Seek in the Dark: Reasoning via Test-Time Instance-Level Policy Gradient in Latent Space

Hengli Li [1,2][*], **Chenxi Li** [2,3][*], **Tong Wu** [2], **Xuekai Zhu** [2,4], **Yuxuan Wang** [2], **Zhaoxin Yu** [5], **Eric Hanchen Jiang** [6], **Song-Chun Zhu** [1,2,3], **Zixia Jia** [2], **Ying Nian Wu** [6][✉] **and Zilong Zheng** [2][✉]

[1] Institute for Artificial Intelligence, Peking University
[2] NLCo Lab, Beijing Institute for General Artificial Intelligence
[3] Department of Automation, Tsinghua University   [4] Shanghai Jiao Tong University
[5] Institute of Automation, Chinese Academy of Sciences   [6] University of California, Los Angeles

`lihengli@stu.pku.edu.cn, lichenxi23@mails.tsinghua.edu.cn, ywu@stat.ucla.edu, zlzheng@bigai.ai`

Reasoning ability, a core component of human intelligence, continues to pose a significant challenge for Large Language Models (LLMs) in the pursuit of AGI. Although model performance has improved under the training scaling law, significant challenges remain, particularly with respect to training algorithms—such as catastrophic forgetting—and the limited availability of novel training data. As an alternative, test-time scaling enhances reasoning performance by increasing test-time computation without parameter updating. Unlike prior methods in this paradigm focused on token space, we propose leveraging latent space for more effective reasoning and better adherence to the test-time scaling law. We introduce LATENTSEEK, a novel framework that enhances LLM reasoning through **Test-Time Instance-level Adaptation (TTIA) within the model's latent space**. Specifically, LATENTSEEK leverages policy gradient to iteratively update latent representations, guided by self-generated reward signals. LATENTSEEK is evaluated on a range of reasoning benchmarks, including GSM8K, MATH-500, and AIME2024, across multiple LLM architectures. Results show that LATENTSEEK consistently outperforms strong baselines, such as Chain-of-Thought prompting and fine-tuning-based methods. Furthermore, our analysis demonstrates that LATENTSEEK is highly efficient, typically converging within a few iterations for problems of average complexity, while also benefiting from additional iterations, thereby highlighting the potential of test-time scaling in the latent space. These findings position LATENTSEEK as a lightweight, scalable, and effective solution for enhancing the reasoning capabilities of LLMs.

    ⭘   **Code**   https://github.com/bigai-nlco/LatentSeek
    🌐   **Project**   https://bigai-nlco.github.io/LatentSeek/

1. Introduction

Large Language Models (LLMs) have demonstrated exceptional performance across a wide array of tasks, particularly in complex reasoning and deductive analysis (Brown et al., 2020; Chowdhery et al., 2022; OpenAI, 2023; Zhao et al., 2025). Despite these advancements, LLMs still exhibit difficulties in reasoning, particularly in tasks that demand structured thinking and meticulous step-by-step analysis

---

[*] Equal Contributions. [✉] *Corresponding author(s): Ying Nian Wu, Zilong Zheng.*
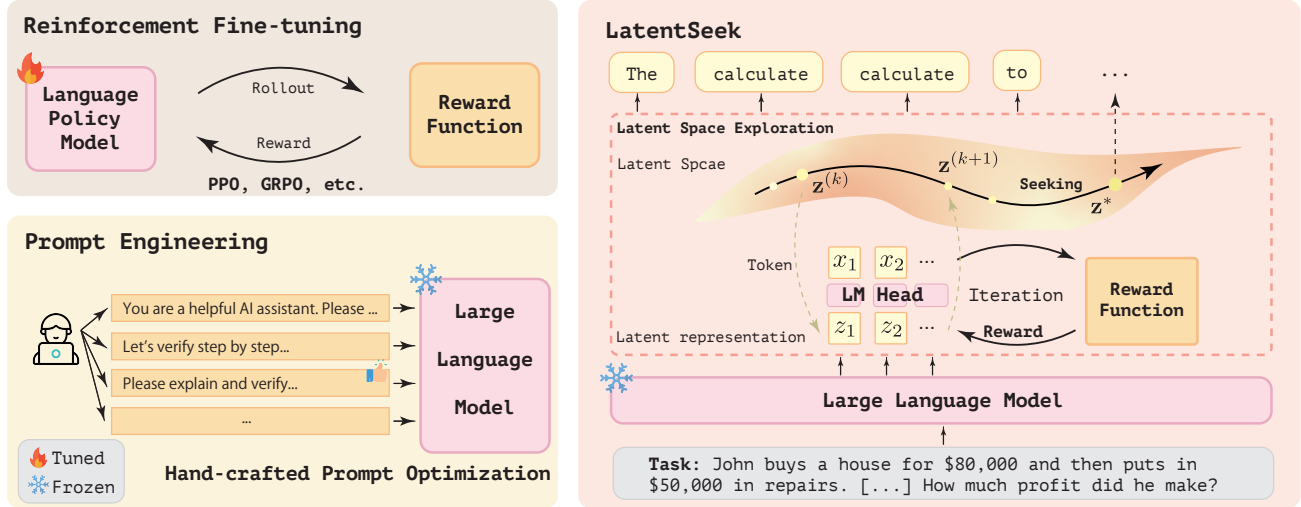
Figure 1: Comparison of LATENTSEEK with RL-based fine-tuning and Prompt Engineering. RL-based fine-tuning methods generally require iterative updates to model parameters guided by reward signals. Prompt engineering approaches depend heavily on manually designed prompts. In contrast, **LATENTSEEK** performs optimization within the latent space. Of note, the output of LATENTSEEK may be incoherent and semantically ungrounded (§3.6).

(Wei et al., 2022; Kojima et al., 2022). Common approaches to enhancing the reasoning capabilities of LLMs involve training model parameters on reasoning-centric datasets or providing reasoning-oriented feedback (Ouyang et al., 2022; Bai et al., 2022; DeepSeek-AI, 2025). However, training methodologies such as supervised fine-tuning, reinforcement learning, and test-time training (Sun et al., 2020; Hardt and Sun, 2023) need to update model parameters, which incur substantial computational costs and present potential risks, including catastrophic forgetting of general competencies (Luo et al., 2025). Moreover, the widely used reinforcement learning approach may reduce the model's exploration capacity (Yue et al., 2025) and, in some cases, lead to the generation of overly verbose responses (Aggarwal and Welleck, 2025; Wu et al., 2025). In light of these challenges, we focus on an alternative paradigm, **Test-Time Instance-Level Adaptation (TTIA)**, that *does not require parameter updating* and operates on a *per-instance basis* during the *testing phase*.

To enhance TTIA performance in reasoning, recent advances (Deng et al., 2022; Hao et al., 2024) reveal that reasoning capabilities can be internalized within the **latent space**[1] through fine-tuning. However, the training strategies adopted in these works tend to substantially modify the latent space and fail to fully leverage its semantic richness. As a result, their performance remains inferior to that of Chain-of-Thought (CoT). Nevertheless, these studies provide evidence supporting the adequacy of semantic information encoded within latent representations, i.e., the hidden states corresponding to language tokens.

Motivated by these observations, we present the first attempt to perform seeking in the latent space by introducing **LATENTSEEK**, a framework that significantly enhances instance-level reasoning at test time. LATENTSEEK introduces updated instance-specific latent representations that steer the pre-trained model's reasoning process without modifying its parameters. These latent representations act as a planning or control mechanism that guides the model toward better reasoning paths for each specific problem instance. We optimize latent representations at test time using the policy gradient method (Williams, 1992) to maximize reward (§2.3). Specifically, for each reasoning problem, we update the token-wise latent representations using guidance from the reward function, treating them as independent variables. In each iteration, the updated latent representations are decoded into tokens,

---

[1]In this work, we take the convention (Kong et al., 2025; Hao et al., 2024) that treats the transformers' output space ahead of the 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.

which serve as inputs for computing the reward. Importantly, the reward function operates in a self-rewarding manner, relying solely on the model's internal capabilities without incorporating any external information. The process continues until the reward exceeds a predefined threshold or the maximum number of iterations is reached.

Our innovative latent space TTIA method is simple yet surprisingly effective: Notably, LATENTSEEK achieves an average improvement of 10.75% over the CoT method on the GSM8K dataset, 3.93% on MATH-500, and 4.73% on AIME2024. Furthermore, when using LlaMA3.1-8B-Instruct as the backbone, LATENTSEEK surpasses prior arts including SimpleRL-Zoo (Williams, 1992) (+18.1%) and Genius (Xu et al., 2025a) (+12.7%), on the GSM8K and MATH-500 datasets. To further assess the potential of the latent space, we conduct idealized experiments utilizing a perfect verifier that relies on ground truth and provides sparse rewards. The experimental results demonstrate a significant performance improvement, with an average gain of 19.12% points over CoT reasoning across all evaluated settings. This highlights the effectiveness of the knowledge elicitation mechanism employed by LATENTSEEK. Further experiments (see §3.4) reveal that the model's performance at test time improves proportionally with the number of update iterations. Notably, when equipped with an ideal verifier, LATENTSEEK is capable of elevating the performance of a 1.5B parameter model from 54.8% to 82.8% on the MATH-500 dataset, approaching the performance of OpenAI's o1-preview model. These findings suggest that **test-time scaling can be effectively implemented in the latent space**, offering a viable alternative to conventional token-space scaling strategies(Liu et al., 2025; Yeo et al., 2025; Xu et al., 2025b).

2. Test-Time Instance-Level Adaptation (TTIA) in Latent Space

2.1. Problem Formulation: TTIA for Reasoning Problems

Given a reasoning problem instance $\mathbf{c}$ as context prompt, a pre-trained auto-regressive language model π, the probability distribution over a reasoning token sequence $\mathbf{x} = (x_1, x_2, \ldots, x_T)$ is:

$$\pi(\mathbf{x}|\mathbf{c}) = \prod_{t=1}^{T} \pi(x_t \mid \mathbf{x}_{<t}, \mathbf{c}), \tag{1}$$

where $\mathbf{x}_{<t}$ denotes the sequence of tokens preceding position t. The objective of solving for $\mathbf{c}$ is to find a reasoning sequence $\mathbf{x}$, such that the final answer $a \sim \pi(\cdot|\mathbf{x}, \mathbf{c})$, generated following Equation (1), is consistent with the ground truth. However, at test time, the ground truth is unknown, and thus a reward function $R(\mathbf{x}, \mathbf{c})$ is introduced to evaluate the reasoning token sequence. Concluding the above, the TTIA objective for a reasoning problem is as follows:

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

Please refer to Appendix B for examples.

