

# Self-Reflective Generation at Test Time

Jian Mu<sup>1</sup>, Qixin Zhang<sup>2</sup>, Zhiyong Wang<sup>3</sup>, Menglin Yang<sup>1</sup>, Shuang Qiu<sup>4</sup>,  
Chengwei Qin<sup>1</sup>, Zhongxiang Dai<sup>5</sup>, Yao Shu<sup>1\*</sup>

<sup>1</sup>Hong Kong University of Science and Technology (Guangzhou),

<sup>2</sup>Nanyang Technological University, <sup>3</sup>University of Edinburgh,

<sup>4</sup>City University of Hong Kong, <sup>5</sup>The Chinese University of Hong Kong, Shenzhen  
jianmu@hkust-gz.edu.cn, yaoshu@hkust-gz.edu.cn

## Abstract

Large language models (LLMs) increasingly solve complex reasoning tasks via long chain-of-thought, but their forward-only autoregressive generation process is fragile; early token errors can cascade, which creates a clear need for self-reflection mechanisms. However, existing self-reflection either performs revisions over full drafts or learns self-correction via expensive training, both fundamentally reactive and inefficient. To address this, we propose *Self-Reflective Generation at Test Time* (SRGen), a lightweight test-time framework that reflects before generating at uncertain points. During token generation, SRGen utilizes dynamic entropy thresholding to identify high-uncertainty tokens. For each identified token, it trains a specific corrective vector, which fully exploits the already generated context for a self-reflective generation to correct the token probability distribution. By retrospectively analyzing the partial output, this self-reflection enables more trustworthy decisions, thereby significantly reducing the probability of errors at highly uncertain points. Evaluated on challenging mathematical reasoning benchmarks and a diverse set of LLMs, SRGen can significantly strengthen model reasoning. Moreover, our findings position SRGen as a plug-and-play method that integrates reflection into the generation process for reliable LLM reasoning, achieving consistent gains with bounded overhead and can be combined with other training-time (e.g., RLHF) and test-time (e.g., SLOT) techniques.

## 1 Introduction

The ability to execute complex multi-step reasoning remains a central frontier in advancing large language models (LLMs). LLMs generate step-by-step reasoning traces, often called chain-of-thought

(CoT) (Wei et al., 2022). This capability has enabled substantial progress in mathematics, program synthesis, and other domains (Yao et al., 2023; Plaat et al., 2024). The fidelity of these traces often determines whether the final answer is correct (Paul et al., 2024; Hammoud et al., 2025). Thus, improving the reliability of the reasoning process is critical to realizing the full potential of LLMs.

A fundamental tension persists between the fluid, self-corrective character of human problem solving and the rigid, forward-only dynamics of standard LLM decoding. Humans iterate: they pause, re-evaluate premises, and change course. In contrast, LLMs perform autoregressive decoding (Vaswani et al., 2017): each token depends on all preceding tokens, and prior outputs cannot be revised. As a result, early errors can propagate and compound, derailing the entire trajectory (Jain et al., 2025). This brittleness in forward-only decoding is a major obstacle to reliable reasoning.

Many prior works tackle this fragility via error correction. One line pursues post hoc iterative refinement: the model critiques and revises a complete draft in subsequent passes (Madaan et al., 2023; Yuksekgonul et al., 2024), incurring substantial latency and computational cost. Another line trains models for intrinsic self-correction, for example via reinforcement learning (Bensal et al., 2025; Ma et al., 2025). This enables mid-reasoning fixes but still requires that an erroneous segment be produced before intervention. Crucially, both approaches are reactive; they address errors only after they have occurred. The challenge of proactive error prevention, namely steering the model away from a mistake before it is committed, remains a research gap.

To this end, we introduce *Self-Reflective Generation at Test Time* (SRGen), a lightweight inference-time framework for proactive error prevention. The key premise is that tokens differ in informativeness: recent work identifies “critical

\*Corresponding author

†<https://github.com/2020-qqtcg/SRGen>

Table 1: Conceptual comparison of self-reflection paradigms.

Paradigm	Timing	Cost		Mode
		Train	Infer	
RL Correction	Training	High	Low	Reactive
Post-hoc Refine	Post-gen	Zero	High	Reactive
<b>SRGen (Ours)</b>	<b>In-gen</b>	<b>Zero</b>	<b>Low</b>	<b>Proactive</b>

tokens” via high predictive entropy (Wang et al., 2025), low confidence (Fu et al., 2025), or spikes in mutual information (Qian et al., 2025). Instead of using these signals solely to adjust sampling or apply post hoc filtering, SRGen intervenes at the moment of risk: during generation it detects critical points, briefly pauses, and optimizes a small corrective vector  $\delta$  with a token-level reflection loss; this vector is injected into the hidden state before emitting the next token. The intervention is local and transient, steering the model away from early errors without additional full passes.

Table 1 positions SRGen relative to existing self-reflection approaches and highlights a new paradigm of proactive, test-time self-reflection. SRGen requires no additional training (unlike RL-based methods), avoids the latency of post hoc iterative refinement, and prevents errors before they compound. Its plug-and-play nature makes it broadly applicable to pre-trained language models and compatible with other reasoning-enhancement techniques, including SFT, RL, and distillation trained models and with some test-time methods even using similar mechanisms.

Our main contributions are as follows:

- We propose SRGen, which consists of two core components: an uncertainty-based token monitor and a self-reflective corrective vector computed online. Without any external feedback, SRGen improves reasoning by reducing decision errors at uncertain tokens during generation.
- We provide theoretical motivation for our design and conduct analyses to support and interpret the proposed method.
- Extensive experiments demonstrate that SRGen achieves larger gains over strong baselines while introducing lower latency overhead. SRGen also composes well with related methods, yielding further improvements when combined.

## 2 Problem Formulation

In autoregressive language models, the generation of a token sequence  $y = (y_0, y_1, \dots, y_t)$  is modeled by the product of conditional probabilities:

$$P(y|x_0) = \prod_{t=1}^T P(y_t|y_{<t}, x_0; \theta), \quad (1)$$

where  $x_0$  is the input prompt and  $\theta$  denotes the model parameters. Such a forward-only decoding process, unfortunately, exhibits a key vulnerability: fragility. An early error in a reasoning chain can propagate and amplify, leading to catastrophic failures in the final output (Jain et al., 2025).

Existing solutions predominantly fall into two categories. **(1) Post-hoc Iterative Refinement.** Methods such as Self-Refine (Madaan et al., 2023) employ a multi-stage pipeline where the model critiques and revises a complete draft (Shinn et al., 2023; Yu et al., 2024; Yuksekogonul et al., 2024). Although often effective, this approach incurs substantial computational overhead and latency. **(2) Training for Intrinsic Self-Correction.** This line of work embeds correction capabilities directly into the model parameters, typically via techniques like reinforcement learning (Bai et al., 2022; Hu et al., 2024; Kumar et al., 2024; Moskvoretiskii et al., 2025; Bensal et al., 2025). These methods require expensive, resource-intensive training and can only intervene after an error has already been generated.

A common thread unites these approaches: their reactive nature, as they correct errors only after they have occurred. This limitation motivates our central research question:

*Can we design a proactive error prevention mechanism that identifies and intervenes at potential error points in real-time during generation, thereby enhancing reasoning reliability within a single decoding pass and at a minimal additional cost?*

## 3 Self-Reflection Generation Process

### 3.1 Overview of SRGen

To mitigate error propagation during autoregressive generation, we introduce a novel *Self-Reflective Generation at Test Time* (SRGen) framework. SRGen embeds a lightweight monitor-reflect-optimize loop into the autoregressive decoding process. This loop enables the model to identify and correct potential errors at intermediate steps, thereby miti-

