
1732 "1. PROBLEM INTERPRETATION:"
1733 " - Assess if the proposed solution clearly understands the
1734 problem statement."
1735 " - Ensure that the proposed solution addresses all relevant
1736 aspects of the problem, without ignoring any key detail."
1737 " - Flag if the solution misinterprets or overlooks the
1738 problem's core requirements or scope."
1739 "2. ALIGNMENT WITH THE TASK:"
1740 " - Verify that the solution responds to the specific question
1741 or task outlined in the problem statement."
1742 " - Ensure that the solution does not deviate from the
1743 problem's context or provides an unrelated answer."
1744 " - Check if any critical parts of the problem have been
1745 misinterpreted or neglected."
1746 "3. TERMINATION PROTOCOL:"
1747 " - If the solution clearly misinterprets or fails to address
1748 the problem correctly, stop and respond in the exact format:"
1749 f" - '{VERA_ANSWER_SYMBOL}False'"
1750 " - If the solution accurately captures the problem statement
1751 and aligns with the required solution, respond in the exact
1752 format:"
1753 f" - '{VERA_ANSWER_SYMBOL}True'"
1754 "EXAMPLES:"
1755 "[Case 1] Problem: A shop is selling a drink at 1.5 times the
1756 original price. If the original price is $10, what is the new
1757 price?"
1758 " Solution: The new price is  $1.15 * \$10 = \$11.50$ ."
1759 " Assessment: The solution misinterprets the problem by
1760 calculating  $1.15$  times the original price instead of  $1.5$ 
1761 times."
1762 f" Result: '{VERA_ANSWER_SYMBOL}False'"
1763 "[Case 2] Problem: The second cup of coffee is half price. If
1764 the first cup costs $5, how much is the second cup?"
1765 " Solution: The second cup costs  $\$5 * 0.5 = \$2.50$ ."
1766 " Assessment: The solution correctly interprets the price as
1767 half the original price for the second cup."
1768 f" Result: '{VERA_ANSWER_SYMBOL}True'"
1769 "[Case 3] Problem: A pizza has a radius of 8 inches. What is
1770 the area of the pizza?"
1771 " Solution: The area is  $\pi r^2$ , where  $r = 4$  inches. The area is
1772  $16\pi$  square inches."
1773 " Assessment: The solution misinterprets the formula for the
1774 area of a circle by using the radius incorrectly."
1775 f" Result: '{VERA_ANSWER_SYMBOL}False'"
1776 "[Case 4] Problem: A train is moving at 60 km/h towards the
1777 east. What is its velocity after 2 hours?"
1778 " Solution: The velocity is 120 km/h west."
1779 " Assessment: The solution correctly calculates the speed, but
misinterprets the direction as west instead of east."
f" Result: '{VERA_ANSWER_SYMBOL}False'"
"CRITICAL REQUIREMENTS:"
"- Assess whether the solution addresses all parts of the
problem."
"- Ensure the solution does not deviate from the problem's
intent."
"- Use exact output formats specified, showing no tolerance for
misinterpretations."

```

---

1782 Table 26: Prompt for answer completeness check  
1783

---

```

1784 f"{{math_prefix}}"
1785 "INSTRUCTIONS:"
1786 "Your task is to verify whether the solution provides a
1787 complete and final answer."
1788 "Follow these rules carefully:"
1789 "1. Check if the solution reaches a clear and definitive final
1790 answer."
1791 "2. The answer must not be left incomplete, such as:"
1792 " - Ending with an unresolved expression or formula instead of
1793 a computed result."
1794 " - Missing a conclusion or final statement explicitly stating
1795 the final answer."
1796 "3. If the solution is incomplete or lacks a final answer,
1797 immediately stop checking further and respond in the exact
1798 format:"
1799 f" - '{VERA_ANSWER_SYMBOL}False'"
1800 "4. If the solution is complete and provides a final, explicit
1801 answer, respond in the exact format:"
1802 f" - '{VERA_ANSWER_SYMBOL}True'"
1803 "Examples:"
1804 "Example 1:"
1805 "final answer: 8."