2.2. TTIA with Policy Gradient in Latent Space

To solve the problem in Equation (2), we reformulate the task as optimizing over a sequence of latent representations rather than directly searching for reasoning tokens. Specifically, for a given input sequence $\mathbf{x}$, we denote a corresponding sequence of latent representations $\mathbf{z} = (z_1, z_2, \ldots, z_T)$, where $z_t \in \mathbb{R}^d$ lies in the latent space of x_t. To identify the optimal sequence of latent representations, TTIA in the latent space aims to optimize the following objective:

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

Based on this formula, we will present the sampling strategy and optimization procedure.

Independent Sampling. For concrete calculation of $\mathbf{x} \sim \pi(\mathbf{x}|\mathbf{z})$, we start by pointing out that the tokens are conditionally independent given their respective latent representations:

$$\pi(\mathbf{x}|\mathbf{z}) = \prod_{t=1}^{T} \pi(x_t|z_t), \tag{4}$$

which implies that the sampling can be carried out on a token-by-token basis.

Test-Time Optimization of Latent Representations. We optimize the latent representations using a direct policy gradient approach based on REINFORCE (Williams, 1992). Assuming the *independence of the latent representations* (see Appendix A for a theoretical justification), the update process is:

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

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

$$\nabla_{\mathbf{z}} \mathcal{J}(\mathbf{z}) = \mathbb{E}_{\mathbf{x} \sim \pi(\mathbf{x}|\mathbf{z})} \left[R(\mathbf{x}, \mathbf{c}) \nabla_{\mathbf{z}} \log \pi(\mathbf{x}|\mathbf{z}) \right]. \tag{6}$$

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

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

where the expectation is approximated using the empirical mean in practical implementation.

2.3. LATENTSEEK Algorithm

Algorithm 1 LATENTSEEK

Require: Problem $\mathbf{c}$, learning rate η, pre-trained model π, reward threshold τ, fraction ratio $\rho \in (0, 1]$
  $\mathbf{z}^{(0)} \leftarrow \pi(\mathbf{c})$ ▷ Initial latent representations before the LM head.
  $r, \mathbf{x} \leftarrow R(\mathbf{x}, \mathbf{c})$ ▷ Self Reward with Equation (10) and the final answer.
  $T \leftarrow |\mathbf{x}|$
  $\mathbf{z}^{(0)} \leftarrow [z_1^{(0)}, z_2^{(0)}, \ldots, z_{\rho T}^{(0)}]$ ▷ Keep a ρ fraction, such as 20%
  while $k \leq K$ and $r \leq \tau$ **do**   ▷ K is typically 10
   $\mathbf{z}^{(k)} \leftarrow \mathbf{z}^{(k-1)} + \eta \nabla_{\mathbf{z}} \mathcal{J}(\mathbf{z}^{(k-1)})$ ▷ Independently update. $\nabla_{\mathbf{z}} \mathcal{J}(\mathbf{z}^{(k-1)})$: Equation (7).
   $x_t \sim \pi(x_t|z_t^{(k)}), t = 1, 2, \ldots, \rho \cdot T$ ▷ Decode tokens: Equation (8).
   $r, \tilde{\mathbf{x}} \leftarrow R([x_1, x_2, \ldots, x_{\rho T}], \mathbf{c})$ ▷ Self Reward with Equation (10) and the final answer.
  end while
  return $\tilde{\mathbf{x}}$

The LATENTSEEK algorithm is described in Algorithm 1. This algorithm iteratively refines the latent representations based on the rewards of generated reasoning paths, effectively performing a guided search through the reasoning space specific to the given problem instance. After each refinement step, the latent representations are decoded into tokens to calculate a reward signal. This signal is then employed to direct the search process in the subsequent iteration. Along with the reward signal, the final output $\tilde{\mathbf{x}}$ is also explicitly provided. The process runs for a small number of iterations (typically 2-10), stopping early if the reward exceeds a threshold.

Decoding Strategy. Greedy decoding is employed as an approximation method for sampling to enhance computational efficiency:

$$x_t = \arg\max_{v \in V} \pi(v|z_t), \tag{8}$$

where V denotes the vocabulary set, t represents the token position.

Reward Function. The reward function $R(\mathbf{x}, \mathbf{c})$ is defined as follows: Given a sequence of decoded tokens $\mathbf{x}$, the pre-trained model is guided to generate a complete sequence $\tilde{\mathbf{x}}$, which encapsulates the final answer. This sequence is obtained by greedy decoding of autoregressive generation:

$$\tilde{\mathbf{x}} = \arg\max_{\tilde{\mathbf{x}}} \pi(\tilde{\mathbf{x}} \mid \mathbf{x}, \mathbf{c}). \tag{9}$$

Subsequently, the sequence $\tilde{\mathbf{x}}$ is evaluated using a self-reward prompt to compute the reward:

$$R(\mathbf{x}, \mathbf{c}) \sim \pi(\cdot \mid \tilde{\mathbf{x}}, \mathbf{c}, \text{prompt}_{\text{self-reward}}). \tag{10}$$

Noticed that, in the algorithm, the complete answer $\tilde{\mathbf{x}}$ is also returned.

Enhancing Techniques. To ensure more stable and efficient optimization, we propose the integration of two complementary techniques. First, *CoT Initialization*: The initial latent representation, denoted as $\mathbf{z}^{(0)}$, 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 $\mathbf{z} = [z_1, z_2, \ldots, z_T]$, we propose to optimize only a subsequence $[z_1, z_2, \ldots, z_{\rho T}]$, where $\rho \in (0, 1]$ is a hyperparameter. This strategy reduces computational cost and promotes stable optimization through the careful selection of ρ. The hyperparameter ρ 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

In this section, we present our experimental setup, results, and analysis. Additional implementation details are provided in Appendices C.1 to C.6.

3.1. Experimental Setup

In this subsection, we list our experimental setup.

Reward Mechanism. We employ the mathematical reasoning prompts introduced by Lifshitz et al. (2025) to elicit the model's self-reward computation. The same prompt structure is applied consistently across all tasks and models. For specific tasks and models, we further incorporate a format-based reward (DeepSeek-AI, 2025). To further illustrate the potential of our paradigm, we introduce a Perfect Sparse Reward Model (PSRM), the details of which are discussed in Section 3.3.

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

Backbones. In order to evaluate the generalizability of LATENTSEEK, we conduct experiments with pre-trained LLMs of different families and sizes: Qwen2-7B-Instruct, Qwen2.5-1.5B-Instruct, Qwen2.5-7B-Instruct, Qwen2.5-14B-Instruct (Yang et al., 2024; Team, 2024a), LLaMA3.1-8B-Instruct (Team, 2024b), and Mistral-7B-Instruct-v0.3 (Jiang et al., 2023).

Benchmarks. Following Deng et al. (2024) and Liu et al. (2025), we focus on mathematical reasoning for evaluation. We conduct experiments on three datasets: GSM8K (Cobbe et al., 2021), MATH-500 (Hendrycks et al., 2021), and AIME2024. GSM8K and MATH-500 are commonly used for evaluating the performance of reasoning systems, thus providing a solid basis for comparative analysis. AIME2024, on the other hand, represents a more challenging dataset designed to assess the robustness and effectiveness of our proposed method in complex problem-solving scenarios.

Baselines. We compare our methods against several established baselines:

- Prompting (Training-Free): CoT (Wei et al., 2022) and Few-Shot CoT (Lambert et al., 2025).
- Explicit Search (Training-Free): Best-of-N (BoN) represents a highly effective search strategy, as demonstrated by Liu et al. (2025).
- Reinforcement Learning: (1) Self Reward: Self-Rewarding (Yuan et al., 2025), ScPO (Prasad et al., 2024), CoH (Liu et al., 2023a), and Genius (Xu et al., 2025a). (2) Verifiable Reward: SimpleRL-Zoo (Zeng et al., 2025), GRPO (GSM8K train set), and SPIN (Chen et al., 2024a).
- Latent Chain-of-Thought: Although iCoT (Deng et al., 2024) requires an augmented training dataset for GSM8K (Cobbe et al., 2021), it remains a computationally efficient baseline and is a canonical example of the latent CoT.
- Supervised Fine-Tuning (SFT): Following (Xu et al., 2025a), we apply SFT on Magpie 25K. For GSM8K, which includes a training set, we also report SFT performance on that set.

Table 1: Accuracy results (%) on GSM8K (Cobbe et al., 2021), MATH-500 (Hendrycks et al., 2021), and AIME2024 datasets. The best performances, excluding *Perfect Sparse Reward*, are highlighted in bold, and the second-best performances are underlined. Numbers highlighted in red indicate the performance gap relative to the CoT method. The symbol † denotes methods that require parameter updates.