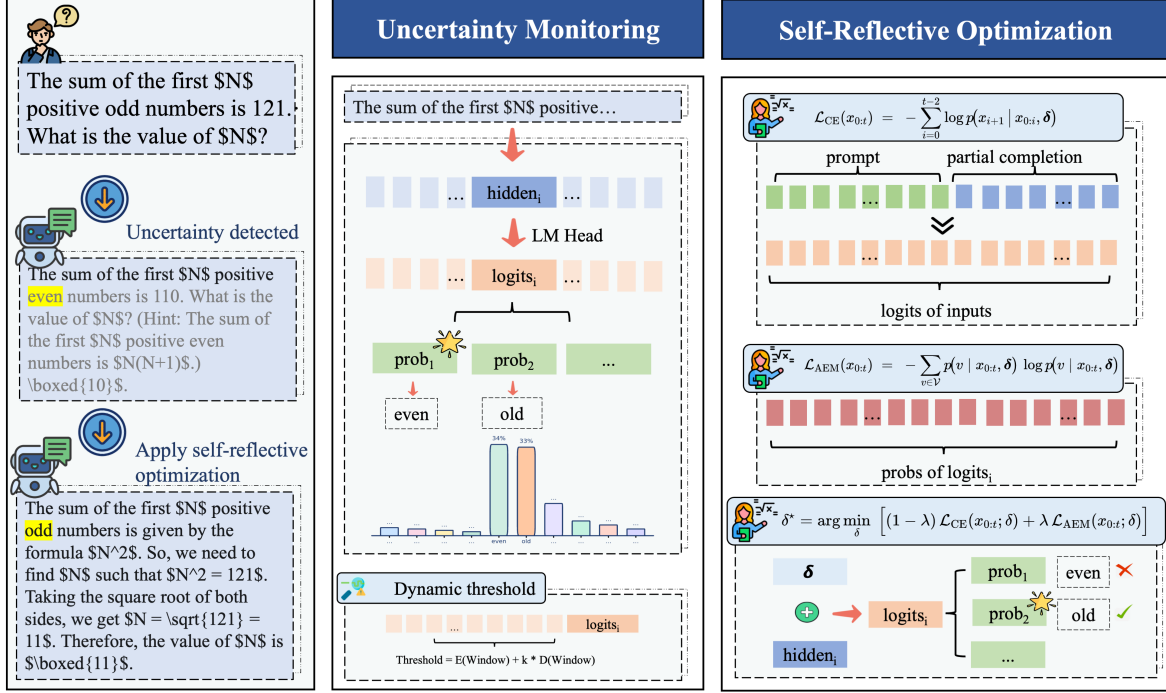


Figure 1: An overview of the *Self-Reflective Generation* (SRGen) framework. This framework consists of two main stages. (1) *Uncertainty Monitoring*. A threshold is dynamically computed from the mean and standard deviation of token entropies within a recent history window of size  $N$ . (2) *Self-Reflective Optimization*. If the current token’s entropy exceeds the threshold, a correction vector,  $\delta$ , is optimized on-the-fly using a joint loss of cross-entropy and entropy minimization. This  $\delta$  is then added to the token’s hidden state to steer the final decision towards a more reliable outcome.

gating their propagation throughout the generated sequence. As illustrated in Figure 1, the process at each generation step  $t$  consists of two stages:

### Stage 1: Dynamic Uncertainty Monitoring.

At each step, the framework assesses the model predictive uncertainty for the next token. We quantify this uncertainty using token entropy. An intervention is triggered if this entropy exceeds a dynamic threshold that adapts to the local context of the generated sequence.

### Stage 2: Self-Reflective Optimization.

If the uncertainty exceeds the threshold, the standard decoding process is paused to initiate a self-reflective optimization. This optimization computes a transient correction vector  $\delta$  by minimizing a self-reflection loss function. The correction vector  $\delta$  is then applied to the current hidden state to refine the next-token probability distribution, guiding it toward a more confident and contextually coherent output. If the threshold is not met, the model proceeds with standard decoding.

The complete workflow is formalized in Algorithm 1. By selectively applying this real-time intervention, SRGen enhances output reliability with

minimal and bounded computational overhead.

## 3.2 Stage 1: Dynamic Uncertainty Monitoring

A primary challenge in identifying critical reasoning junctures is that a fixed uncertainty threshold is suboptimal. Models with different architectures, training paradigms, or scales have distinct entropy profiles, even on the same task, as shown in Appendix F. A static threshold would therefore fail to reliably detect moments of high uncertainty across diverse contexts. To address this issue, we propose a dynamic thresholding strategy that adapts to the recent generation history of a model. Formally, at each decoding step  $t$ , before sampling the next token  $y_t$ , we first compute the predictive entropy of the next-token distribution given the prefix  $y_{<t} = (y_0, \dots, y_{t-1})$ :

$$H_t = H(p(\cdot | y_{<t})). \quad (2)$$

We maintain a sliding window  $\mathcal{H}_t$  containing the  $N$  most recent entropy values,  $\{H_{t-N}, \dots, H_{t-1}\}$ . From this history, we compute the running mean  $\mu(\mathcal{H}_t)$  and standard deviation  $\sigma(\mathcal{H}_t)$ . The self-reflection process is triggered if the current entropy

$H_t$  represents a statistically significant deviation from the recent trend. Formally, reflection is activated if:

$$H_t > \mu(\mathcal{H}_t) + k \cdot \sigma(\mathcal{H}_t), \quad (3)$$

where  $k$  is a sensitivity hyperparameter. This adaptive approach enables our method to distinguish between naturally high-entropy passages and anomalous uncertainty spikes that warrant intervention. We present in the Section 5.6 the tokens identified by this method.

### 3.3 Stage 2: Self-Reflective Optimization

Once a high-uncertainty juncture is identified at step  $t$ , our goal is to compute a transient correction to steer the generative process of a model.

Just as humans pause to deliberate when facing indecision, eventually settling on the most prudent answer, our intuitive goal is for the model to reduce uncertainty at the current step and make a decisive choice, rather than sampling from an ambiguous distribution. However, blindly minimizing entropy carries a risk: it can cause the output distribution to collapse onto high-frequency tokens that are statistically dominant but contextually vacuous. Human reasoning typically involves a retrospective review of established thoughts, serving as a foundation for further deduction. Consequently, any reduction in uncertainty must be predicated on strict fidelity to the already-generated context. Therefore, a self-reflective generation process should inherently address two dual objectives: sharpening the predictive distribution to reduce uncertainty while preserving the semantic coherence established by the preceding context  $y_{<t}$ .

Inspired by the work of (Hu et al., 2025), we introduce a transient correction vector  $\delta \in \mathbb{R}^d$ , where  $d$  is the dimension of the model hidden state. This vector is initialized to zero and optimized only when the uncertainty monitor is triggered. The correction is applied to the final hidden state  $h_{t-1}$  before the vocabulary projection head  $\mathcal{W}$ , yielding a modified logits vector:

$$\text{logits}'_t = \mathcal{W}(h_{t-1} + \delta). \quad (4)$$

The optimization of  $\delta$  is guided by a hybrid loss function  $\mathcal{L}_{\text{SRGen}}$  defined over the prefix  $y_{<t}$ :

$$\mathcal{L}_{\text{SRGen}}(\delta; \lambda, y_{<t}) = (1 - \lambda)\mathcal{L}_{\text{CE}}(y_{<t}; \delta) + \lambda\mathcal{L}_{\text{AEM}}(y_{<t}; \delta). \quad (5)$$

This loss comprises two components:

① **Retrospective Context Loss ( $\mathcal{L}_{\text{CE}}$ )**. This term ensures contextual fidelity by penalizing corrections  $\delta$  that disrupt the model predictions for the already-generated prefix. It is the negative log-likelihood of the prefix, where the same correction  $\delta$  is applied to all historical hidden states:

$$\mathcal{L}_{\text{CE}}(y_{<t}; \delta) = - \sum_{i=0}^{t-2} \log p(y_{i+1}|y_{\leq i}, \delta), \quad (6)$$

where  $p(y_{i+1}|y_{\leq i}, \delta) = \text{softmax}(\mathcal{W}(h_i + \delta))_{y_{i+1}}$  and  $h_i$  is the hidden state corresponding to the prefix  $y_{\leq i}$ .

② **Anticipatory Entropy Minimization ( $\mathcal{L}_{\text{AEM}}$ )**. This term directly targets the high uncertainty at the current step  $t$ . By minimizing the entropy of the next-token predictive distribution, it encourages the model to make a more confident prediction:

$$\mathcal{L}_{\text{AEM}}(y_{<t}; \delta) = H(p(\cdot|y_{<t}, \delta)), \quad (7)$$

where the perturbed distribution is  $p(\cdot|y_{<t}, \delta) = \text{softmax}(\mathcal{W}(h_{t-1} + \delta))$ .

After optimizing  $\delta$  for a few gradient steps to find  $\delta^*$ , we use this correction to generate the token  $y_t$ . The vector is then discarded, ensuring that each intervention is localized to its specific context.

## 4 Method Analysis and Insights

This section provides a deeper analysis of the SR-Gen framework, justifying its core design principles and theoretical underpinnings.

### 4.1 Rationale for Dynamic and Selective Intervention

The design of our uncertainty monitor is based on the principle of targeted intervention, which is crucial for two reasons: **(1) Efficiency:** Limiting the intervention to a few key tokens significantly reduces computational overhead. Applying reflection at every step would be computationally prohibitive, whereas our targeted approach adds only a modest, bounded overhead by focusing resources on the most critical junctures. **(2) Quality:** Excessive self-reflection can be counterproductive. Unnecessary intervention on low-uncertainty tokens may disrupt the fluency of the generated text. Our selective strategy, guided by the dynamic threshold, ensures that intervention is not only efficient but also beneficial to the final output quality.

## 4.2 Theoretical Basis of the Hybrid Loss

Our central theoretical result establishes that the hybrid loss in SRGen is not a heuristic but emerges directly from a principled optimization problem. We formalize this by showing that our loss function is equivalent to the Lagrangian of a constrained objective that seeks to minimize uncertainty while preserving contextual fidelity. This is stated formally in Theorem 1 and a detailed proof is provided in Appendix C.1.