1806 f"Your response: '{VERA_ANSWER_SYMBOL}True' (The solution
1807 provides a final, definitive answer of 8.)"
1808 "Example 2:"
1809 "final answer: The area of the circle is  $\pi r^2$ , where  $r = 4$ ."
1810 f"Your response: '{VERA_ANSWER_SYMBOL}False' (The answer ends
1811 with an unresolved formula, not a computed result.)"

```

---

1812 Table 27: Estimated FLOPs for a single forward pass of the LLaMA3.1-8B backbone model with  
1813 a context length of 512 tokens. The total is derived by summing the operational costs for both the  
1814 prefill and decode stages.  
1815

Prefill Stage (512 tokens)		Decode Stage (512 tokens)	
Component	FLOPs	Component	FLOPs
q_proj	17.2G	q_proj	33.6M
k_proj	4.3G	k_proj	8.4M
v_proj	4.3G	v_proj	8.4M
out_proj	17.2G	out_proj	33.6M
gate_proj	60.1G	gate_proj	117M
up_proj	60.1G	up_proj	117M
down_proj	60.1G	down_proj	117M
qk_matmul	2.1G	qk_matmul	4.2M
sv_matmul	2.1G	sv_matmul	4.2M
softmax	41.9M	softmax	81.9K
attn_norm	14.7M	attn_norm	28.7K
mlp_norm	14.7M	mlp_norm	28.7K
attn_add	2.1M	attn_add	4.1K
mlp_add	2.1M	mlp_add	4.1K
mlp_act	4.2M	mlp_act	8.2K
lm_head	525M	lm_head	525M

1836  
1837

## G.1 TOTAL FLOPS FOR GENIUS

1838 On the GSM8K dataset, the Genius framework involves a full training phase on 100000 instances  
1839 followed by testing on 1319 instances. We assume a backward pass is approximately twice as  
1840 computationally expensive as a forward pass.

- 1841 •
- Training FLOPs:**
- For each of the 100000 instances, one forward pass and one full backward
- 
- 1842 pass are required.

1843 
$$\text{FLOPs}_{\text{train}} = 100000 \times (\text{FLOPs}_{\text{forward}} + 2 \times \text{FLOPs}_{\text{forward}}) = 300000 \times \text{FLOPs}_{\text{forward}}$$

- 1844 •
- Testing FLOPs:**
- For each of the 1319 instances, one forward pass is required.

1845 
$$\text{FLOPs}_{\text{test}} = 1319 \times \text{FLOPs}_{\text{forward}}$$

1846 
$$\begin{aligned} \text{FLOPs}_{\text{Genius}} &= \text{FLOPs}_{\text{train}} + \text{FLOPs}_{\text{test}} \\ 1847 &= (300000 + 1319) \times (2.29 \times 10^{11}) \\ 1848 &\approx 6.90 \times 10^{16} \text{ FLOPs} \end{aligned}$$

1852 G.2 TOTAL FLOPS FOR LATENTSEEK

1854 LATENTSEEK operates exclusively at test time on 1319 instances. Each iteration involves two  
1855 forward passes (one for generation, one for reward evaluation) and one partial backward pass through  
1856 only the LM head. From Table 9, the average number of iterations on GSM8K is 1.27.

- 1857 •
- Number of Forward Passes:**
- $1319 \text{ instances} \times 1.27 \text{ iter/inst} \times 2 \text{ forward/iter} \approx 3350$
- 
- 1858 •
- Number of Backward Passes:**
- $1319 \text{ instances} \times 1.27 \text{ iter/inst} \times 1 \text{ forward/iter} \approx 1675$
- 
- 1859 •
- Cost of Backward Pass:**
- The backward pass is only through the LM head (525 MFLOPs
- 
- 1860 forward).