Model / Methods	Qwen2 7B	Qwen2.5 1.5B	Qwen2.5 7B	Qwen2.5 14B	LLaMA3.1 8B	Mistral 7B	Avg
GSM8K: Prompt 1							
CoT	68.01	68.08	88.48	92.03	50.19	23.28	65.01
BoN ($N = 3$)	79.76	68.31	89.08	92.27	72.93	24.11	71.30
SFT†	65.86	49.20	72.55	82.39	40.33	24.92	55.88
SFT (Magpie 25K)†	76.50	66.48	83.01	90.30	70.81	13.72	66.80
iCoT† (Deng et al., 2024)	47.54	23.28	41.02	-	47.08	49.13	41.61
LATENTSEEK (*Self*)	84.38+16.37	69.37+1.29	89.46+0.98	92.49+0.46	78.54+28.35	23.88+0.60	73.02+8.01
LATENTSEEK (*PSRM*)	92.80+24.79	85.44+17.36	93.93+5.45	95.91+3.88	88.55+38.36	65.96+42.68	87.10+22.09
GSM8K: Prompt 2							
CoT	65.20	15.31	66.41	91.81	69.07	41.70	58.25
BoN ($N = 3$)	61.33	8.49	74.04	92.27	75.97	43.06	59.19
LATENTSEEK (*Self*)	80.21+15.01	44.20+28.89	85.06+18.65	92.72+0.91	83.70+14.63	44.58+2.88	71.74+13.49
LATENTSEEK (*PSRM*)	92.80+27.60	67.48+52.17	93.78+27.37	96.13+4.32	94.77+25.70	78.24+36.54	87.20+28.95
MATH-500: Prompt 1							
CoT	51.40	54.80	72.80	77.20	47.60	16.40	53.37
BoN ($N = 3$)	53.40	47.40	75.40	78.80	51.20	16.60	53.80
SFT (Magpie 25K)†	46.6	44.40	55.40	68.2	31.00	3.2	41.46
LATENTSEEK (*Self*)	57.40+6.00	55.60+1.80	75.60+2.80	80.00+2.80	54.60+7.00	17.20+0.60	56.73+3.36
LATENTSEEK (*PSRM*)	75.80+24.40	75.80+21.00	86.40+13.60	87.20+10.00	74.60+27.00	41.80+25.40	73.60+20.23
MATH-500: Prompt 2							
CoT	37.40	27.60	53.80	68.00	40.40	8.20	39.23
BoN ($N = 3$)	41.60	24.80	55.80	64.20	44.40	11.60	40.40
LATENTSEEK (*Self*)	44.80+7.39	32.20+4.60	57.60+3.80	71.00+3.00	47.00+6.60	9.80+1.60	43.73+4.50
LATENTSEEK (*PSRM*)	67.80+30.40	57.60+30.00	77.20+23.40	86.00+18.00	66.40+26.00	31.40+23.20	64.39+25.16
AIME2024: Prompt1							
CoT	0.0	3.33	6.67	10.00	0.00	0.00	3.33
BoN ($N = 3$)	0.00	0.00	10.00	16.67	0.00	0.00	4.45
SFT (Magpie 25K)†	3.33	0.0	3.33	10.00	3.33	0.00	3.33
LATENTSEEK (*Self*)	3.33+3.33	6.67+3.33	13.33+6.37	16.67+6.67	10.00+10.00	3.33+3.33	8.33+4.45
LATENTSEEK (*PSRM*)	13.33+13.33	6.67+3.33	16.67+10.00	26.67+16.67	10.00+10.00	3.33+3.33	12.78+9.45
AIME2024: Prompt2							
CoT	0.00	0.00	0.00	3.33	0.00	0.00	0.56
BoN ($N = 3$)	3.33	0.00	6.67	10.00	6.67	0.00	4.45
LATENTSEEK (*Self*)	3.33+3.3	3.33+3.3	13.33+13.33	10.00+6.67	6.67+6.67	0.00+0.00	5.55+5.00
LATENTSEEK (*PSRM*)	3.33+3.33	6.67+6.67	13.33+13.33	23.33+20.00	10.00+10.00	0.00+0.00	9.44+8.88

3.2. State-of-the-art Test-time Reasoning Performance

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

Best Performance on GSM8K (Cobbe et al., 2021) and MATH-500 (Hendrycks et al., 2021). As demonstrated in Table 2, using LLaMA3.1-8B-Instruct as the backbone, our method outperforms all baseline approaches across both the GSM8K and MATH-500 datasets. Specifically, on the GSM8K dataset, our method achieves an improvement of 14.6% points over the CoT baseline, and a 7.0%-point gain on the MATH-500 dataset. In comparison to the training-free BoN approach, our method yields improvements of 7.7% points and 3.4% points on GSM8K and MATH-500, respectively. When compared to the SFT (Magpie 25K) (Xu et al., 2025a), the minimum improvement across both datasets is 12.9% points. Moreover, our approach surpasses the state-of-the-art self-reward RL-based method,

Table 2: Accuracy score (%) compared with more baseline methods on GSM8K and MATH-500 datasets with Llama3.1-8B as backbone.

Methods	Supervision	Train	Backbone	GSM8K	MATH-500	Average
CoT	-	✗	Instruct	69.1	47.6	58.3
Few-Shot CoT (Lambert et al., 2025)	-	✗	Instruct	83.4	42.5	63.0
Self-Rewarding (Yuan et al., 2025)	Self	✔	Instruct	76.0	30.2	53.1
ScPO (Prasad et al., 2024)	Self	✔	Instruct	71.1	31.00	51.1
SPIN (Chen et al., 2024a)	Data	✔	Instruct	74.9	31.5	53.2
CoH (Liu et al., 2023a)	Self	✔	Instruct	74.4	32.3	53.3
Genius (Xu et al., 2025a)	Self	✔	Instruct	78.3	34.6	56.5
iCoT (Deng et al., 2024)	Data	✔	Instruct	47.1	-	47.1
BoN ($N = 3$)	Self	✗	Instruct	76.0	51.2	63.6
SFT (Magpie 25K)	Data	✔	Instruct	70.8	31.0	50.9
GRPO (GSM8K Train)	Data	✔	Instruct	-	50.2	50.2
SimpleRL-Zoo (Zeng et al., 2025)	Data	✔	Base	79.2	23.0	51.1
LATENTSEEK	Self	✗	Instruct	**83.7**	**54.6**	**69.2**
LATENTSEEK	PSRM	✗	Instruct	94.8	74.6	84.7

Genius (Xu et al., 2025a), by 5.4% points on GSM8K and 20.0 points on MATH-500. Notably, it also outperforms iCoT (Deng et al., 2024)—a latent method trained on an augmented GSM8K dataset—by 36.6% points on the GSM8K test set. Even when compared to reinforcement learning with verifiable feedback methods, like SimpleRL-Zoo (Zeng et al., 2025), which was trained on the base version of the model, our method exceeds it by an average of 9.1% points across both datasets. These results underscore the effectiveness of our approach in tackling reasoning tasks.

Challenging Problems — AIME2024. AIME2024 results are listed in Table 1. AIME2024 is a highly challenging mathematics benchmark consisting of only 30 problems, designed to evaluate advanced reasoning capabilities. Experimental results demonstrate the power and potential of our method in complex settings. As shown in Table 1, our approach consistently outperforms all baselines, achieving an average improvement of 4.73% points over CoT across all model families and prompt configurations. Notably, even weaker models such as Mistral and Qwen2.5-1.5B-Instruct benefit from our method, with average gains of 1.67% and 3.33% points compared with CoT, respectively. The largest improvement is observed on AIME2024 (prompt 1), where our method surpasses the CoT baseline by 13.33% points and outperforms BoN by 6.67% points using Qwen2.5-7B-Instruct.

Generalizability Across Backbones. As presented in Section 3, the generalizability of our approach with respect to backbone models can be examined along two dimensions: (1) *Model Families*: LATENTSEEK demonstrates superior performance across multiple model families. Specifically, it achieves the best results when using backbones from the Qwen2, Qwen2.5, and LLaMA3.1 series. Additionally, it ranks first in four out of six evaluation settings and second in one when employing a relatively weaker backbone from the Mistral family. The suboptimal performance of the Mistral-based model is attributed to its limited capacity to function effectively as a reward model. (2) *Model Size*: At the 1.5B parameter scale, our method consistently outperforms all baseline models across diverse datasets and prompt types, with a particularly notable gain of 28.89% points over the CoT baseline on the GSM8K dataset (Prompt 2). At the 7–8B scale, models based on Qwen2, Qwen2.5, and LLaMA3.1 consistently surpass all baselines. In particular, Qwen2 yields an average improvement of 11.75% points over BoN on GSM8K. At the 14B scale, our approach continues to outperform all baselines, achieving a significant 6.67%-point improvement on the AIME2024 dataset. These results provide robust evidence of our method's generalizability across diverse model families and scales.

Generalizability on Model-specific prompt. The Qwen2.5 series was explicitly trained using Prompt 1 (Team, 2024a); nevertheless, our methods still achieve notable performance gains. Specifically, the proposed method attains average scores of 83.77% on GSM8K and 70.40% on MATH-500, reflecting
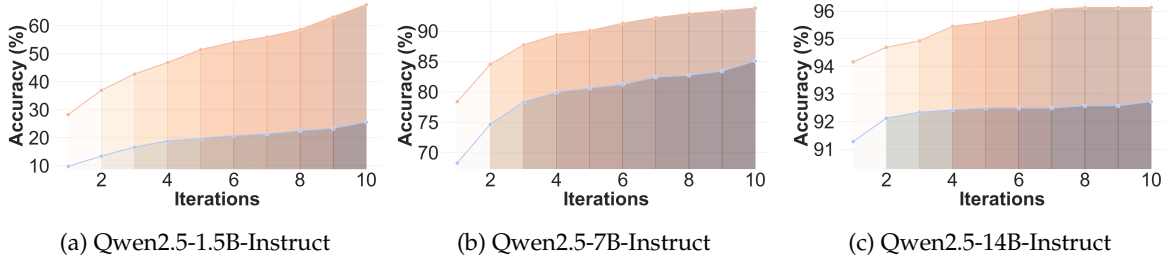
(a) Qwen2.5-1.5B-Instruct   (b) Qwen2.5-7B-Instruct   (c) Qwen2.5-14B-Instruct

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

Table 3: 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. TTS: Liu et al. (2025).

Model	Size	Type	MATH-500		AIME2024	
			Acc	K	Acc	K
GPT-4o	-	CoT	74.6	-	9.3	-
o1-preview	-	CoT	85.5	-	44.6	-
Llama-3.1-70B-Inst.	70B	CoT	65.2	-	16.7	-
Llama-3.1-405B-Inst.	405B	CoT	71.4	-	23.3	-
Qwen2.5-1.5B-Inst.	1.5B	TTS (Liu et al., 2025)	81.8	256.0	20.0	256.0
LATENTSEEK *(PSRM)* (Qwen2.5-1.5B-Inst.)	1.5B	Latent Search	82.8	61.8	23.3	211.8

improvements of 0.91% and 2.47% points, respectively, despite the strong baseline performance achieved by CoT. Additionally, significant improvements are consistently observed across all other models utilizing Prompt 1, apart from Qwen2.5. For example, Qwen2 achieves substantial increases of 16.37% points on GSM8K and 7.39% points on MATH-500 compared to the CoT baseline. These results demonstrate that our method effectively enhances performance not only with normal prompts but also with model-specific ones.