**Theorem 1** (Hybrid Loss as Principled Constrained Optimization). *Given a trade-off parameter  $\lambda \in (0, 1)$ , the minimizer  $\delta^*$  of the hybrid loss objective*

$$\mathcal{L}_{SRGen}(\delta; \lambda) = (1 - \lambda) \mathcal{L}_{CE}(\delta) + \lambda \mathcal{L}_{AEM}(\delta), \quad (8)$$

*is also the solution to the constrained optimization problem*

$$\min_{\delta} \mathcal{L}_{AEM}(\delta) \quad \text{s.t.} \quad \mathcal{L}_{CE}(\delta) \leq \epsilon. \quad (9)$$

*The choice of  $\lambda$  implicitly defines the constraint boundary  $\epsilon = \mathcal{L}_{CE}(\delta^*)$ , establishing a formal equivalence between tuning the loss weight and setting a fidelity tolerance.*

**Remark.** This theorem provides a strong theoretical grounding for our method. The most powerful insight is that *the SRGen objective is not an arbitrary blend of losses but a principled, tractable solution to a well-defined constrained optimization problem*. This reframes the intuitive goal of cautious generation, i.e., reducing future uncertainty ( $\mathcal{L}_{AEM}$ ) without sacrificing fidelity to the current context ( $\mathcal{L}_{CE} \leq \epsilon$ ), in the rigorous language of optimization theory.

The parameter  $\lambda$  is thus revealed to be more than a simple weighting factor; it implicitly controls the “price” of violating the contextual fidelity constraint. A small  $\lambda$  (corresponding to a large Lagrange multiplier  $\alpha$ ) enforces a strict fidelity requirement, heavily penalizing deviations. Conversely, a large  $\lambda$  prioritizes uncertainty reduction, effectively relaxing the constraint. This perspective provides a formal basis for tuning  $\lambda$ . Furthermore, this result justifies our use of a simple weighted sum for the loss function. It demonstrates that this common practical approach is, in this case, equivalent to the more complex but formally correct Lagrangian relaxation, making the objective

both theoretically sound and easily optimizable via standard gradient-based methods.

## 4.3 Computational Overhead Analysis

The computational overhead of SRGen comprises two components: a negligible monitoring stage and a more substantial, on-demand optimization stage. The monitoring stage, which computes predictive entropy at each token, incurs minimal cost because its inputs (logits and softmax distributions) are already computed during standard autoregressive generation.

Consequently, the primary overhead stems from the on-the-fly optimization of the correction vector  $\delta$ , which is triggered only at sparse, high-uncertainty junctures. This cost can be approximated as:

$$\text{Overhead} \approx N_{\text{act}} \times T \times C_{\text{bp}}, \quad (10)$$

where  $N_{\text{act}}$  is the number of reflection activations,  $T$  is the number of inner optimization steps, and  $C_{\text{bp}}$  is the cost of a single backpropagation pass. This design is inherently efficient. Unlike post-hoc refinement methods, whose costs scale linearly with the full sequence length, the overhead of our SRGen scales only with the number of critical interventions. Our experiments in Section 5.3 showing that our method incurs only a small additional overhead. This makes SRGen a practical solution for enhancing model reasoning without prohibitive computational expense.

## 5 Experiment

### 5.1 Experimental Setup

**Models.** We assess generality on open-weight models spanning scales, architectures, and post-training regimes: Qwen2.5-Math-7B (Team, 2024), DeepSeek-R1-Distill-Qwen-7B, DeepSeek-R1-Distill-Llama-8B (Guo et al., 2025), and Qwen3-32B (Yang et al., 2025). This set covers two architecture families (Qwen, Llama), sizes from 7B to 32B, and heterogeneous post-training pipelines (distillation, SFT, RL) that yield distinct entropy profiles. This diversity probes whether SRGen remains effective across decoding behaviors, tokenizers, and training regimes, rather than overfitting to any single family or size.

**Benchmarks.** We evaluate on AIME2024, AIME2025 (Art of Problem Solving, 2024), HMMT2025 (Balunović et al., 2025), and AMC (LI et al., 2024) for mathematical reasoning problems,

Table 2: Performance comparison on mathematical, general reasoning and code tasks. Best performance is bolded. Gray rows indicate SRGen.

Model	Method	Mathematical Reasoning				General & Code	
		AIME24	AIME25	HMMT25	AMC	GPQA	EvalPlus
Qwen2.5-Math-7B	CoT	14.6	6.0	1.3	34.0	31.8	45.7
	Self-Refine	15.3 (+0.7)	5.7 (-0.3)	0.0 (-1.3)	32.4 (-1.6)	31.5 (-0.3)	43.9 (-1.8)
	SRGen	<b>22.0</b> (+7.4)	<b>9.3</b> (+3.3)	<b>3.3</b> (+2.0)	<b>41.2</b> (+7.2)	<b>33.8</b> (+2.0)	<b>47.6</b> (+1.9)
DS-R1-Qwen-7B	CoT	49.3	35.3	15.3	51.0	50.0	73.1
	Self-Refine	52.0 (+2.7)	38.0 (+2.7)	14.7 (-0.6)	<b>51.6</b> (+0.6)	51.2 (+1.2)	72.6 (-0.5)
	SRGen	<b>61.3</b> (+12.0)	<b>42.7</b> (+7.4)	<b>19.3</b> (+4.0)	51.2 (+0.2)	<b>51.3</b> (+1.3)	<b>73.7</b> (+0.6)
DS-R1-Llama-8B	CoT	48.0	30.7	14.0	50.0	46.4	71.3
	Self-Refine	48.7 (+0.7)	32.0 (+1.3)	16.0 (+2.0)	49.4 (-0.6)	47.9 (+1.5)	<b>73.1</b> (+1.8)
	SRGen	<b>53.3</b> (+5.3)	<b>36.0</b> (+5.3)	<b>18.0</b> (+4.0)	<b>50.6</b> (+0.6)	<b>48.5</b> (+2.1)	72.0 (+0.7)
Qwen3-32B	CoT	76.7	70.7	23.3	54.0	63.5	87.1
	Self-Refine	76.7 (+0.0)	69.3 (-1.4)	25.3 (+2.0)	54.4 (+0.4)	64.9 (+1.4)	87.2 (+0.1)
	SRGen (Ours)	<b>82.7</b> (+6.0)	<b>76.0</b> (+5.3)	<b>28.0</b> (+4.7)	<b>56.8</b> (+2.8)	<b>65.7</b> (+2.2)	<b>87.8</b> (+0.7)

GPQA (Rein et al., 2024) for general reasoning and EvalPlus (Liu et al., 2023) for code generation. These tasks typically require accurate answers, so small early slips tend to propagate and can overturn the final result.

**Experimental Settings.** For inference, we cap the maximum generation length at 4,096 tokens for Qwen2.5-Math-7B and 32,768 for all other models. Decoding uses temperature 0.6 with nucleus sampling at top- $p = 0.95$ , following common recommendations for reasoning models. We also report Qwen2.5-Math-7B performance at temperature 0 in subsequent analyses. All experiments were conducted on NVIDIA A800-80G GPUs. Unless otherwise noted, our method uses the following hyperparameters: inner update steps  $T = 3$ , learning rate  $\eta = 0.01$ , entropy-detection window  $N = 25$ , and standard-deviation multiplier  $k = 4$ . We conduct hyperparameter ablation studies in Appendix G.

**Metrics.** Our accuracy is based on five runs of pass@1.

## 5.2 Main Results

In Table 2, we compare the improvements brought by Self-Refine and SRGen on CoT-style reasoning. The results show that SRGen consistently achieves stronger performance across mathematical, general reasoning, and coding tasks, and this trend holds across models trained under different paradigms. Notably, on AIME2024, SRGen improves the accuracy of DeepSeek-Distill-Qwen-7B by 12 percentage points and Qwen3-32B by 6 percentage points, substantially larger than the gains obtained

by Self-Refine on this type of task.

These results suggest that, even without any external feedback, SRGen can leverage a self-reflective generation process to reduce reasoning errors during decoding. By mitigating cascading mistakes and preventing early errors from propagating into irreversible failure modes, SRGen yields more reliable intermediate reasoning and ultimately improves the model’s overall reasoning capability.

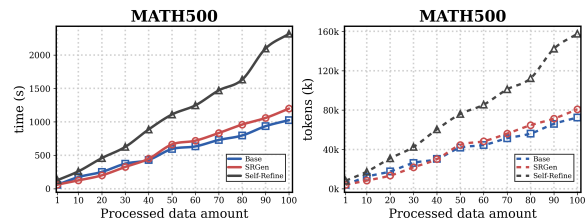


Figure 2: Time increase and token cost.

In addition, we directly compare SRGen with several methods that also intervene during the generation process. To align with the experimental setup of MI-Peak, we use greedy decoding for the math tasks. The results show that SRGen yields more stable improvements in most cases. Table 4 reports detailed results.

## 5.3 Efficiency Analysis

We evaluate inference efficiency after integrating SRGen by measuring both wall-clock runtime and token usage. Experiments are conducted with Qwen2.5-Math-7B on MATH500. We report the mean per-task runtime and the total number of gen-

Table 3: Efficiency comparison on 100 MATH500 problems using Qwen2.5-Math-7B. We report wall-clock time, generated tokens, peak memory usage, and average time per example. Lower is better.