1861 
$$\text{FLOPs}_{\text{backward,LATENTSEEK}} = 2 \times (525 \times 10^6) = 1.05 \times 10^9 \text{ FLOPs}$$

1863 The total estimated FLOPs for LATENTSEEK is:

1864 
$$\begin{aligned} \text{FLOPs}_{\text{LATENTSEEK}} &= (3350 \times \text{FLOPs}_{\text{forward}}) + (1675 \times \text{FLOPs}_{\text{backward,LATENTSEEK}}) \\ 1865 &= (3350 \times 2.29 \times 10^{11}) + (1675 \times 1.05 \times 10^9) \\ 1866 &\approx 7.67 \times 10^{14} + 1.76 \times 10^{12} \\ 1867 &\approx 7.69 \times 10^{14} \text{ FLOPs} \end{aligned}$$

1871 G.3 EFFICIENCY THRESHOLD ANALYSIS

1872 We calculate the number of inference instances ( $x$ ) at which the total computational cost of using the  
1873 pre-trained Genius framework would equal the cost of using LATENTSEEK. This threshold is found  
1874 by solving the following equation, where the left side represents the total cost of Genius and the right  
1875 side represents the total cost of LATENTSEEK for  $x$  inferences:

1876 
$$6.90 \times 10^{16} + x \cdot (2.29 \times 10^{11}) = 1.27 \times (2 \times 2.29 \times 10^{11} + 1.05 \times 10^9) \cdot x$$

1877 Solving for  $x$  yields the threshold:  $x \approx 1.94 \times 10^5$ . This calculation demonstrates that LATENTSEEK  
1878 remains more efficient than the Genius method up to approximately  $1.94 \times 10^5$  inference instances,  
1879 which exceeds the size of Genius's training set ( $1.00 \times 10^5$  examples).

1881 G.4 COMPARISON WITH iCoT

1883 To the best of our knowledge, all latent reasoning methods experience a training stage, and iCoT  
1884 (Deng et al., 2024) is one of the representative. We perform FLOPs comparison with iCoT using  
1885 Llama3.1-8B-Instruct evaluated on the GSM8K test set. As shown in Table 28, the **total FLOPs**  
1886 of LatentSeek are roughly  $10^4$  times smaller than those of iCoT. This efficiency comes from  
1887 **eliminating the need of training**: LatentSeek performs latent updates *only at test time*, and each  
1888 update requires just *a single forward-backward pass of the language-model head*, making the  
1889 procedure efficient. Moreover, as shown in Table 9, LatentSeek typically converges in fewer than two  
update iterations during inference.

Table 28: FLOPs comparison with iCoT (Deng et al., 2024)

Name	Type	# Train Dataset	# Train Epoch	# Test Dataset	Acc (%)	Total FLOPs
LatentSeek	Latent	0	0	1.3K	83.7	$7.69 \times 10^{14}$
iCoT	Latent	37.8K	200	1.3K	47.08	$5.19 \times 10^{18}$

## H LATENT SPACE TRAJECTORIES

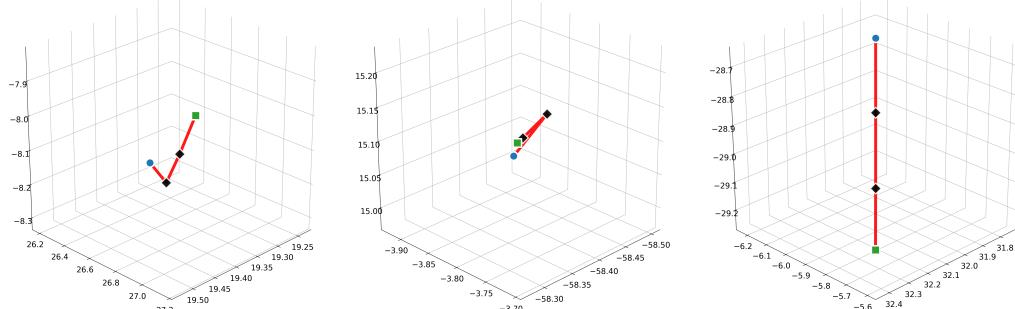


Figure 8: Examples of latent-space optimization trajectories. Blue: initial latent vector; Black: intermediate steps; Green: final latent vector; Red: optimization trajectory. For more trajectories, please refer to Figure 9 and Figure 10

To better understand the optimization dynamics within the continuous latent space, we conduct a visualization study of latent-vector trajectories. Specifically, for each token position  $t$  and optimization step  $k$ , we collect the corresponding latent representations  $z_t^{(k)}$  and apply Principal Component Analysis (PCA) to project these high-dimensional vectors into a three-dimensional space.