3.3. Ideal Experiment: Perfect Sparse Reward Model

To further evaluate the effectiveness of exploration within the latent space, we conduct experiments using a Perfect Sparse Reward Model (PSRM). In this experimental setup, a reward value of 0 is assigned exclusively when the generated final answer exactly matches the ground truth. In all other cases, a reward of −1 is given. The results of these experiments are presented in Table 1. Despite the limited feedback provided by this sparse reward signal—particularly in contrast to the self-reward model—the PSRM demonstrates notable performance gains. Specifically, it achieves an average improvement of 19.12% score over the CoT method and surpasses the self-reward version by an average of 12.57% score. These results highlight the potential of LATENTSEEK, even when guided by sparse reward.

Notably, when Qwen2.5-1.5B-Instruct is employed as the backbone, the performance under the PSRM often matches or exceeds that of CoT using the larger Qwen2.5-7B-Instruct. For example, on GSM8K (Prompt 2), the former achieves a score of 67.48% compared to 66.41% for the latter. Similarly, on MATH-500 (Prompt 1), the smaller model reaches 75.80%, surpassing the larger model's score of 72.80%. These findings suggest that **smaller models have acquired substantial knowledge but may lack effective mechanisms to elicit them**. Our proposed approach offers a mechanism for activating implicit knowledge, thereby enhancing the model's ability.

3.4. Test-Time Scaling: scaling up the iteration 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, the ideal reward model yields a consistently improving trend and outperforms the self-reward method across all model backbones, suggesting that **test-time scaling can be achieved without necessitating a dense reward function in our setting**. Conversely, the self-reward method exhibits rapid initial gains followed by a plateau, attributable to the limitations in the reward model's accuracy. Notably, with an increased number of iterations, performance eventually surpasses this plateau and continues to improve. These findings indicate that, given an appropriate reward model, **searching through the latent space offers a promising new direction for test-time scaling**.

Extreme Scaling with Perfect Sparse Reward Model   Following the methodology proposed by Liu et al. (2025), we conduct extreme scaling experiments using the Qwen2.5-1.5B-Instruct model (Team, 2024a). In contrast to Liu et al. (2025), which leverages Process Reward Models (PRM), our approach is specifically tailored to align with Outcome Reward Models (ORM). Given the current limitations in ORM design (see Appendix F) and the inherent challenges of self-rewarding in small-scale models, we investigate the effectiveness of this novel test-time scaling strategy through the use of the PSRM. As shown in Table 3, Qwen2.5-1.5B-Instruct achieves a 14% point performance gain over GPT-4o on the AIME2024 benchmark. Furthermore, on the MATH-500 dataset, it attains the highest overall accuracy among all evaluated models, with a marginal 2.7% point gap behind o1-preview. Remarkably, the model requires only an average of 61.8 iterations on MATH-500 to reach the performance, outperforming a 256 compute consumption of explicit space scaling methods in terms of efficiency. This implies that, under an appropriate reward model, *the latent space represents a more efficient option for test-time scaling compared to the explicit space*.

3.5. Algorithmic Statistics

Table 4 reports two key statistical features: (1) the ratio of final answer tokens to the original CoT tokens, and (2) the average number of update iterations across the entire dataset. In nearly all cases, excluding Qwen2.5-1.5B-Instruct on GSM8K with prompt 2, the token ratio does not exceed 1.1. This suggests that, unlike long CoT approaches (Xu et al., 2025b), our method does not rely on generating extended outputs to achieve strong performance, thus avoiding overly verbose generation. Moreover, concerning the number of update iterations, which constitutes the core com-

Table 4: Algorithmic Statistics: (1) The ratio of answer length to CoT length. (2) Average update iterations.

Model	Qwen2 7B	Qwen2.5 1.5B	Qwen2.5 7B	Qwen2.5 14B	LLaMA3.1 8B	Mistral 7B	Avg
GSM8K: Prompt 1							
|Answer| / |CoT|	0.97	0.99	1.00	1.00	0.94	0.99	0.98
Avg.# Iter	0.94	0.75	0.24	0.05	1.66	0.08	0.62
GSM8K: Prompt 2							
|Answer| / |CoT|	1.06	3.80	1.08	1.00	1.02	1.00	1.49
Avg.# Iter	0.55	4.59	1.52	0.14	0.58	0.24	1.27
MATH-500: Prompt 1							
|Answer| / |CoT|	0.92	0.97	0.99	0.99	0.92	0.99	0.96
Avg.# Iter	2.09	1.93	1.10	0.62	2.35	0.25	1.39
MATH-500: Prompt 2							
|Answer| / |CoT|	1.01	0.94	0.98	0.97	0.90	0.98	0.96
Avg.# Iter	0.72	2.19	0.98	0.49	1.65	0.31	1.06

putational component of the algorithm, the average iteration counts for GSM8K and MATH-500 across various model backbones and prompts are 0.86 and 1.23, respectively. Both values are below 2, indicating that the search process in the latent space typically converges in fewer than two iterations for average-level questions. This highlights the computational efficiency and rapid convergence behavior of our method.

3.6. Qualitative Analysis

To gain deeper insight into model-generated reasoning, 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. The frequent appearance of nonsensical or obscure tokens—such as "thecy," "theella," and "Sid"—indicates that the model's reasoning mechanisms often diverge from human cognitive strategies, even when producing superficially coherent outputs. A representative example, detailed in Table 5, further illustrates this divergence: despite generating linguistically anomalous expressions such as "total downloads of downloads," and "down step two andLet," the model still arrives at the correct answer. These findings suggest that **the optimal reasoning trajectories for language models are fundamentally different from those of humans and may be more effectively navigated within the model's latent space**. For further discussion, see Appendix D.

Table 5: Case Studies: the questions, the answers, the CoT generations, and the LATENTSEEK generations are listed. The incoherence examples are marked with blue color. See Appendix D for more cases.

```
QUESTION: Two trains leave San Rafael at the same time [...] What's the distance covered by each train
in the two days?
ANSWER: 230
CoT: To solve this problem, we need to break it down into steps. [...] So, the distance covered by each
train in the two days is 197.5 miles. \boxed{197.5}
LatentSeek: 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}
```

```
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?
ANSWER: 366
CoT: {"thought process": "To find the total number [...] to get the total number of downloads.", "final
answer": 213}
LATENTSEEK: {"thought process": "First solve the total downloads of downloads, the three months, [...]
total downloads = 60 + 180 + 126 = 366.", "final answer": 366}
```

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; Li et al., 2024) 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 CoT methods (Hao et al., 2024; Shen et al., 2025; Cheng and Van Durme, 2024; Deng et al., 2024) 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), which commonly employs algorithms such as Proximal Policy Optimization (PPO) (Schulman et al., 2017). Recent advancements include Direct Preference Optimization (Rafailov et al., 2023), Statistical Rejection Sampling (Liu et al., 2023b), and GRPO (Guo et al., 2025), which specifically address reasoning tasks. Innovations in reward modeling have explored human preference-based architectures (Schulman et al., 2017; Hazra et al., 2024; Yuan et al., 2022), automated language model-driven design (Kwon et al., 2023), and multi-agent verification frameworks (Lifshitz et al., 2025). Chen et al. (2024b) employs a variational method to improve the fitting of latent trajectories by updating model parameters. While these methods focus on modifying model parameters during training, our approach optimizes latent

representations during testing, without altering the model parameters.

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 like self-consistency (Wang et al., 2022), recitation-augmented generation (Sun et al., 2023), and test-time alignment (Zhang et al., 2025) 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 and Prefix Tuning adapt 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 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.

Acknowledgement

References

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 1877–1901, 2020.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *CoRR*, 2022.

OpenAI. Gpt-4 technical report. *CoRR*, 2023.

Andrew Zhao, Yiran Wu, Yang Yue, Tong Wu, Quentin Xu, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. Absolute zero: Reinforced self-play reasoning with zero data. *arXiv preprint arXiv:2505.03335*, 2025.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems (NeurIPS)*, 35: 24824–24837, 2022.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:27730–27744, 2022.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *CoRR*, 2022.

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

Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. In *International Conference on Machine Learning (ICML)*, 2020.

Moritz Hardt and Yu Sun. Test-time training on nearest neighbors for large language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. An empirical study of catastrophic forgetting in large language models during continual fine-tuning, 2025. URL https://arxiv.org/abs/2308.08747.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model?, 2025. URL https://arxiv.org/abs/2504.13837.

Pranjal Aggarwal and Sean Welleck. L1: Controlling how long a reasoning model thinks with reinforcement learning, 2025. URL https://arxiv.org/abs/2503.04697.

Tong Wu, Junzhe Shen, Zixia Jia, Yuxuan Wang, and Zilong Zheng. Tokenswift: Lossless acceleration of ultra long sequence generation. *Forty-Second International Conference on Machine Learning*, 2025.

Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric P Xing, and Zhiting Hu. Rlprompt: Optimizing discrete text prompts with reinforcement learning. In *Annual Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2022.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. *CoRR*, 2024.

Deqian Kong, Minglu Zhao, Dehong Xu, Bo Pang, Shu Wang, Edouardo Honig, Zhangzhang Si, Chuan Li, Jianwen Xie, Sirui Xie, et al. Scalable language models with posterior inference of latent thought vectors. *arXiv preprint arXiv:2502.01567*, 2025.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

Fangzhi Xu, Hang Yan, Chang Ma, Haiteng Zhao, Qiushi Sun, Kanzhi Cheng, Junxian He, Jun Liu, and Zhiyong Wu. Genius: A generalizable and purely unsupervised self-training framework for advanced reasoning. *CoRR*, 2025a.

Runze Liu, Junqi Gao, Jian Zhao, Kaiyan Zhang, Xiu Li, Biqing Qi, Wanli Ouyang, and Bowen Zhou. Can 1b llm surpass 405b llm? rethinking compute-optimal test-time scaling. *CoRR*, 2025.