Method	Wall-clock Time (s)	Tokens Generated (w)	Memory Peak (GB)	Avg Time per example (s)
Base	1025	7.2	15.7	10.2
Self-Refine	2316	15.6	18.7	23.2
SLOT	1148	7.5	18.5	11.5
MI-Peak	1744	12.1	16.6	17.4
SRGen	1198	8.0	17.8	12.0

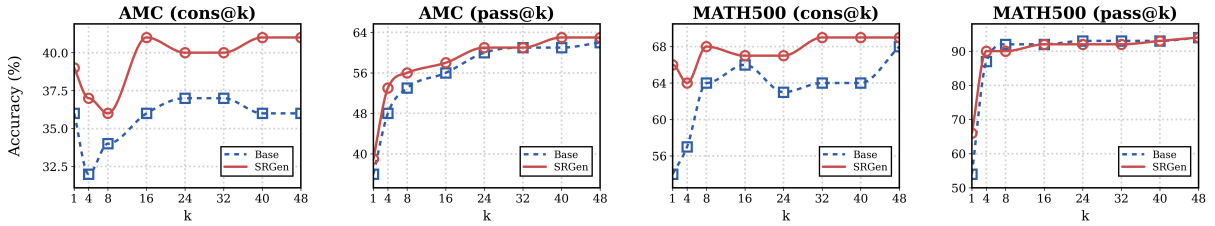


Figure 3: Cons@k and Pass@k accuracy of Qwen2.5-Math-7B on AMC and MATH500

Table 4: Deterministic performance comparison using Greedy Decoding.

Model	Method	MATH	AIME	AMC
Qwen2.5-3B-Instruct	CoT	62.6	10.0	39.0
	MI-Peak	63.8	6.7	41.0
	SLOT	66.2	10.0	40.0
	SRGen	<b>66.6</b>	<b>10.0</b>	<b>43.0</b>
Qwen2.5-Math-7B	CoT	63.8	13.3	35.0
	MI-Peak	64.0	10.0	35.0
	SLOT	64.2	20.0	35.0
	SRGen	<b>69.4</b>	<b>20.0</b>	<b>37.0</b>
DS-R1-Qwen-7B	CoT	85.6	40.0	48.0
	MI-Peak	87.0	50.0	46.0
	SLOT	85.0	46.7	<b>49.0</b>
	SRGen	<b>87.2</b>	<b>53.3</b>	48.0
DS-R1-Llama-8B	CoT	80.4	43.3	42.0
	MI-Peak	81.8	40.0	43.0
	SLOT	84.4	43.3	45.0
	SRGen	<b>84.8</b>	<b>46.7</b>	<b>46.0</b>

erated tokens, and we use greedy decoding for all settings to remove randomness. Figure 2 shows results as we vary the processed data amount. Across the full range, SRGen closely tracks the baseline in both runtime and token consumption. The time overhead remains limited and increases smoothly with the processed data amount, while the token count stays comparable to the base model. In contrast, Self-Refine incurs substantially higher cost, with noticeably steeper growth in both runtime and tokens as the processed data amount increases.

These results indicate that SRGen improves performance while maintaining efficient inference, and it avoids the large efficiency degradation observed with Self-Refine.

By evaluating the efficiency of different methods on 100 MATH500 problems using Qwen2.5-Math-7B, we also provide a more detailed comparison, including wall-clock time, tokens generated, and peak memory usage, as shown in Table 3. The results indicate that, even compared with test-time methods, SRGen still maintains relatively high efficiency while delivering stronger performance.

#### 5.4 Cons@k and Pass@k

Using Qwen2.5-Math-7B on AMC and MATH500, we further examine how Cons@k and Pass@k change with increasing  $k$  after applying SRGen. Detailed results are shown in Figure 3. When SRGen-enhanced samples are used for self-consistency voting, Cons@k remains consistently higher than the base model as  $k$  grows, whereas Pass@k gradually converges to the base model. These trends indicate that self-reflective generation in SRGen reduces the probability of reasoning errors and improves single-pass accuracy, thereby producing higher-quality candidates for self-consistency and boosting Cons@k. Importantly, the reflection mechanism in SRGen does not diminish the exploratory capacity of the model: it primarily reduces mistakes by reweighting uncertain tokens, correcting erroneous traces while leaving correct ones intact. Taken together, these results suggest that SRGen is a promis-

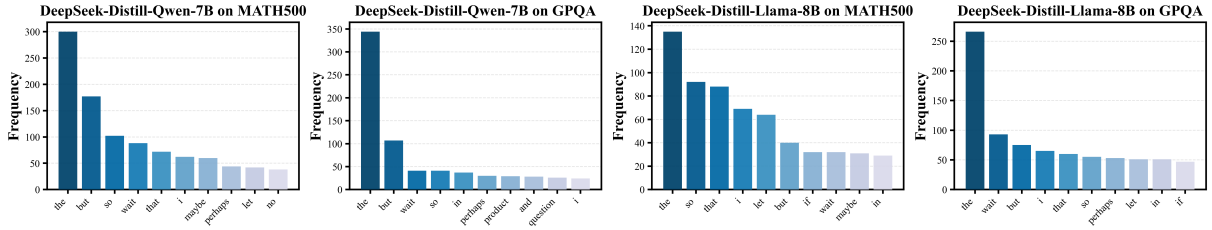


Figure 4: Critical tokens identified by the dynamic entropy monitor.

ing plug-in for achieving higher single-pass accuracy or more sample-efficient self-consistency.

### 5.5 Orthogonality: Integration with Related Methods

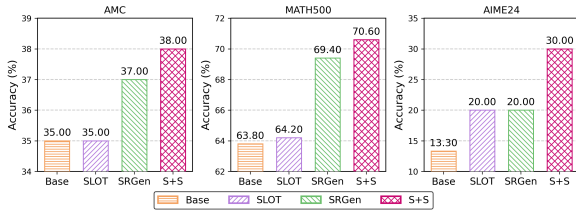


Figure 5: Performance of SLOT, SRGen, and their combination for Qwen2.5-Math-7B across the AMC, MATH500, and AIME2024 benchmarks.

Building on our initial results showing that SRGen can pair with diverse training paradigms and further improve models post-training, we next provide stronger evidence of its orthogonality and potential synergy by combining it with a method from the same family. Specifically, we adopt SLOT, a representative test-time optimization approach. During inference, SLOT optimizes a sample-specific vector  $\delta_{\text{SLOT}}$  over the prompt-processing stage and injects it into the hidden states of the model to steer generation. All experiments are conducted on the Qwen2.5-Math-7B model, and we use greedy decoding throughout to ensure reproducibility.

As shown in Figure 5, combining SLOT with SRGen further strengthens the reasoning ability of the model, with particularly pronounced gains on mathematical tasks. On MATH500, the joint approach lifts Qwen2.5-Math-7B from 63.8% to 70.6%, outperforming either SLOT or SRGen used in isolation. On AMC, it improves performance from 35.0% to 38.0%, exceeding both SLOT (35.0%) and SRGen (37.0%). The synergy is even more evident on AIME2024, where the combined method boosts accuracy from 13.3% to 30.0%, substantially outperforming either SLOT or SRGen alone (both at

20.0%). These results provide additional evidence that SRGen

### 5.6 Critical Token Identified

In this section, we analyze which tokens are flagged by our dynamic entropy monitor. We run DeepSeek-R1-Distill-Qwen-7B and DeepSeek-R1-Distill-Llama-8B on MATH500 and GPQA, and collect tokens whose next-token uncertainty exceeds an adaptive threshold. Figure 4 reports the aggregated frequencies.

Across both models and benchmarks, the most frequent tokens are mainly function words and discourse connectives, including the, so, but, that, perhaps, maybe, let, if, and for. These tokens often appear near clause boundaries and decision points in the chain of thought, where the model chooses between plausible continuations such as introducing a new premise, changing direction, or committing to the next step. As a result, uncertainty spikes are more likely to occur on tokens that guide the reasoning trajectory, rather than on content tokens that only add details.

This pattern supports our design choice. A simple dynamic threshold tends to surface high-impact connective tokens, which provides meaningful intervention points without spending computation on low-value positions. In practice, concentrating inner updates on this compact set keeps optimization stable and efficient. Early in the updates, the cross-entropy term and the entropy-reduction term often push in similar directions, improving both confidence and consistency with the current context. As updates continue, the two objectives may start to conflict. The weight  $\lambda$  then controls the trade-off between fitting the current context and reducing uncertainty at each triggered step.

Overall, Figure 4 suggests that entropy-guided, token-aware intervention targets the specific junctures that most strongly influence downstream reasoning. This helps explain the consistent gains we observe across models and benchmarks, while

keeping the overhead modest and bounded.