As shown in Figure 9 and Figure 10 below, we visualize the projected trajectories of token latents across different positions (1st to 80th). Furthermore, Figure 8 highlights three representative patterns of latent-space optimization. The Blue dot denotes the initial latent vector, the Black dots indicate intermediate latent states, the Green dot marks the final latent vector, and the Red line traces the optimization trajectory. As illustrated in the figures, some token-specific latents exhibit zig-zag search, others follow a consistent unidirectional trajectory, and still others trace an V-shaped search path.

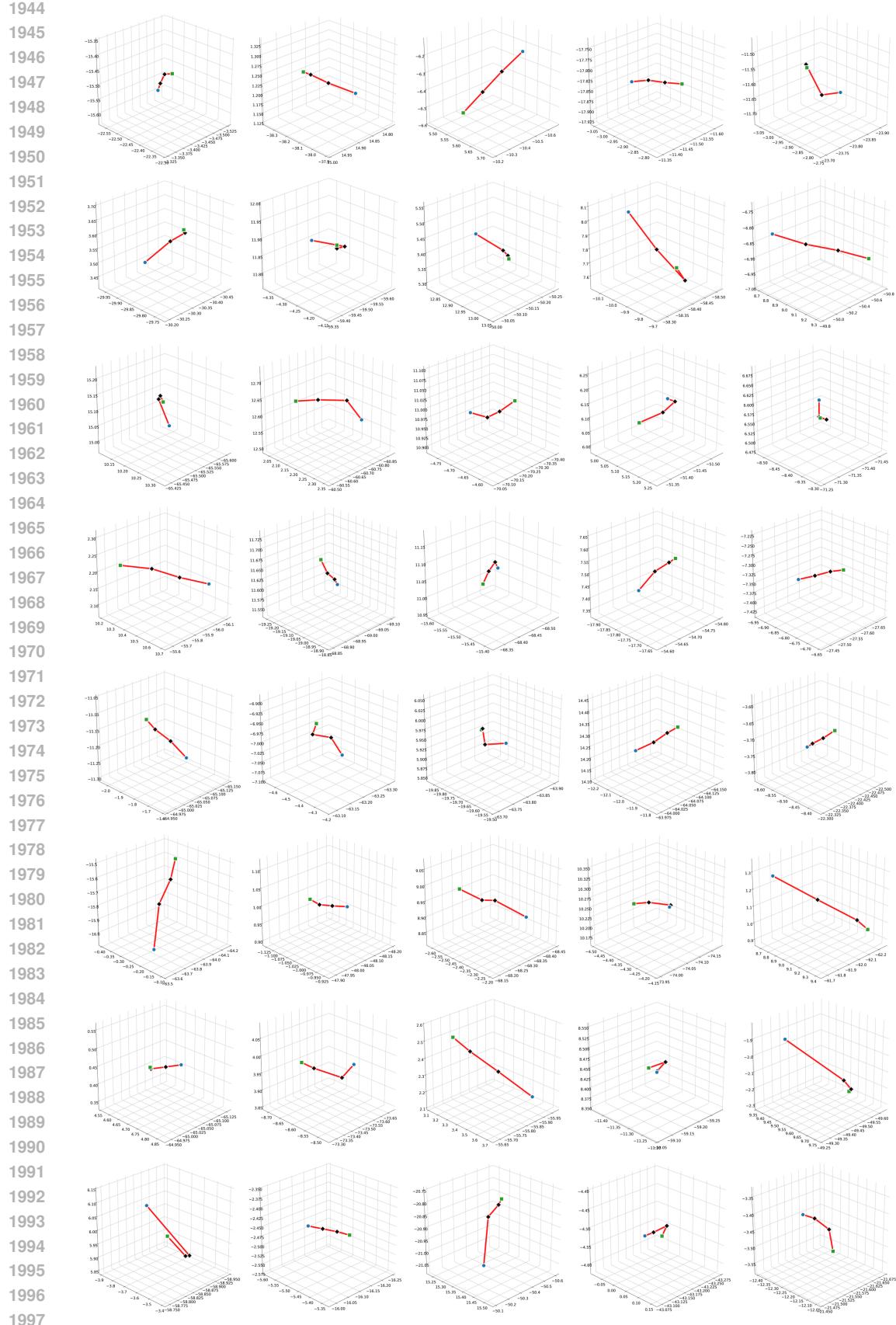


Figure 9: Latent-space trajectories for tokens at positions 1–40. Blue: initial latent vector; Black: intermediate steps; Green: final latent vector; Red: trajectory.