Edward Yeo, Yuxuan Tong, Morry Niu, Graham Neubig, and Xiang Yue. Demystifying long chain-of-thought reasoning in llms. *CoRR*, 2025.

Haotian Xu, Xing Wu, Weinong Wang, Zhongzhi Li, Da Zheng, Boyuan Chen, Yi Hu, Shijia Kang, Jiaming Ji, Yingying Zhang, Zhijiang Guo, Yaodong Yang, Muhan Zhang, and Debing Zhang. Redstar: Does scaling long-cot data unlock better slow-reasoning systems?, 2025b. URL https://arxiv.org/abs/2501.11284.

Shalev Lifshitz, Sheila A McIlraith, and Yilun Du. Multi-agent verification: Scaling test-time compute with multiple verifiers. *CoRR*, 2025.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zhihao Fan. Qwen2 technical report. *CoRR*, 2024.

Qwen Team. Qwen2.5: A party of foundation models, September 2024a. URL https://qwenlm.github.io/blog/qwen2.5/.

Llama Team. The llama 3 herd of models, 2024b. URL https://arxiv.org/abs/2407.21783.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023. URL https://arxiv.org/abs/2310.06825.

Yuntian Deng, Yejin Choi, and Stuart Shieber. From explicit cot to implicit cot: Learning to internalize cot step by step. *CoRR*, 2024.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *CoRR*, 2021.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tulu 3: Pushing frontiers in open language model post-training, 2025. URL https://arxiv.org/abs/2411.15124.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. Self-rewarding language models. *CoRR*, 2025.

Archiki Prasad, Weizhe Yuan, Richard Yuanzhe Pang, Jing Xu, Maryam Fazel-Zarandi, Mohit Bansal, Sainbayar Sukhbaatar, Jason Weston, and Jane Yu. Self-consistency preference optimization, 2024. URL https://arxiv.org/abs/2411.04109.

Hao Liu, Carmelo Sferrazza, and Pieter Abbeel. Chain of hindsight aligns language models with feedback. In *International Conference on Learning Representations (ICLR)*, 2023a.

Weihao Zeng, Yuzhen Huang, Qian Liu, Wei Liu, Keqing He, Zejun Ma, and Junxian He. Simplerl-zoo: Investigating and taming zero reinforcement learning for open base models in the wild, 2025. URL https://arxiv.org/abs/2503.18892.

Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. Self-play fine-tuning converts weak language models to strong language models, 2024a.

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

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations (ICLR)*, 2025.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Olivier Bousquet, Quoc Le, and Ed Chi. Least-to-most prompting enables complex reasoning in large language models. In *International Conference on Learning Representations (ICLR)*, 2022.

Jiaqi Li, Xiaobo Wang, Wentao Ding, Zihao Wang, Yipeng Kang, Zixia Jia, and Zilong Zheng. Ram: Towards an ever-improving memory system by learning from communications. *arXiv preprint arXiv: 2404.12045*, 2024.

Kou Misaki, Yuichi Inoue, Yuki Imajuku, So Kuroki, Taishi Nakamura, and Takuya Akiba. Wider or deeper? scaling llm inference-time compute with adaptive branching tree search. *CoRR*, 2025.

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

Jeffrey Cheng and Benjamin Van Durme. Compressed chain of thought: Efficient reasoning through dense representations. *CoRR*, 2024.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Michele Catasta, Johan Legrand, Jelena Luketina, Andrew Lampinen, Aja Brownsmith, Zoya Bylinskii, Victoria Ellison, et al. Solving math word problems with process-based and outcome-based feedback. *CoRR*, 2022.

Jiayuan Huang, Jierui Kwon, Kevin Cohen, and Nanyun Peng. Language models as inductive reasoners. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2022.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, 2017.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems (NeurIPS)*, 36, 2023.

Tianle Liu, Tianyi Zhou, Xiaofei Lin, Percy Liang, and Tao Xiao. Statistical rejection sampling improves preference optimization. *International Conference on Learning Representations (ICLR)*, 2023b.

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

Rishi Hazra, Alkis Sygkounas, Andreas Persson, Amy Loutfi, and Pedro Zuidberg Dos Martires. Revolve: Reward evolution with large language models using human feedback. In *International Conference on Learning Representations (ICLR)*, 2024.

Luyao Yuan, Xiaofeng Gao, Zilong Zheng, Mark Edmonds, Ying Nian Wu, Federico Rossano, Hongjing Lu, Yixin Zhu, and Song-Chun Zhu. In situ bidirectional human-robot value alignment. *Science Robotics*, 7(68):eabm4183, 2022. doi: 10.1126/scirobotics.abm4183. URL https://www.science.org/doi/abs/10.1126/scirobotics.abm4183.

Minae Kwon, Sang Michael Xie, Kalesha Bullard, and Dorsa Sadigh. Reward design with language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Haolin Chen, Yihao Feng, Zuxin Liu, Weiran Yao, Akshara Prabhakar, Shelby Heinecke, Ricky Ho, Phil Mui, Silvio Savarese, Caiming Xiong, and Huan Wang. Language models are hidden reasoners: Unlocking latent reasoning capabilities via self-rewarding, 2024b. URL https://arxiv.org/abs/2411.04282.

Nitish Shirish Keskar, Bryan McCann, Lav R Varshney, Caiming Xiong, and Richard Socher. Ctrl: A conditional transformer language model for controllable generation. *CoRR*, 2019.

Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. Plug and play language models: A simple approach to controlled text generation. In *International Conference on Learning Representations (ICLR)*, 2019.

Yao Qin, Asli Celikyilmaz, Wenjing Li, Jung H Oh, Victoria Lin, Semih Agrawal, Yang Zhou, Singaram Kumar, Jing Shen, Mitesh M Khapra, et al. Pomp: Unsupervised controllable generation by optimizing prompts via reinforcement learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *CoRR*, 2022.

David Sun, Jasmine Jiang, Terry Nguyen, James Tsai, Yizhe Xia, Kyunghyun Cho, Heng Ji, Hugo Larochelle, Xiang Yuan, David Simchi-Levi, et al. Recitation-augmented language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Zhaowei Zhang, Fengshuo Bai, Qizhi Chen, Chengdong Ma, Mingzhi Wang, Haoran Sun, Zilong Zheng, and Yaodong Yang. Amulet: Realignment during test time for personalized preference adaptation of llms, 2025.

Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, et al. Learning to (learn at test time): Rnns with expressive hidden states. *CoRR*, 2024.

Shibo Hao, Yi Wang, and Bin Xiao. Reasoning with language model is planning with world model. In *Annual Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Annual Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2021.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. Gpt understands, too. *AI Open*, 5: 208–215, 2024a.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2021.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Lam Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2021.

Salil Vadhan and Qian Zhang. Lecture 31: Multiprover interactive proofs and probabilistically checkable proofs. In *Scribe Notes of CS221, Harvard School*, 2002.

S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2006. ISBN 978-0-521-42426-4. URL https://theory.cs.princeton.edu/complexity/book.pdf.

Michael Ben-Or, Shafi Goldwasser, Joe Kilian, and Avi Wigderson. Multi-prover interactive proofs: how to remove intractability assumptions. In *Annual ACM Symposium on Theory of Computing*, STOC '88, page 113–131, New York, NY, USA, 1988. Association for Computing Machinery. ISBN 0897912640. doi: 10.1145/62212.62223. URL https://doi.org/10.1145/62212.62223.

L. Babai, L. Fortnow, and C. Lund. Nondeterministic exponential time has two-prover interactive protocols. In *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, pages 16–25 vol.1, 1990. doi: 10.1109/FSCS.1990.89520.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *CoRR*, 2024.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations (ICLR)*, 2023.

Chris Yuhao Liu, Liang Zeng, Jiacai Liu, Rui Yan, Jujie He, Chaojie Wang, Shuicheng Yan, Yang Liu, and Yahui Zhou. Skywork-reward: Bag of tricks for reward modeling in llms. *CoRR*, 2024b.

Appendices

Contents

A. Theoretical Analysis

This section offers a theoretical framework for LATENTSEEK, with a main focus on justifying the expressiveness of the independence among latent variables, which may be a good start for understanding and inspecting our algorithm.

A.1. Preliminaries: Multiple Prover Interactive Proofs and NEXP

This section introduces the concept of Multi-prover Interactive Proofs (MIP) and NEXP for completeness. A comprehensive exposition of MIP is provided in Vadhan and Zhang (2002), while Arora and Barak (2006) offers a detailed introduction to theoretical computer science, encompassing concepts such as language and Turing Machine.

In the Multi-Prover Interactive Proof (MIP) model, provers may communicate with one another prior to the initiation of the proof process. Once the proof process begins, however, such communication is prohibited, and each prover interacts with the verifier in a fully private manner.

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

$$q_{11}, q_{21}, \ldots, q_{k1} = V(x)$$
$$a_{11} = P_1(x, q_{11})$$
$$a_{21} = P_2(x, q_{21})$$
$$\ldots$$
$$a_{k1} = P_k(x, q_{k1})$$
$$q_{12}, q_{22}, \ldots, q_{k2} = V(x, q_{11}, a_{11}, \ldots, q_{k1}, a_{k1})$$
$$\ldots$$
$$a_{kn} = P_k(x, q_{k1}, a_{k1}, \ldots, q_{kn})$$
$$v = V(x, q_{11}, a_{11}, \ldots, a_{kn}, q_{kn})$$

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

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

- Efficiency: The number and length of all messages exchanged is at most polynomial in the common input x.

- Completeness: $x \in L \Rightarrow \exists P_1, P_2, \ldots, P_k, \ Pr[\langle (P_1, P_2, \ldots, P_k), V \rangle_V (x) = 1] \geq \frac{2}{3}$

- Soundness: $x \notin L \Rightarrow \forall P_1, P_2, P_3, \ldots, P_k, \ Pr(\langle (P_1, P_2, \ldots, P_k), V \rangle_V (x) = 1] \leq \frac{1}{3}$

Definition A.3 (MIP (Vadhan and Zhang, 2002)). $\text{MIP} = \cup_k k - \text{MIP}$

Next, we'll introduce NEXP.