## 6 Conclusion

We introduce *Self-Reflective Generation at Test Time* (SRGen), a lightweight, plug-and-play framework that performs token-level self-reflection only at critical tokens detected by *dynamic uncertainty monitoring*. When triggered, a brief *self-reflective optimization* learns an on-the-fly correction vector  $\delta$  and injects it into the hidden state under a hybrid loss  $L_{\text{SRGen}}$  to reduce predictive uncertainty while preserving contextual fidelity. Across challenging benchmarks spanning mathematical reasoning, general reasoning, and code generation, SRGen improves accuracy with small additional inference time and yields more effective self-consistency voting without harming exploration. SRGen is practical: it strengthens reasoning and generation across model architectures and training paradigms and composes with other test-time methods, making it a promising inference-time plug-in for more reliable LLM generation.

Moreover, the effectiveness of SRGen further validates the importance of critical tokens in the model’s reasoning process, suggesting that appropriately intervening on these tokens can effectively improve reasoning performance

## Limitations

In this section, we discuss several potential limitations of SRGen. First, SRGen intervenes during generation based on an uncertainty signal and performs online optimization of the corrective vector at each trigger point. This typically requires access to the model’s hidden states and gradient information. As a result, SRGen is most suitable for controllable open-weight or white-box inference settings, and it may be difficult to deploy in scenarios that only provide a text-only interface or a restricted inference stack. Second, SRGen relies on a dynamic threshold to identify uncertain positions. As shown in Section 5.6, this strategy can effectively capture key junctions in the reasoning process, and intervening at these positions leads to consistent gains. However, the current method for identifying such critical points still has substantial room for improvement. Future work could explore stronger position-selection strategies so that SRGen can target more precise and informative locations. Third, our method introduces several additional hyperparameters. In Appendix

G, we show that SRGen is relatively stable across different hyperparameter choices and we provide a general default configuration. Nevertheless, the hyperparameters that yield the best performance may vary across models and tasks. For settings that require peak performance, future work may study efficient tuning on a small validation set, or develop adaptive hyperparameter schemes to improve the applicability of SRGen.

## Ethical Considerations

Our method aims to incorporate self-reflective decision making into the model’s generation process and can be easily integrated as a test-time technique. It does not create new datasets or train new models and introduce additional risks related to discrimination, bias, exploitation, human rights violations, or other ethical concerns.

## References

- Art of Problem Solving. 2024. Aime problems and solutions. [https://artofproblemsolving.com/wiki/index.php/AIME\\_Problems\\_and\\_Solutions](https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions). Accessed: 2025-09-14.
- Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, and 1 others. 2022. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*.
- Mislav Balunović, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. 2025. [Matharena: Evaluating llms on uncontaminated math competitions](#).
- Shelly Bensal, Umar Jamil, Christopher Bryant, Melisa Russak, Kiran Kamble, Dmytro Mozolevskiy, Muayad Ali, and Waseem AlShikh. 2025. Reflect, retry, reward: Self-improving llms via reinforcement learning. *arXiv preprint arXiv:2505.24726*.
- Zhenni Bi, Kai Han, Chuanjian Liu, Yehui Tang, and Yunhe Wang. 2024. Forest-of-thought: Scaling test-time compute for enhancing llm reasoning. *arXiv preprint arXiv:2412.09078*.
- Yung-Sung Chuang, Yujia Xie, Hongyin Luo, Yoon Kim, James R Glass, and Pengcheng He. 2023. Dola: Decoding by contrasting layers improves factuality in large language models. In *The Twelfth International Conference on Learning Representations*.
- Yichao Fu, Xuewei Wang, Yuandong Tian, and Jiawei Zhao. 2025. Deep think with confidence. *arXiv preprint arXiv:2508.15260*.

- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shitong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.
- Hasan Abed Al Kader Hammoud, Hani Itani, and Bernard Ghanem. 2025. Beyond the last answer: Your reasoning trace uncovers more than you think. *arXiv preprint arXiv:2504.20708*.
- Chi Hu, Yimin Hu, Hang Cao, Tong Xiao, and Jingbo Zhu. 2024. Teaching language models to self-improve by learning from language feedback. *arXiv preprint arXiv:2406.07168*.
- Yang Hu, Xingyu Zhang, Xueji Fang, Zhiyang Chen, Xiao Wang, Huatian Zhang, and Guojun Qi. 2025. Slot: Sample-specific language model optimization at test-time. *arXiv preprint arXiv:2505.12392*.
- Kushal Jain, Moritz Miller, Niket Tandon, and Kumar Shridhar. 2025. **First-step advantage: Importance of starting right in multi-step math reasoning**. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 766–778, Vienna, Austria. Association for Computational Linguistics.
- Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, and 1 others. 2024. Training language models to self-correct via reinforcement learning. *arXiv preprint arXiv:2409.12917*.
- Jia LI, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang, Kashif Rasul, Longhui Yu, Albert Jiang, Ziju Shen, Zihan Qin, Bin Dong, Li Zhou, Yann Fleureau, Guillaume Lample, and Stanislas Polu. 2024. Numinamath.
- Yafu Li, Xuyang Hu, Xiaoye Qu, Linjie Li, and Yu Cheng. 2025. Test-time preference optimization: On-the-fly alignment via iterative textual feedback. *arXiv preprint arXiv:2501.12895*.
- Zicheng Lin, Tian Liang, Jiahao Xu, Qiuzhi Liu, Xing Wang, Ruilin Luo, Chufan Shi, Siheng Li, Yujiu Yang, and Zhaopeng Tu. 2025. Critical tokens matter: Token-level contrastive estimation enhances llm’s reasoning capability. In *Forty-second International Conference on Machine Learning*.
- Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in Neural Information Processing Systems*, 36:21558–21572.
- Ruotian Ma, Peisong Wang, Cheng Liu, Xingyan Liu, Jiaqi Chen, Bang Zhang, Xin Zhou, Nan Du, and Jia Li. 2025. S<sup>2</sup>r: Teaching llms to self-verify and self-correct via reinforcement learning. *arXiv preprint arXiv:2502.12853*.
- Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegrefe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, and 1 others. 2023. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594.
- Viktor Moskvoretskii, Chris Biemann, and Irina Nishina. 2025. Self-taught self-correction for small language models. *arXiv preprint arXiv:2503.08681*.
- Debjit Paul, Robert West, Antoine Bosselut, and Boi Faltings. 2024. Making reasoning matter: Measuring and improving faithfulness of chain-of-thought reasoning. *arXiv preprint arXiv:2402.13950*.
- Aske Plaat, Annie Wong, Suzan Verberne, Joost Broekens, Niki van Stein, and Thomas Bäck. 2024. Reasoning with large language models, a survey. *CoRR*.
- Chen Qian, Dongrui Liu, Haochen Wen, Zhen Bai, Yong Liu, and Jing Shao. 2025. Demystifying reasoning dynamics with mutual information: Thinking tokens are information peaks in llm reasoning. *arXiv preprint arXiv:2506.02867*.
- David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. 2024. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*.
- Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652.
- Nishad Singhi, Hritik Bansal, Arian Hosseini, Aditya Grover, Kai-Wei Chang, Marcus Rohrbach, and Anna Rohrbach. 2025. When to solve, when to verify: Compute-optimal problem solving and generative verification for llm reasoning. *arXiv preprint arXiv:2504.01005*.
- Amir Taubenfeld, Tom Sheffer, Eran Ofek, Amir Feder, Ariel Goldstein, Zorik Gekhman, and Gal Yona. 2025. Confidence improves self-consistency in llms. *arXiv preprint arXiv:2502.06233*.
- Qwen Team. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.
- Fengwei Teng, Zhaoyang Yu, Quan Shi, Jiayi Zhang, Chenglin Wu, and Yuyu Luo. 2025. Atom of thoughts for markov llm test-time scaling. *arXiv preprint arXiv:2502.12018*.
- Jean Vassoyan, Nathanaël Beau, and Roman Plaud. 2025. Ignore the kl penalty! boosting exploration on critical tokens to enhance rl fine-tuning. *arXiv preprint arXiv:2502.06533*.

- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.
- Shenzhi Wang, Le Yu, Chang Gao, Chujie Zheng, Shixuan Liu, Rui Lu, Kai Dang, Xionghui Chen, Jianxin Yang, Zhenru Zhang, and 1 others. 2025. Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for llm reasoning. *arXiv preprint arXiv:2506.01939*.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.
- Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.
- Xiao Yu, Baolin Peng, Michel Galley, Jianfeng Gao, and Zhou Yu. 2024. Teaching language models to self-improve through interactive demonstrations. In *NAACL-HLT*.
- Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. 2024. Textgrad: Automatic "differentiation" via text. *arXiv preprint arXiv:2406.07496*.
- Tianyu Zheng, Tianshun Xing, Qingshui Gu, Taoran Liang, Xingwei Qu, Xin Zhou, Yizhi Li, Zhoufutu Wen, Chenghua Lin, Wenhao Huang, and 1 others. 2025. First return, entropy-eliciting explore. *arXiv preprint arXiv:2507.07017*.
- Zhi Zhou, Tan Yuhao, Zenan Li, Yuan Yao, Lan-Zhe Guo, Xiaoxing Ma, and Yu-Feng Li. 2025. Bridging internal probability and self-consistency for effective and efficient llm reasoning. *arXiv preprint arXiv:2502.00511*.
- Yuqi Zhu, Ge Li, Xue Jiang, Jia Li, Hong Mei, Zhi Jin, and Yihong Dong. 2025. Uncertainty-guided chain-of-thought for code generation with llms. *arXiv preprint arXiv:2503.15341*.