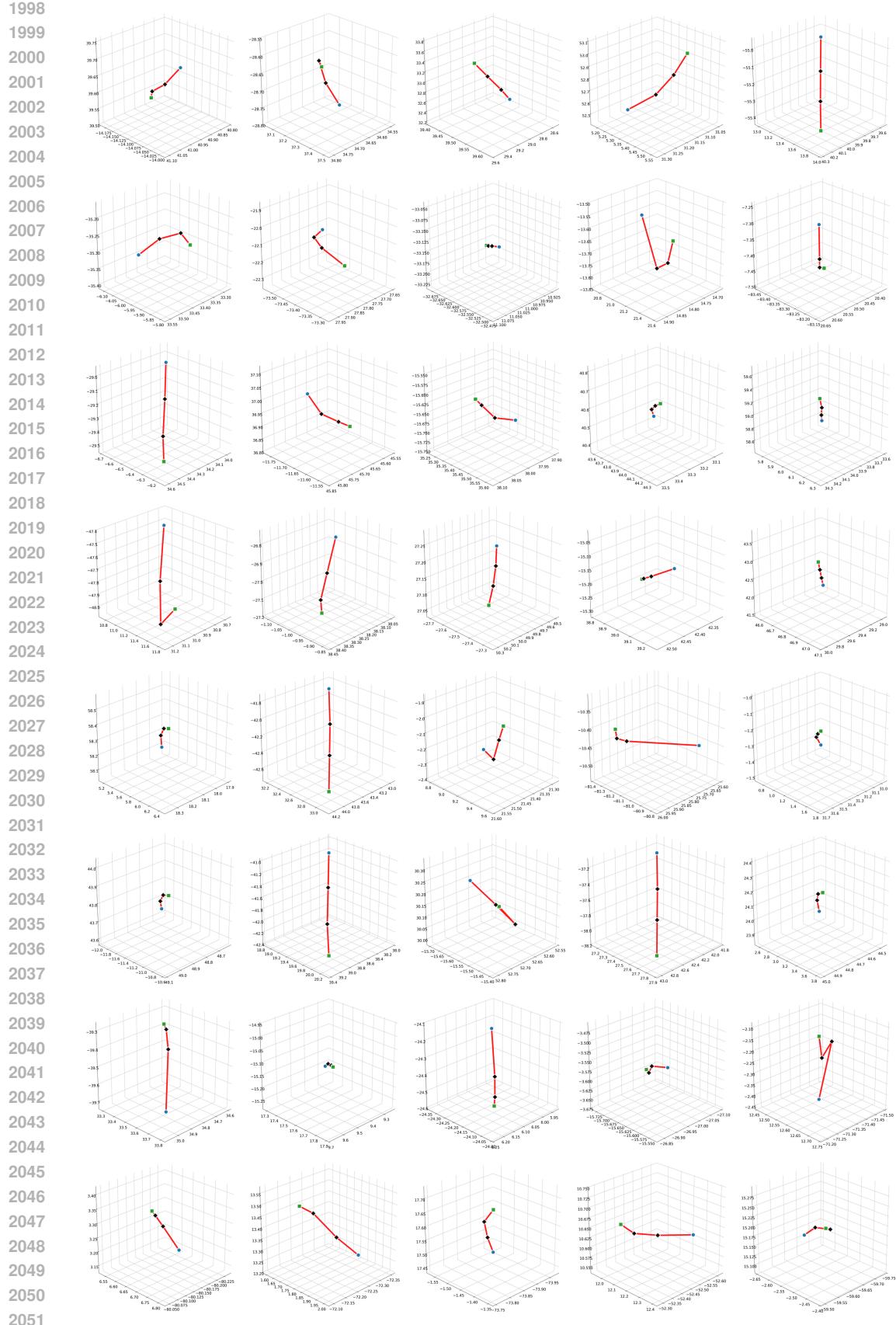


Figure 10: Latent-space trajectories for tokens at positions 41–80. Blue: initial latent vector; Black: intermediate steps; Green: final latent vector; Red: trajectory.