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

Definition A.5 (NP). $\text{NP} = \cup_{k \in \mathbb{N}} \text{NTIME}(n^k)$

Definition A.6 (NEXP). $\text{NEXP} = \cup_{k \in \mathbb{N}} \text{NTIME}(2^{n^k})$

A.2. Theoretical Analysis: Independent Updating

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

Our method is strongly related to a TCS complexity class, Multi Prover Interactive Proofs (Ben-Or et al., 1988) or simply MIP. We initially outline the notations for multiple prover interactions and subsequently define the associated complexity class for our latent thought framework, which we later demonstrate is equivalent to MIP and the complexity class NEXP (solved in exponential time by a non-deterministic Turing Machine). Please refer to Appendix A.1 for preliminaries.

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

$$q_{11}, q_{21}, \ldots, q_{k1} = V(x)$$
$$a_{11} = P_1(x, q_{11})$$
$$a_{21} = P_2(x, q_{21})$$
$$\ldots$$
$$a_{k1} = P_k(x, q_{k1})$$
$$q_{12}, q_{22}, \ldots, q_{k2} = V(x, q_{11}, a_{11}, \ldots, q_{k1}, a_{k1})$$
$$\ldots$$
$$a_{kn} = P_k(x, q_{k1}, a_{k1}, \ldots, q_{kn})$$
$$v = V(x, q_{11}, a_{11}, \ldots, a_{kn}, q_{kn})$$

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

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

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

- Bounded: $\forall i$, output of P_i is bounded, its output denoted as $|a_{ij}|$ satisfies that $|a_{ij}| \leq C$, where C is a constant greater than 1.

- Completeness: $x \in L \Rightarrow \exists P_1, P_2, \ldots, P_{\text{poly}(|x|)}, Pr[\langle (P_1, P_2, \ldots, P_{\text{poly}(|x|)}), V \rangle_V (x) = 1] \geq \frac{2}{3}$

- Soundness: $x \notin L \Rightarrow \forall P_1, P_2, \ldots, P_{\text{poly}(|x|)}, Pr[\langle (P_1, P_2, \ldots, P_{\text{poly}(|x|)}), V \rangle_V (x) = 1] \leq \frac{1}{3}$

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

Theorem A.10. *MIP-Bounded = MIP*

Theorem A.10 establishes the efficacy of our framework. By building on the classical result of Babai et al. (1990), we derive Corollary A.11 that clearly illustrates the expressive power of our approach. Refer to Appendix A.3 for detailed proofs.

Corollary A.11. *NP $\subset$ NEXP = MIP-Bounded*

A.3. Proofs of Theorem A.10 and Corollary A.11

Theorem A.12. *MIP-Bounded = MIP*

Proof. By the classical results of MIP = 2-MIP = NEXP (Babai et al., 1990), it's sufficient to consider only two provers in the MIP class.

We will first prove that MIP $\subseteq$ MIP-Bounded.

For all $L \in$ MIP, $\forall x$, in the j-th turn, the proofs offered by the two provers are denoted as a_{1j}, a_{2j}, the verifier as V. According to the definition of MIP, $\exists$ polynomial function $\rho(\cdot)$ such that $|a_{ij}| \leq \rho(|x|)$), we set $\text{poly}(\cdot) = \frac{2\rho(\cdot)}{C}$, which is also polynomial. We design the verifier $V' = V$.

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

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

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

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

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

Therefore, $L \in$ MIP-Bounded, and thus MIP $\subseteq$ MIP-Bounded.

Next, we are going to prove MIP-Bounded $\subseteq$ MIP.

$\forall L \in$ MIP-Bounded, $\forall x$, in the j-th turn, the proofs offer by the bounded provers are denoted as: $a'_{1j}, a'_{2j}, a'_{3j}, \ldots, a'_{\text{poly}(|x|),j}$, the verifier as V'. We design V as follows: for each concat string $s = (x, q_{11}, a_{11}, \ldots,)$, V first truncates the first prover's answer to length of $\text{LIM}_1 := C \cdot \lfloor \frac{\text{poly}(|x|)}{2} \rfloor$ 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 resulted string is denoted as $s' := (x, q_{11}, a_{11;0:\text{LIM}_1}, q_{21}, a_{21;0:\text{LIM}_2} \ldots)$, and output $V'(s')$.

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

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

19

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

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

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

$$Pr(\langle (P_1, P_2), V \rangle_V (x) = 1) = Pr[\langle (P'_1, P'_2, \ldots, P'_{\text{poly}(|x|)}), V \rangle_V (x) = 1]$$
$$= Pr[\langle (P'_1, P'_2, \ldots, P'_{\text{poly}(|x|)})', V' \rangle_{V'} (x) = 1]$$

As a result,

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

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

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

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

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

Theorem A.13 ((Babai et al., 1990)). *MIP = 2-MIP = NEXP*

Remark A.14. NEXP is the complexity class that a non-deterministic Turing Machine can solve in exponential time, which means that NP $\subset$ NEXP.

Based on this theorem, we can easily derive Corollary A.11.

Corollary A.15. *NP $\subset$ NEXP = MIP-Bounded*

B. Methods of TTIA

We list the formulations of two classical TTA methods:

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

$$x^* = \arg\max_x \pi(x|c) \tag{11}$$

- Best-of-N (BoN): Given N i.i.d sequences $x_{(1)}, x_{(2)}, \ldots, x_{(N)} \sim \pi(\cdot|c)$,

$$x^* = \arg\max_{x \in \{x_{(1)}, x_{(2)}, \ldots, x_{(N)}\}} R(x, c) \tag{12}$$

C. Experimental Details

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

C.1. Prompt Designation

The corresponding prompts are shown in the following tables.

Prompt. The prompt of CoT is as follows.

- Prompt 1: Table 7 and Table 8.
- Prompt 2: Table 6

Reward Prompt. The prompts are listed in Tables 15 to 18.

C.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`)
- Mistral-7B-Instruct-v0.3 (Jiang et al., 2023): A high-performance 7B parameter model from Mistral, fine-tuned for instruction-based tasks. (accessible at `https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.3`)

C.3. Baselines

We describe all baselines in experiments as follows:

Table 6: Prompt 1 for GSM8k (Cobbe et al., 2021), MATH-500 (Hendrycks et al., 2021) and AIME2024.

```
{''role'': ''system'', ''content'': ''Please reason step by step, and put your
final answer within \boxed\{\}.'' },
\{''role'': ''user'', ''content'': ''{q}''\}
```

- 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 Appendix C.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).

- Best-of-N (BoN): This method involves generating N independent samples or candidates from a given model, evaluating each based on a predefined objective function, and selecting the candidate that maximizes the desired reward. In our experiment, since the average update iterations of LATENTSEEK is less than 2, we select $N = 3$, which is larger, for BoN. The reward function for BoN remains the same as that of LATENTSEEK, which is performed under self-reward mechanism.

- Self-Rewarding (Yuan et al., 2025): This leverages intrinsic feedback mechanisms to iteratively enhance model performance without reliance on external reward signals.

- ScPO (Prasad et al., 2024): A framework for self-consistency preference optimization, rigorously formalizing the alignment of decision-making processes with logically coherent and preference-driven outcomes in complex systems.

- CoH (Liu et al., 2023a): A framework which systematically aligns language models with human feedback through a structured, iterative process, enhancing their performance in complex reasoning tasks.

- Genius (Xu et al., 2025a): A purely unsupervised self-training framework designed to enhance advanced reasoning capabilities in artificial intelligence systems, offering generalizable performance across diverse tasks without reliance on labeled data.

- SimpleRL-Zoo (Zeng et al., 2025): A reinforcement learning framework designed to enhance sample efficiency and performance stability in complex decision-making environments through simplified algorithmic structures and adaptive exploration strategies.

- GRPO (Shao et al., 2024): Group Relative Policy Optimization (GRPO), introduced in the DeepSeek-Math framework, is a novel reinforcement learning algorithm that enhances mathematical reasoning in large language models by optimizing policy updates through group-based reward comparisons, significantly reducing memory consumption compared to traditional Proximal Policy Optimization (PPO). In our experiment, we adopt the result from https://www.perplexity.ai/hub/blog/rl-training-for-math-reasoning?utm_source=chatgpt.com.

- SPIN (Chen et al., 2024a): A self-play fine-tuning methodology that significantly enhances the performance of weaker language models, transforming them into robust and highly capable systems competitive with stronger counterparts.

- iCoT (Deng et al., 2024): a transition from explicit Chain-of-Thought (CoT) reasoning to implicit CoT internalization, proposing a step-by-step learning framework to enhance logical reasoning capabilities in artificial intelligence systems.

- Supervised Fine-Tuning (SFT) was conducted using the LLaMA-Factory framework (Zheng et al., 2024). All models were trained with a learning rate of 1×10^{-5}, employing a cosine learning rate scheduler, a warmup ratio of 0.1, and the bfloat16 (bf16) data type.

Table 7: Prompt 2 for GSM8k (Cobbe et al., 2021).

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

Table 8: Prompt 2 for MATH-500 (Hendrycks et al., 2021) and AIME2024.

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

C.4. GSM8K

We provide details for GSM8K experiments as follows:

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.

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 9 and Table 10, 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 15 to 18.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.

Scaling. We offer more scaling figures in Figure 3.