## A Related Work

**Self-Reflection in LLMs.** Self-reflection seeks to move LLMs from impulsive first-pass outputs to more deliberative and accurate responses. Existing approaches largely fall into two categories. (1) Post hoc iterative refinement. These methods use multi-stage inference pipelines: the model first drafts an answer, then critiques it, and finally revises it based on its own feedback (Shinn et al., 2023; Yu et al., 2024; Yuksekogul et al., 2024; Li et al., 2025). Frameworks such as Self-Refine (Madaan et al., 2023) formalize the generate, critique, and refine loop. While effective, they incur substantial latency and computational overhead because they require multiple full forward passes. (2) Training for intrinsic self-correction. This line embeds self-correction directly in the parameters, typically via fine-tuning on corrective data or reinforcement learning (RL) (Bai et al., 2022; Hu et al., 2024; Kumar et al., 2024; Moskvoretskii et al., 2025). Such models can produce refined outputs in a single pass but demand extensive and costly training. SRGen offers a distinct alternative that operates at test time. It avoids the high overhead of iterative methods by requiring neither auxiliary LLM calls for feedback nor the generation of multiple complete outputs. Operating at the token level, SRGen is a lightweight method and can be synergistically combined with both training-based and post-hoc self-reflection approaches.

### Identifying and Leveraging Critical Tokens.

Recent work rests on the observation that tokens are not equally informative (Lin et al., 2025). Studies identify “critical” or “pivotal” tokens that mark decision points along a reasoning path and leverage them in several ways. (1) Guiding training: policy gradients are applied selectively at high-entropy positions to focus learning (Wang et al., 2025; Vas-soyan et al., 2025). (2) Triggering exploration: critical tokens act as branching points for sampling diverse reasoning paths (Zheng et al., 2025; Zhu et al., 2025) or for localized iterative refinement that probes the solution space more deeply (Qian et al., 2025). (3) Pruning search: low-confidence tokens trigger the removal of less promising paths within self-consistency frameworks (Fu et al., 2025; Taubenfeld et al., 2025; Zhou et al., 2025). We introduce a new paradigm for the use of critical tokens. SRGen employs them as real-time triggers for a corrective intervention directly on the model’s hidden state. This allows for an on-the-fly steer-

ing of a single generation path, a fundamentally different and more direct mechanism than prior art.

**Test-Time Scaling.** To improve performance without costly retraining, a range of methods increase computation at test time. These approaches fall into two broad strategies. The first generates multiple reasoning paths and selects an outcome via voting or scoring. This includes producing multiple complete solutions, as in self-consistency (Wang et al., 2022; Singhi et al., 2025), or exploring a tree or graph of intermediate steps, as in Tree-of-Thoughts and its variants (Yao et al., 2023; Bi et al., 2024; Teng et al., 2025). The second intervenes within a single decoding process. Prompt-based techniques such as chain of thought (Wei et al., 2022) elicit more deliberative reasoning. More directly, methods adjust the model’s internal computations during a single pass, e.g., DoLa (Chuang et al., 2023) contrasts layer logits to steer decoding and SLOT (Hu et al., 2025) injects a sample-specific vector into the hidden states to steer generation globally and indirectly encourages longer reasoning by suppressing the EOS token. SRGen advances the second strategy with a fine-grained, dynamic intervention. Whereas SLOT applies a static, sample-level vector throughout decoding, SRGen computes a token-level corrective vector  $\delta$  on-the-fly at detected critical junctures. This targeted adjustment adapts to the immediate context and steers the reasoning process without branching or multiple full passes.

## B More Results

## C Theorems and proofs

### C.1 Proof of Thm. 1

**Statement.** Fix a trade-off parameter  $\lambda \in (0, 1)$ . Let

$$F_\lambda(\delta) \triangleq (1 - \lambda)L_{CE}(\delta) + \lambda L_{AEM}(\delta),$$

where  $L_{CE}$  and  $L_{AEM}$  are defined in the main text (§ 3.3) on the current prefix  $y_{<t}$  with the same correction vector  $\delta$  injected as in Eq. 4. Let  $\delta^* \in \arg \min_\delta F_\lambda(\delta)$  be any minimizer. Then  $\delta^*$  also solves the constrained problem

$$\min_\delta L_{AEM}(\delta) \quad \text{s.t.} \quad L_{CE}(\delta) \leq \varepsilon, \quad (11)$$

with the *implicitly induced* tolerance  $\varepsilon \triangleq L_{CE}(\delta^*)$ .

**Proof.** Let  $\varepsilon = L_{CE}(\delta^*)$ . By construction,  $\delta^*$  is feasible for (11) with the constraint held at equality.

Suppose, for contradiction, that  $\delta^*$  is not an optimizer of (11); then there exists a feasible  $\widehat{\delta}$  with  $L_{\text{CE}}(\widehat{\delta}) \leq \varepsilon$  and  $L_{\text{AEM}}(\widehat{\delta}) < L_{\text{AEM}}(\delta^*)$ . Consider the hybrid objective values:

$$\begin{aligned} F_\lambda(\widehat{\delta}) &= (1 - \lambda)L_{\text{CE}}(\widehat{\delta}) + \lambda L_{\text{AEM}}(\widehat{\delta}) \\ &\leq (1 - \lambda)\varepsilon + \lambda L_{\text{AEM}}(\widehat{\delta}) \\ &< (1 - \lambda)\varepsilon + \lambda L_{\text{AEM}}(\delta^*) \\ &= F_\lambda(\delta^*). \end{aligned} \quad (12)$$

which contradicts the optimality of  $\delta^*$  for  $F_\lambda$ . Hence no feasible point can achieve a strictly smaller  $L_{\text{AEM}}$  under the tolerance  $L_{\text{CE}} \leq \varepsilon$ , and  $\delta^*$  solves (11).

**Lagrangian view and the  $\lambda$ - $\alpha$  mapping.** The constrained problem (11) has Lagrangian

$$\mathcal{L}(\delta, \alpha) = L_{\text{AEM}}(\delta) + \alpha(L_{\text{CE}}(\delta) - \varepsilon), \quad \alpha \geq 0.$$

For any fixed  $\alpha \geq 0$ , minimizing  $\mathcal{L}$  over  $\delta$  is, up to an additive constant  $-\alpha\varepsilon$ , equivalent to minimizing the *weighted sum*  $L_{\text{AEM}}(\delta) + \alpha L_{\text{CE}}(\delta)$ . Identifying weights gives the bijection

$$\lambda = \frac{1}{1 + \alpha}, \quad 1 - \lambda = \frac{\alpha}{1 + \alpha}, \quad \alpha = \frac{1 - \lambda}{\lambda}.$$

Therefore, the hybrid loss  $F_\lambda$  is exactly a rescaled Lagrangian with dual variable  $\alpha = \frac{1 - \lambda}{\lambda}$ . Under standard regularity ensuring KKT optimality (e.g., existence of a primal optimum and either convexity with Slater’s condition or other sufficient conditions for strong duality), any primal-dual optimal pair  $(\delta^*, \alpha^*)$  of (11) also minimizes a weighted sum, and the mapping above recovers  $\lambda$  from  $\alpha^*$ . This provides the converse direction under mild assumptions: *for a given active tolerance  $\varepsilon$ , an appropriate choice of  $\lambda$  (equivalently,  $\alpha$ ) recovers the same optimizer  $\delta^*$ .*

**Pareto-optimality interpretation.** Consider the bi-objective vector  $G(\delta) \triangleq (L_{\text{CE}}(\delta), L_{\text{AEM}}(\delta))$ . By the proof above, any minimizer  $\delta^*$  of  $F_\lambda$  with  $\lambda \in (0, 1)$  is *Pareto-optimal*: if there existed  $\widehat{\delta}$  with  $G(\widehat{\delta}) \preceq G(\delta^*)$  and one component strictly smaller, it would violate the optimality of  $\delta^*$  for  $F_\lambda$ . Hence the hybrid loss selects points on the Pareto front of the two desiderata “contextual fidelity” and “uncertainty reduction.” In particular, the induced tolerance  $\varepsilon = L_{\text{CE}}(\delta^*)$  characterizes the specific frontier point attained.