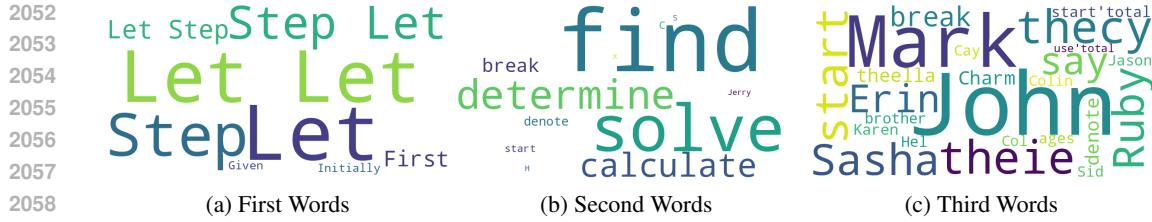


Figure 11: Wordclouds of the first three words of the generated sequence using prompt 2 on the GSM8k dataset with Llama3.1-8B-Instruct as the backbone.

## I QUALITATIVE ANALYSIS AND CASE STUDIES

To gain a deeper understanding of model-generated responses, we analyze the distribution of the first three tokens produced under Prompt 1 in the GSM8K dataset [Cobbe et al. \(2021\)](#), using the Llama3.1-8B-Instruct model. These distributions are visualized in the word cloud shown in Figure 11. The appearance of nonsensical or obscure tokens—such as “theey,” “theella,” and unrelated proper names like “Mark” and “John”—suggests that the reasoning processes employed by language models can diverge significantly from human reasoning, despite generating superficially plausible outputs. Notably, consistent patterns emerge: the first generated tokens are often prepositions (e.g., “let”), the second tokens are primarily verbs (e.g., “find,” “solve”), and the third tokens are predominantly proper names. These findings suggest that **optimal reasoning paths for language models may differ from human strategies and could be more effectively modeled in latent space.**

We present several representative cases in Tables 29 to 35 to demonstrate the divergence between machine reasoning patterns and human cognition processes. These cases reveal that while our method (*OURS*) generates linguistically anomalous expressions such as "total downloads of downloads", "let need calculate to calculate" and "let'll more understand it", it still achieves correct solutions. This suggests that even if the decoded tokens from our updated embeddings lack syntactic and semantic coherence, the logic and answer of the final generated output remain accurate. Notably, although the baseline model produces erroneous answers that appear fundamentally distinct from ours (*OURS*) in textual form, their latent representations exhibit structural proximity - through minimal adjustments to embeddings in the latent space, the original output (*ORIGINAL MODEL*) can be transformed into correct solutions (*OURS*). These empirical evidences yield two critical implications. First, it suggests that **the reasoning paths employed by machines are fundamentally distinct from those of humans**. Second, while model errors may appear substantially deviant from correct answers in textual outputs, they may maintain significant proximity to valid solutions in latent space.