C.5. MATH-500

We provide details for MATH-500 experiments as follows:

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 models. Sourced
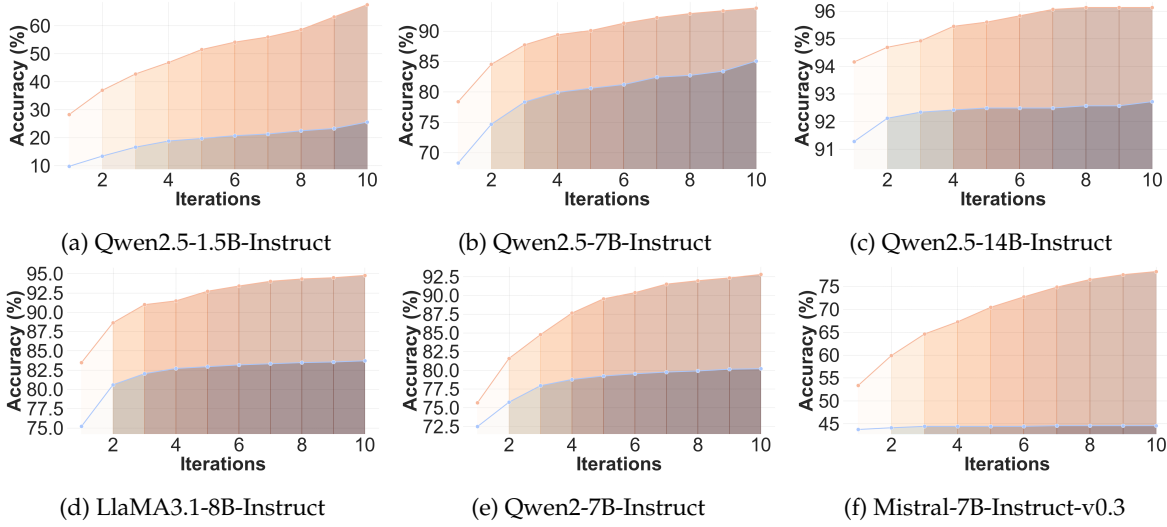
(a) Qwen2.5-1.5B-Instruct   (b) Qwen2.5-7B-Instruct   (c) Qwen2.5-14B-Instruct

(d) LlaMA3.1-8B-Instruct   (e) Qwen2-7B-Instruct   (f) Mistral-7B-Instruct-v0.3

Figure 3: GSM8k(Cobbe et al., 2021) Prompt 2 Accuracy changes with respect to the increasing number of iterations. Orange: Perfect Reward Model. Blue: Self Reward Model.

Table 9: LATENTSEEK (Self) Hyperparameters on GSM8K. lr: learning rate. ρ: fraction ratio

methods	model	max len	prompt idx	#GPU	lr	optimizer	ρ	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	10	Adam	0.2	bf16	10
LATENTSEEK (Self)	Qwen2.5-1.5B-Instruct	1024	2	1 3090	0.03	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
LATENTSEEK (Self)	Mistral-7B-Instruct-v0.3	1024	1	1 L40	0.05	Adam	0.2	bf16	10
LATENTSEEK (Self)	Mistral-7B-Instruct-v0.3	1024	2	1 L40	0.05	Adam	0.2	bf16	10

Table 10: LATENTSEEK (PSRM) Hyperparameters on GSM8K. lr: learning rate. ρ: fraction ratio

methods	model	max len	prompt idx	#GPU	lr	optimizer	ρ	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
LATENTSEEK (PSRM)	Mistral-7B-Instruct-v0.3	1024	1	1 A100	0.05	Adam	0.2	bf16	10
LATENTSEEK (PSRM)	Mistral-7B-Instruct-v0.3	1024	2	1 A100	0.05	Adam	0.2	bf16	10

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.

Table 11: LATENTSEEK (Self) Hyperparameters on MATH-500. lr: learning rate. ρ: fraction ratio

methods	model	max len	prompt idx	#GPU	lr	optimizer	ρ	dtype	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 A100	0.03	Adam	0.1	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
LATENTSEEK (Self)	Mistral-7B-Instruct-v0.3	1024	1	1 3090	0.03	Adam	0.1	bf16	10
LATENTSEEK (Self)	Mistral-7B-Instruct-v0.3	1024	2	1 3090	0.03	Adam	0.1	bf16	10

Table 12: LATENTSEEK (PSRM) Hyperparameters on MATH-500. lr: learning rate. ρ: fraction ratio

methods	model	max len	prompt idx	#GPU	lr	optimizer	ρ	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 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
LATENTSEEK (PSRM)	Mistral-7B-Instruct-v0.3	1024	1	1 3090	0.03	Adam	0.1	bf16	10
LATENTSEEK (PSRM)	Mistral-7B-Instruct-v0.3	1024	2	1 3090	0.03	Adam	0.1	bf16	10

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 11 and Table 12, 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 reward following the methodology of DeepSeek-R1 is incorporated, with weight of 2 assigned to format-based criteria.

C.6. AIME2024

We provide details for AIME2024 experiments as follows:

Dataset. The American Invitational Mathematics Examination (AIME) is a prestigious competition designed to challenge high-achieving high school students with complex mathematical problems, requiring advanced problem-solving and reasoning skills. The AIME2024 dataset, as introduced in this context, serves as a valuable resource for evaluating the capabilities of language models in tackling such sophisticated mathematical tasks. Sourced from the Huggingface repository Maxwell-Jia/AIME_2024 (accessible at https://huggingface.co/datasets/Maxwell-Jia/AIME_2024), the AIME2024 dataset comprises 30 meticulously curated problems. Although modest in quantity, each problem is deliberately designed to reflect the style, rigor, and difficulty of the AIME, thereby providing a robust benchmark for assessing advanced mathematical reasoning in computational models.

Table 13: LATENTSEEK (Self) Hyperparameters on AIME2024. lr: learning rate. ρ: fraction ratio

methods	model	max len	prompt idx	#GPU	lr	optimizer	ρ	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
LATENTSEEK (Self)	Mistral-7B-Instruct-v0.3	1024	1	1 3090	0.03	Adam	0.1	bf16	10
LATENTSEEK (Self)	Mistral-7B-Instruct-v0.3	1024	2	1 3090	0.05	Adam	0.2	bf16	10

Table 14: LATENTSEEK (PSRM) Hyperparameters on AIME2024. lr: learning rate. ρ: fraction ratio

methods	model	max len	prompt idx	#GPU	lr	optimizer	ρ	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
LATENTSEEK (PSRM)	Mistral-7B-Instruct-v0.3	1024	1	1 3090	0.03	Adam	0.1	bf16	10
LATENTSEEK (PSRM)	Mistral-7B-Instruct-v0.3	1024	2	1 3090	0.03	Adam	0.1	bf16	10

Table 15: Prompt for answer correctness check

```
f"{math_prefix}"
"INSTRUCTIONS:"
"Your task is to determine whether the provided answer is correct."
"Think through the verification process carefully and logically."
"IMPORTANT RULES:"
"1. Do NOT analyze the steps or methods used to arrive at the answer."
"2. Only evaluate the final answer's correctness."
"3. Your response must strictly follow the required format:"
f"- If the answer is correct, respond with: '{VERA_ANSWER_SYMBOL}True'."
f"- If the answer is incorrect, respond with: '{VERA_ANSWER_SYMBOL}False'."
```

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 13 and Table 14, 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.

Table 16: Prompt for calculation check

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

Table 17: Prompt for understanding check

```
f"{math_prefix}"
"INSTRUCTIONS:"
"1. PROBLEM INTERPRETATION:"
" - Assess if the proposed solution clearly understands the problem
statement."
" - Ensure that the proposed solution addresses all relevant aspects of
the problem, without ignoring any key detail."
" - Flag if the solution misinterprets or overlooks the problem's core
requirements or scope."
"2. ALIGNMENT WITH THE TASK:"
" - Verify that the solution responds to the specific question or task
outlined in the problem statement."
" - Ensure that the solution does not deviate from the problem's context
or provides an unrelated answer."
" - Check if any critical parts of the problem have been misinterpreted
or neglected."
"3. TERMINATION PROTOCOL:"
" - If the solution clearly misinterprets or fails to address the
problem correctly, stop and respond in the exact format:"
f" - '{VERA_ANSWER_SYMBOL}False'"
" - If the solution accurately captures the problem statement and aligns
with the required solution, respond in the exact format:"
f" - '{VERA_ANSWER_SYMBOL}True'"
"EXAMPLES:"
"[Case 1] Problem: A shop is selling a drink at 1.5 times the original
price. If the original price is $10, what is the new price?"
" Solution: The new price is 1.15 * $10 = $11.50."
" Assessment: The solution misinterprets the problem by calculating 1.15
times the original price instead of 1.5 times."
f" Result: '{VERA_ANSWER_SYMBOL}False'"
"[Case 2] Problem: The second cup of coffee is half price. If the first
cup costs $5, how much is the second cup?"
" Solution: The second cup costs $5 * 0.5 = $2.50."
" Assessment: The solution correctly interprets the price as half the
original price for the second cup."
f" Result: '{VERA_ANSWER_SYMBOL}True'"
"[Case 3] Problem: A pizza has a radius of 8 inches. What is the area of
the pizza?"
" Solution: The area is $\pi r^2$, where r = 4 inches. The area is $16\pi$ square
inches."
" Assessment: The solution misinterprets the formula for the area of a
circle by using the radius incorrectly."
f" Result: '{VERA_ANSWER_SYMBOL}False'"
"[Case 4] Problem: A train is moving at 60 km/h towards the east. What
is its velocity after 2 hours?"
" Solution: The velocity is 120 km/h west."
" 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."
```

Table 18: Prompt for answer completeness check

```
f"{math_prefix}"
"INSTRUCTIONS:"
"Your task is to verify whether the solution provides a complete and
final answer."
"Follow these rules carefully:"
"1. Check if the solution reaches a clear and definitive final answer."
"2. The answer must not be left incomplete, such as:"
" - Ending with an unresolved expression or formula instead of a
computed result."
" - Missing a conclusion or final statement explicitly stating the final
answer."
"3. If the solution is incomplete or lacks a final answer, immediately
stop checking further and respond in the exact format:"
f" - '{VERA_ANSWER_SYMBOL}False'"
"4. If the solution is complete and provides a final, explicit answer,
respond in the exact format:"
f" - '{VERA_ANSWER_SYMBOL}True'"
"Examples:"
"Example 1:"
"final answer: 8."
f"Your response: '{VERA_ANSWER_SYMBOL}True' (The solution provides a
final, definitive answer of 8.)"
"Example 2:"
"final answer: The area of the circle is $\pi r^2$, where r = 4."
f"Your response: '{VERA_ANSWER_SYMBOL}False' (The answer ends with an
unresolved formula, not a computed result.)"
"Example 3:"
"final answer: This question does not have an answer or I cannot solve
this problem."
f"Your response: '{VERA_ANSWER_SYMBOL}False' (The solution lacks a
clear, final answer.)"
```

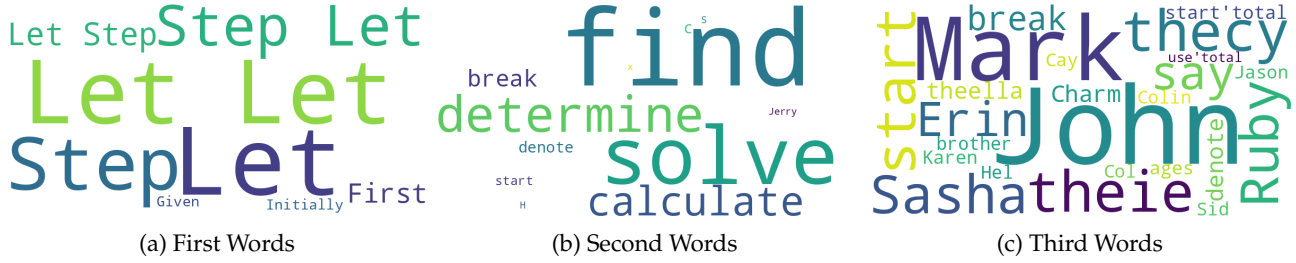| (a) First Words | (b) Second Words | (c) Third Words |

Figure 4: 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.