**On the  $\lambda \leftrightarrow \varepsilon$  trade-off.** Intuitively, larger  $\lambda$  increases the relative price on  $L_{\text{AEM}}$  and relaxes the pressure on  $L_{\text{CE}}$ , thus tending to yield solutions

with lower  $L_{\text{AEM}}$  and (weakly) higher  $L_{\text{CE}}$  (i.e., a looser fidelity tolerance). Formally, consider any  $0 < \lambda_1 < \lambda_2 < 1$  with corresponding minimizers  $\delta_1, \delta_2$ . Optimality implies

$$F_{\lambda_1}(\delta_1) \leq F_{\lambda_1}(\delta_2), \quad F_{\lambda_2}(\delta_2) \leq F_{\lambda_2}(\delta_1).$$

Writing  $F_\lambda = (1 - \lambda)L_{\text{CE}} + \lambda L_{\text{AEM}}$  and rearranging yields the *weighted* trade-off bounds:

$$\begin{aligned} &\frac{1 - \lambda_1}{\lambda_1} [L_{\text{CE}}(\delta_1) - L_{\text{CE}}(\delta_2)] \\ &\leq L_{\text{AEM}}(\delta_2) - L_{\text{AEM}}(\delta_1) \\ &\leq \frac{1 - \lambda_2}{\lambda_2} [L_{\text{CE}}(\delta_1) - L_{\text{CE}}(\delta_2)]. \end{aligned} \quad (13)$$

Thus the *relative weights*  $(1 - \lambda)/\lambda$  govern the paired improvements: as  $\lambda$  increases (placing more emphasis on  $L_{\text{AEM}}$ ), the achievable decrease in  $L_{\text{AEM}}$  per unit increase in  $L_{\text{CE}}$  becomes tighter. In strictly convex or uniqueness regimes this typically induces a monotone path  $\varepsilon(\lambda) = L_{\text{CE}}(\delta_\lambda)$  that is nondecreasing in  $\lambda$ .<sup>1</sup>

**Boundary cases and feasibility.** When  $\lambda \rightarrow 1$  ( $\alpha \rightarrow 0$ ),  $F_\lambda$  approaches  $L_{\text{AEM}}$ , i.e., the unconstrained entropy-minimization objective. The induced tolerance becomes  $\varepsilon = L_{\text{CE}}(\delta^*)$  for an  $L_{\text{AEM}}$ -minimizer  $\delta^*$ , so the constraint is *tight (active at equality)* in this mapping. When  $\lambda \rightarrow 0$  ( $\alpha \rightarrow \infty$ ),  $F_\lambda$  emphasizes  $L_{\text{CE}}$  and the solution tends to minimize contextual distortion subject to making any progress on  $L_{\text{AEM}}$ ; operationally this corresponds to a nearly “hard” fidelity constraint.

**Existence of minimizers.** Both  $L_{\text{CE}}$  and  $L_{\text{AEM}}$  in our setting are nonnegative and continuous in  $\delta$  (they are compositions of smooth maps: affine shift in logits, softmax, entropy, and prefix NLL). Thus  $F_\lambda$  is lower-bounded by 0 and continuous. If  $\arg \min F_\lambda$  fails to exist on  $\mathbb{R}^d$  due to lack of coercivity, it suffices (and is standard at test time) to either: (i) restrict  $\delta$  to a compact trust region  $\{\|\delta\| \leq R\}$ , or (ii) add a tiny quadratic regularizer  $\frac{\gamma}{2}\|\delta\|^2$  (this does not affect the equivalence to (11) because the same regularizer can be added to both the constrained and weighted formulations). Under either modification, a minimizer exists and the above arguments apply verbatim.

**Joint-Descent Lemma.** If the gradients form an acute angle, i.e.,  $\langle \nabla L_{\text{CE}}(\delta), \nabla L_{\text{AEM}}(\delta) \rangle > 0$ ,

<sup>1</sup>We avoid global convexity claims for  $L_{\text{AEM}}$ ; the sufficiency result above does not require convexity. In practice, a unique local minimizer selected by a deterministic inner optimizer makes the  $\lambda$ -path stable.

then for any  $\lambda \in (0, 1)$  and sufficiently small  $\eta > 0$ , the update

$$\delta^+ = \delta - \eta [(1 - \lambda) \nabla L_{\text{CE}}(\delta) + \lambda \nabla L_{\text{AEM}}(\delta)]$$

strictly decreases *both* objectives to first order. *Proof.* Directional derivatives give

$$\begin{aligned} \frac{d}{d\eta} L_{\text{CE}}(\delta^+) \Big|_{\eta=0} &= -[(1 - \lambda) \|\nabla L_{\text{CE}}\|^2 + \lambda \langle \nabla L_{\text{CE}}, \nabla L_{\text{AEM}} \rangle] \\ &< 0. \end{aligned} \quad (14)$$

$$\begin{aligned} \frac{d}{d\eta} L_{\text{AEM}}(\delta^+) \Big|_{\eta=0} &= -[\lambda \|\nabla L_{\text{AEM}}\|^2 + (1 - \lambda) \langle \nabla L_{\text{AEM}}, \nabla L_{\text{CE}} \rangle] \\ &< 0. \end{aligned} \quad (15)$$

where both inequalities use the acute-angle assumption.  $\square$

**Takeaway for SRGen.** The proof establishes that the SRGen loss is not an ad-hoc blend: it is precisely a Lagrangian relaxation of the constrained goal “reduce uncertainty while keeping contextual fidelity within tolerance.” Therefore  $\lambda$  is an interpretable knob: it *implicitly* sets the fidelity tolerance  $\varepsilon = L_{\text{CE}}(\delta^*)$  and moves SRGen along the fidelity–confidence Pareto frontier. In practice (cf. Fig. 5 in the main text), small but nonzero  $\lambda$  often works well, reflecting a regime where fidelity is enforced strongly while still reaping entropy reductions at high-uncertainty points.

## D Algorithm of SRGen

**Explanation Alg. 1.** **L1–2** Inputs/outputs and hyperparameters.  $k$  controls trigger sensitivity;  $N$  is the entropy history window;  $\lambda$  balances contextual fidelity vs. entropy minimization;  $T, \eta$  set the inner-loop budget;  $\tau$  is the decoding temperature. **L3** Initialize the sequence and a ring buffer  $E$  to maintain recent entropies for on-the-fly calibration. This enables model/temperature/position-agnostic triggering. **L4–6** At each step, obtain the last hidden state  $h_{t-1}$ , project to logits  $z$ , and compute predictive entropy  $E_t = H(\text{softmax}(z/\tau))$ . **L7** Dynamic trigger: activate reflection iff  $E_t$  significantly exceeds the local baseline via  $E_t > \mu(E) + k \sigma(E)$ , where  $\mu, \sigma$  are rolling stats over the last  $N$  steps. **L8–9** Enter a short inner optimization while freezing  $M, W$  and optimizing a transient correction vector  $\delta$  only

---

### Algorithm 1 SRGen: Self-Reflective Generation

---

- 1: **Input:** pre-trained model  $M$  with head  $W$ ; prompt  $x_0$   
*Hyperparameters:*  $k$  (sensitivity),  $N$  (window size),  $\lambda$  (loss weight),  $T$  (steps),  $\eta$  (lr),  $\tau$  (temperature)
- 2: **Output:** generated sequence  $y$
- 3:  $y \leftarrow ()$ ,  $t \leftarrow 1$ ,  $\mathcal{E} \leftarrow$  empty ring buffer of size  $N$
- 4: **while** EOS not generated **and**  $|y| < \text{MAX\_LENGTH}$  **do**
- 5:    $h_{t-1} \leftarrow M(x_{0:t})$     $\triangleright$  last hidden state for current context
- 6:    $z \leftarrow W h_{t-1}$ ;    $E_t \leftarrow \text{Entropy}(\text{softmax}(z/\tau))$
- 7:   **if**  $|\mathcal{E}| = N$  **and**  $E_t > \mu(\mathcal{E}) + k \sigma(\mathcal{E})$  **then**  
     $\triangleright$  dynamic trigger
- 8:      $\delta \leftarrow \mathbf{0}$
- 9:     **for**  $i = 1$  **to**  $T$  **do**  $\triangleright$  inner optimization of  $\delta$
- 10:        $\mathcal{L}_{\text{CE}} \leftarrow -\sum_{j=0}^{t-2} \log p(x_{j+1} \mid x_{0:j}, \delta)$
- 11:        $\mathcal{L}_{\text{AEM}} \leftarrow -\sum_{v \in \mathcal{V}} p(v \mid x_{0:t}, \delta) \log p(v \mid x_{0:t}, \delta)$
- 12:        $\mathcal{L} \leftarrow (1 - \lambda) \mathcal{L}_{\text{CE}} + \lambda \mathcal{L}_{\text{AEM}}$
- 13:        $\delta \leftarrow \delta - \eta \nabla_{\delta} \mathcal{L}$
- 14:     **end for**
- 15:      $z \leftarrow W(h_{t-1} + \delta)$   $\triangleright$  modify only the last state at sampling
- 16:     **end if**
- 17:      $y_t \sim \text{softmax}(z/\tau)$ ;    $y \leftarrow y \oplus y_t$ ;  
     $x_{0:t+1} \leftarrow x_{0:t} \oplus y_t$ ;    $t \leftarrow t + 1$
- 18:     push  $E_t$  into  $\mathcal{E}$  and keep the most recent  $N$
- 19: **end while**
- 20: **return**  $y$

---

when needed. **L10** Retrospective context loss  $L_{\text{CE}}$  preserves prefix fidelity by applying the same  $\delta$  to historical states when computing teacher-forced likelihood of  $x_{j+1}$ . **L11** Anticipatory entropy minimization  $L_{\text{AEM}}$  sharpens the current predictive distribution to reduce uncertainty at the flagged token. **L12** Hybrid objective  $(1 - \lambda)L_{\text{CE}} + \lambda L_{\text{AEM}}$  trades off stability and decisiveness; small  $\lambda$  avoids collapse while still decreasing entropy. **L13** Update  $\delta$  with a few small steps ( $T$  typically  $\leq 5$ ), keeping overhead bounded. **L15** Inject  $\delta$  only at the current step for sampling; historical injection appears only inside the loss terms, so past tokens are not altered. **L17–20** Sample, update the context and entropy buffer, and continue until EOS or length limit.