Table 29: Case 1

QUESTION: A new program had 60 downloads in the first month. The number of downloads in the second month was three times as many as the downloads in the first month, but then reduced by 30% in the third month. How many downloads did the program have total over the three months?

TRUE ANSWER: 366

ORIGINAL MODEL: {"thought process": "To find the total number [...] Finally, we add up the number of downloads in all three months to get the total number of downloads.", "final answer": 213}

OURS: {"thought process": "First solve the total downloads of downloads, the three months, we need calculate the downloads of the second month and the third month. [...] Now, we can calculate the total downloads over the three months, total downloads =  $60 + 180 + 126 = 366$ .", "final answer": 366}

2106

2107

2108

2109 QUESTION: Jen enters a lottery by picking 4 distinct numbers  
 2110 from  $S = \{1, 2, 3, \dots, 9, 10\}$ . 4 numbers are randomly chosen from  
 2111  $S$ . She wins a prize if at least two of her numbers match the  
 2112 chosen numbers, and wins the grand prize if all four match.  
 2113 Find the conditional probability of winning the grand prize  
 2114 given that she won any prize.

2114 TRUE ANSWER: 116

2115 ORIGINAL MODEL: {"thought process": "To solve this problem  
 2116 [...] Finally, we can add m and n to get the final answer.",  
 2117 "final answer": 5}

2118 OURS: {"thought process": "First find this problem, let need  
 2119 calculate to calculate the probability number of possible to  
 2120 can choose a prize and the number of possible to can choose the  
 2121 grand prize. [...] Therefore, the answer is  $\$1 + 115 = 116\$$ .",  
 2122 "final answer": 116}

2123

2124

2125

Table 30: Case 2

2126

2127 QUESTION: Two trains leave San Rafael at the same time. They  
 2128 begin traveling westward, both traveling for 80 miles. The next  
 2129 day, they travel northwards, covering 150 miles. What's the  
 2130 distance covered by each train in the two days?

2130 TRUE ANSWER: 230

2131 ORIGINAL MODEL: To solve this problem, we need to break it down  
 2132 into steps. [...] So, the distance covered by each train in the  
 2133 two days is 197.5 miles. \boxed{197.5}

2134 OURS: Let find this,,, let'll more understand it down step  
 2135 two andLet 1: BothThe the trains are same route west both  
 2136 first time on80ward) on same and and can consider they the  
 2137 travel travel same distance of So's denote the common  $xd'$  they  
 2138 both 80 miles on x have write up an following: [...] Since  
 2139 both trains travel the same distance in each direction, the  
 2140 distance covered by each train is 230 miles. The final answer  
 2141 is: \boxed{230}

2142

2143

2144

Table 31: Case 3

2145

2146

2147

2148

2149 QUESTION: Carlos is planting a lemon tree. The tree will cost  
 2150 \$90 to plant. Each year it will grow 7 lemons, which he can  
 2151 sell for \$1.5 each. It costs \$3 a year to water and feed the  
 2152 tree. How many years will it take before he starts earning  
 2153 money on the lemon tree?

2153 TRUE ANSWER: 13

2154 ORIGINAL MODEL: {"thought process": "To find out how many years  
 2155 [...] We will continue this process for each subsequent year,  
 2156 adding the maintenance cost and calculating the income until  
 2157 the total income is greater than the total cost.", "final  
 2158 answer": 7}

2159 OURS: {"thought process": "Let determine the when long years  
 2160 Carlos will take before the to earn earning money from his  
 2161 tree tree, I first to find the revenue revenue for planting the  
 2162 maintaining the tree. [...] Therefore, it will take 13 years  
 2163 for Carlos to start earning money on the lemon tree.", "final  
 2164 answer": 13}

2160

2161

2162

2163

2164 QUESTION: Gunter is trying to count the jelly beans in a jar.  
 2165 He asks his friends how many they think are in the jar. One  
 2166 says 80. Another says 20 more than half the first one. A third  
 2167 says 25% more than the first one. What is their average guess?  