Table 19: 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}
```

D. 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 4. The appearance of nonsensical or obscure tokens—such as "thecy," "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 19 to 25 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 transfromed 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 20: Case 2

QUESTION: Jen enters a lottery by picking 4 distinct numbers from
$S = \{1,2,3,\cdots,9,10\}$. 4 numbers are randomly chosen from S. She wins a
prize if at least two of her numbers match the chosen numbers, and wins
the grand prize if all four match. Find the conditional probability of
winning the grand prize given that she won any prize.
TRUE ANSWER: 116
ORIGINAL MODEL: {"thought process": "To solve this problem [...]
Finally, we can add m and n to get the final answer.", "final answer":
5}
OURS: {"thought process": "First find this problem, let need calculate
to calculate the probability number of possible to can choose a
prize and the number of possible to can choose the grand prize. [...]
Therefore, the answer is $1 + 115 = 116$.", "final answer": 116}

Table 21: Case 3

QUESTION: Two trains leave San Rafael at the same time. They begin
traveling westward, both traveling for 80 miles. The next day, they
travel northwards, covering 150 miles. What's the distance covered by
each train in the two days?
TRUE ANSWER: 230
ORIGINAL MODEL: To solve this problem, we need to break it down into
steps. [...] So, the distance covered by each train in the two days is
197.5 miles. \boxed{197.5}
OURS: 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}

Table 22: Case 4

QUESTION: Carlos is planting a lemon tree. The tree will cost $90 to
plant. Each year it will grow 7 lemons, which he can sell for $1.5 each.
It costs $3 a year to water and feed the tree. How many years will it
take before he starts earning money on the lemon tree?
TRUE ANSWER: 13
ORIGINAL MODEL: {"thought process": "To find out how many years [...]
We will continue this process for each subsequent year, adding the
maintenance cost and calculating the income until the total income is
greater than the total cost.", "final answer": 7}
OURS: {"thought process": "Let determine the when long years Carlos will
take before the to earn earning money from his tree tree, I first to
find the revenue revenue for planting the maintaining the tree. [...]
Therefore, it will take 13 years for Carlos to start earning money on
the lemon tree.", "final answer": 13}

Table 23: Case 5

QUESTION: Gunter is trying to count the jelly beans in a jar. He asks
his friends how many they think are in the jar. One says 80. Another
says 20 more than half the first one. A third says 25% more than the
first one. What is their average guess?
TRUE ANSWER: 80
ORIGINAL MODEL: {"thought process": "To find the average guess [...]
After finding the individual guesses, we can add them up and divide by
the number of guesses to find the average.", "final answer": 76}
OURS: {"thought process": "First find the average of, we first to
calculate calculate each number guesses of The first friend 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}

Table 24: Case 6

QUESTION: Frankie watches TV after he finishes his homework every night.
On Monday and Tuesday, he watched a 1-hour episode of his favorite show
each night. On Wednesday, he watched a few episodes of a 30-minute show.
On Thursday, he finished homework early and watched a 1-hour episode and
a 30-minute show. On Friday, he got to stay up late for the weekend, so
he watched two 1-hour episodes. If he watched 7 hours of TV in all, how
many 30-minute episodes did he watch on Wednesday?
TRUE ANSWER: 3
ORIGINAL MODEL: To determine how many 30-minute episodes Frankie watched
on Wednesday, [...] So, Frankie watched \boxed{197.5} 30-minute episodes
on Wednesday.
OURS: Let determine how many 30-minute episodes Frankie watched on
Wednesday, let will to break the total amount he spent watching TV
from night and then solve the the much time was spent on Wednesday
specifically1, we's summarize the time time Frankie watching TV each
day: [...] Thus, the number of 30-minute episodes Frankie watched on
Wednesday is \boxed{3}.

Table 25: Case 7

QUESTION: Lloyd has an egg farm. His chickens produce 252 eggs per day
and he sells them for $2 per dozen. How much does Lloyd make on eggs per
week?
TRUE ANSWER: 294
ORIGINAL MODEL: {"thought process": "To find out how much Lloyd makes
on eggs per week, [...] Finally, we multiply the number of dozens by the
price per dozen to get the total amount Lloyd makes in a week.", "final
answer": 5880}
OURS: {"thought process": "First calculate Lloyd how much Lloyd makes
per eggs per week, first need calculate to determine out the many eggs
he produces in a week. [...] Total amount per week = 147 * 2 = $294.
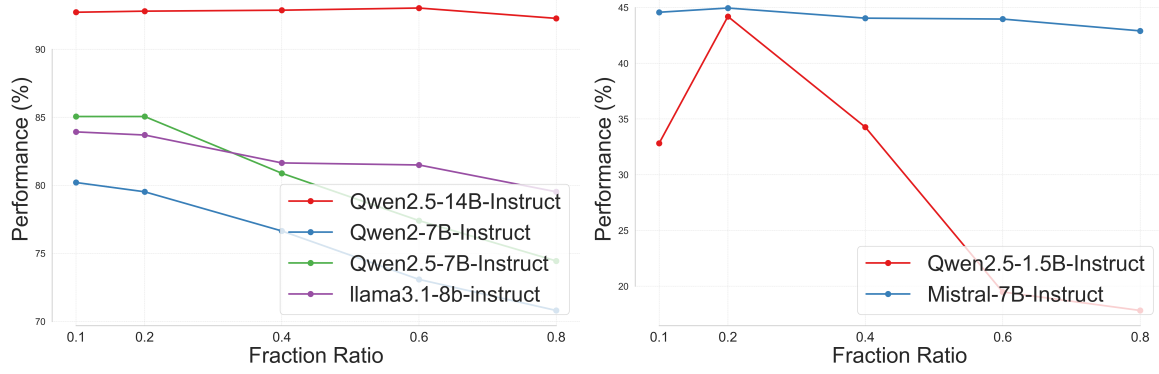Therefore, Lloyd makes $294 per week on eggs..", "final answer": 294}

Figure 5: Performance vs Fraction Ratio

E. Performance vs Fraction Ratio

Figure Figure 5 illustrates the performance trends of various instruction-tuned language models across different fraction ratios, which range from 0.1 to 0.8. Performance is reported as a percentage and plotted on the y-axis, while the fraction ratio is shown on the x-axis.

- *Qwen2.5-14B-Instruct* exhibits a relatively stable performance curve. It shows a slight increase in performance from 0.1 to 0.6, peaking around the 0.6 fraction ratio, followed by a minor decrease at 0.8. Overall, this model maintains consistently high performance above 91% across all fraction ratios.

- *Qwen2.5-7B-Instruct* demonstrates a mild upward trend from 0.1 to 0.2, followed by a steady decrease as the fraction ratio increases beyond 0.2, indicating reduced effectiveness at higher ratios.

- *Qwen2-7B-Instruct* shows a consistent downward trend across the full range of fraction ratios, suggesting that its performance deteriorates steadily with increasing input fraction.

- *LLaMA3.1-8B-Instruct* remains relatively stable at first but shows a slight decreasing trend overall, with performance gently declining from 0.2 onwards.

- *Qwen2.5-1.5B-Instruct* follows a sharp non-monotonic trend. It increases markedly from 0.1 to peak at 0.2, then drops rapidly as the fraction ratio increases further, reaching its lowest performance at 0.8.

- *Mistral-7B-Instruct* exhibits a subtle peak at 0.2, followed by a gradual and modest decrease in performance as the fraction ratio increases, maintaining relatively stable performance throughout.

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

F. Limitations of Outcome Reward Model

The reward function of using a outcome reward model is defined as follows:

$$\tilde{x} \sim \pi(\tilde{x}|x) \tag{13}$$

$$R(x, c) \sim \text{RewardModel}(\cdot|\tilde{x}, c) \tag{14}$$

The performance is described in (Liu et al., 2024b), with results presented in Table 26. Although this model achieves an average score of 60.16—representing an improvement of 1.91 points over the Chain-of-Thought (CoT) method—it remains significantly inferior to the performance attained using

Table 26: 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	7B	14B	LLaMA3.1 8B	Mistral 7B	Avg
GSM8k: Prompt 2							
CoT	65.20	15.31	66.41	91.81	69.07	41.70	58.25
LATENTSEEK (*Math*)	61.18	28.28	66.64	91.05	76.80	37.00	60.16
LATENTSEEK (*Self*)	**80.21**	**44.20**	**85.06**	**92.72**	**83.70**	**44.58**	**71.74**

a self-rewarding mechanism, which reaches 71.74. 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.

G. Computational Resources

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

H. Ethics and Societal Impact

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

I. Limitations and future works

Reward Models. One limitation of our current approach lies in the use of a self-rewarding mechanism. Ideally, employing a more principled outcome-based reward model would be preferable. However, our experiments indicate that existing outcome reward models are not yet sufficiently effective, suggesting a promising direction for future research.

Latent Optimization. We adopt basic policy gradient methods in our implementation, deliberately leaving the exploration of more advanced reinforcement learning algorithms—such as Proximal Policy Optimization (PPO)—to future work by the research community.

Large Base Model. Our experiments are conducted using 14B-parameter models, constrained by available computational resources. Scaling the approach to larger base models remains an important avenue for future investigation.