 2168 TRUE ANSWER: 80

2169 ORIGINAL MODEL: {"thought process": "To find the average guess  
 2170 [...] After finding the individual guesses, we can add them  
 2171 up and divide by the number of guesses to find the average.",  
 2172 "final answer": 76}

2173 OURS: {"thought process": "First find the average of, we first  
 2174 to calculate calculate each number guesses of The first friend  
 2175 guesses 80. [...] Then we add the three guesses together,  $80 +$   
 $60 + 100 = 240$ . Finally, we divide by 3 to get the average,  $240$   
 $/ 3 = 80.$ ", "final answer": 80}

2176

2177

2178

2179

Table 33: Case 5

2180

2181 QUESTION: Frankie watches TV after he finishes his homework  
 2182 every night. On Monday and Tuesday, he watched a 1-hour episode  
 2183 of his favorite show each night. On Wednesday, he watched a few  
 2184 episodes of a 30-minute show. On Thursday, he finished homework  
 2185 early and watched a 1-hour episode and a 30-minute show. On  
 2186 Friday, he got to stay up late for the weekend, so he watched  
 2187 two 1-hour episodes. If he watched 7 hours of TV in all, how  
 2188 many 30-minute episodes did he watch on Wednesday?  
 2189 TRUE ANSWER: 3

2190 ORIGINAL MODEL: To determine how many 30-minute episodes  
 2191 Frankie watched on Wednesday, [...] So, Frankie watched  
 2192  $\boxed{197.5}$  30-minute episodes on Wednesday.

2193 OURS: Let determine how many 30-minute episodes Frankie watched  
 2194 on Wednesday, let will to break the total amount he spent  
 2195 watching TV from night and then solve the the much time was  
 2196 spent on Wednesday specifically, we's summarize the time  
 2197 time Frankie watching TV each day: [...] Thus, the number of  
 2198 30-minute episodes Frankie watched on Wednesday is  $\boxed{3}$ .

2199

2200

2201

Table 34: Case 6

2202

2203

2204

2205

2206 QUESTION: Lloyd has an egg farm. His chickens produce 252 eggs  
 2207 per day and he sells them for \$2 per dozen. How much does Lloyd  
 2208 make on eggs per week?  
 2209 TRUE ANSWER: 294

2210 ORIGINAL MODEL: {"thought process": "To find out how much Lloyd  
 2211 makes on eggs per week, [...] Finally, we multiply the number  
 2212 of dozens by the price per dozen to get the total amount Lloyd  
 2213 makes in a week.", "final answer": 5880}

2214 OURS: {"thought process": "First calculate Lloyd how much Lloyd  
 2215 makes per eggs per week, first need calculate to determine out  
 2216 the many eggs he produces in a week. [...] Total amount per  
 2217 week =  $147 * 2 = \$294$ . Therefore, Lloyd makes \$294 per week on  
 2218 eggs..", "final answer": 294}

2219

2220

2221

2222

2223

2214  
2215

## J COMPUTATIONAL RESOURCES

2216  
2217

All experiments are performed with a single A100, or a single L40, or a single 4090, or a single 3090.

2218

## K THE USE OF LARGE LANGUAGE MODELS (LLMs)

2219

Our use of Large Language Models (LLMs) was strictly limited to polishing the language and generating figures for the manuscript. All underlying research and intellectual content of this paper, including the LATENTSEEK framework, its theoretical foundations, experimental design, and the analysis of results, was completed entirely by the authors without assistance from LLMs.

2220

2221

2222

2223

2224

2225

2226

2227

2228

2229

2230

2231

2232

2233

2234

2235

2236

2237

2238

2239

2240

2241

2242

2243

2244

2245

2246

2247

2248

2249

2250

2251

2252

2253

2254

2255

2256

2257

2258

2259

2260

2261

2262

2263

2264

2265

2266

2267