## E Prompt Used

To facilitate reproducibility, we provide the system prompt used in our benchmark evaluations.

### AIME2024 and AIME2025

You are a helpful assistant. Solve the following math problem efficiently and clearly. The last line of your response should be of the following format: 'Therefore, the final answer is:

$\boxed{\{ANSWER\}}$

I hope it is correct' (without quotes) where ANSWER is just the final number that solves the problem. Think step by step before answering.

### MATH500 and HMMT2025 and AMC

Solve the following math problem efficiently and clearly. The last line of your response should be of the following format: 'Therefore, the final answer is:

$\boxed{\{ANSWER\}}$

I hope it is correct' (without quotes) where ANSWER is just the final number or expression that solves the problem. Think step by step before answering.

### GPQA

You are a helpful assistant. A conversation between User and Assistant. The user asks a question, and the Assistant solves it. The Assistant first thinks about the reasoning process in the mind and then provides the user with the answer. Answer the following multiple choice question. The last line of your response should be of the following format: 'Answer: \$LETTER' (without quotes) where LETTER is one of ABCD. Think step by step before answering.

### EvalPlus

Please provide a self-contained Python script that solves the following problem in a markdown code block:

## F Entropy Analysis

A fixed entropy threshold does not generalize across models, temperatures, or positions in the same sequence. Figures 6a ( $T=0$ ) and 6b ( $T=0.6$ ) show large differences in the scale and variance of token-level entropy across architectures and post-training regimes. For example, at  $T=0$  the final-step entropy ranges from  $\approx 2 \times 10^{-4}$  for Qwen2.5-Math-7B to  $\approx 0.66$  for Qwen3-32B. At  $T=0.6$  the final-step entropy is  $\approx 0.0002$  (Qwen2.5-Math-7B),  $\approx 0.3145$  (DeepSeek-R1-Distill-Llama-8B),  $\approx 0.0918$  (DeepSeek-R1-Distill-Qwen-7B), and  $\approx 0.0011$  (Qwen3-32B). Sequence lengths also vary widely (e.g.,  $\sim 11,174$  steps for DeepSeek-R1-Distill-Llama-8B versus  $\sim 780$  for Qwen3-32B at  $T=0.6$ ), and within a sequence the baseline entropy drifts while sharp local spikes persist. Under any fixed threshold  $\tau$ , low-entropy models would rarely trigger (missed high-risk segments), whereas high-entropy models would trigger excessively (many false positives); temperature changes further skew the trigger rate.

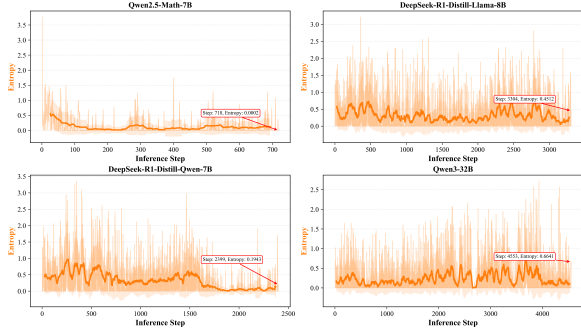
To handle these distribution shifts, SRGen uses a dynamic threshold based on a rolling estimate of the local entropy distribution. At step  $t$ , with predictive entropy  $H_t$ , we compute the mean  $\mu_t$  and standard deviation  $\sigma_t$  over a history window of length  $N$ , and set

$$\tau_t = \mu_t + k \sigma_t, \quad \text{trigger if } H_t \geq \tau_t. \quad (16)$$

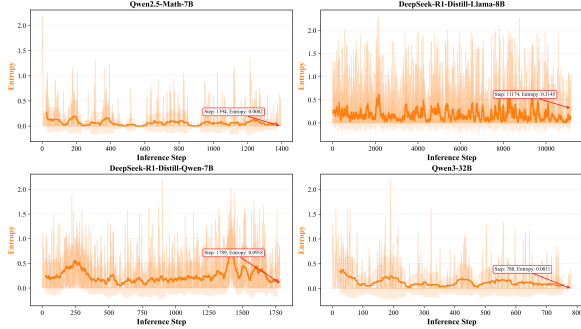
This adaptive rule calibrates to each model, temperature, and stage of decoding: it detects relative spikes in low-entropy models, avoids always-on firing in high-entropy models, and tracks non-stationary drift along the reasoning trajectory. The trajectories in Figures 6a and 6b illustrate that the rule consistently activates on local high-risk segments, enabling proactive test-time intervention without extra decoding passes.

## G Hyperparameter Ablation Study

We perform an ablation study to examine the key hyperparameters of SRGen. Specifically, we vary **iterations** (the number of gradient steps used to optimize  $\delta$ ), **learning rate** (the step size for updating  $\delta$ ), the **balancing coefficient**  $\lambda$ , and the two parameters of our dynamic-entropy monitoring module: **history window size**  $N$  and **standard-deviation multiplier**  $k$ . To clearly reveal performance trends while keeping computation manageable, we evaluate on the first 100 instances of the MATH500 bench-



(a) Entropy trajectories of different models (temperature = 0)



(b) Entropy trajectories of different models (temperature = 0.6)

Figure 6: Entropy trajectories of different models (temperature = 0 and 0.6).

mark. All experiments use the Qwen2.5-Math-7B with *greedy decoding*.

Table 5: Effects of iterations and learning rate. Iterations=0 corresponds to the baseline (original model). The balancing hyperparameter  $\lambda$  is fixed to 0.05.

lr	Iterations					
	0	1	3	5	7	9
<b>0.01</b>	64.0	71.0	70.0	<b>72.0</b>	70.0	71.0
<b>0.05</b>	64.0	<b>72.0</b>	71.0	71.0	71.0	71.0
<b>0.10</b>	64.0	71.0	71.0	71.0	71.0	71.0

As shown in Table 5, varying the number of optimization iterations and the learning rate within reasonable ranges yields only minor changes in performance. Across all hyperparameter settings in our ablation, accuracy remains stable at roughly 71% ( $\pm 1\%$ ), indicating that SRGen is relatively insensitive to these choices.

Figure 7 shows even at the extremes ( $\lambda = 0$ , disabling entropy minimization;  $\lambda = 1$ , disabling cross-entropy), SRGen yields substantial gains over the baseline, reinforcing our claim that targeted intervention on critical tokens is effective. The strongest performance arises when both losses are

used, indicating synergy and motivating careful calibration of  $\lambda$ . In practice, small  $\lambda$  values that place greater weight on the cross-entropy term work best; for example,  $\lambda = 0.05$  performs consistently well across our tests. Varying  $N$  and  $k$ , we observe that a small  $N$  fails to capture the recent entropy trend, whereas an overly large  $N$  becomes unrepresentative by incorporating too many outliers. Choosing  $N \in [25, 40]$  better tracks short-horizon entropy dynamics. For  $k$ , smaller values flag more tokens as uncertain, leading to abnormally high trigger counts and reduced efficiency and reflecting on too many tokens yields little additional gain while risking disruption of correct reasoning by perturbing high-confidence tokens. Conversely, very large  $k$  misses many critical tokens. Values around  $k \in [2.5, 4]$  strike a balance: they identify critical tokens broadly while keeping triggers modest, yielding larger improvements.

## H Loss in training

We analyze the loss-reduction dynamics observed during optimization, providing empirical guidance for selecting the learning rate. SRGen is a method for rapid, on-the-fly optimization at test time.

Using Qwen2.5-Math-7B and DeepSeek-R1-Distill-Qwen, we perform up to 1000 inner-loop updates on the correction vector  $\delta$  at a single uncertainty trigger, and report the resulting loss curves in Figure 8a and 8b. The curves show that larger learning rates are well suited to our on-the-fly procedure: they drive the objective down quickly and reach a stable plateau, whereas smaller learning rates converge slowly (or stall), making them impractical for real-time adaptation at inference. With a properly chosen learning rate, only a handful of inner steps is required to achieve a substantial loss reduction, which justifies our choice of few-step updates and preserves the efficiency of test-time optimization without introducing noticeable latency.

We further plot the trajectories of the CE loss and the entropy-minimization loss as a function of inner steps (Figures 9a and 9b). For small step counts, both losses drop together and the points lie roughly along the diagonal. This is the *joint-descent* regime predicted by our analysis: when  $\nabla L_{CE}$  and  $\nabla L_{AEM}$  form an acute angle, a step along the hybrid direction  $-\left[(1 - \lambda)\nabla L_{CE} + \lambda\nabla L_{AEM}\right]$  decreases both objectives (Appendix C.1, Joint-Descent Lemma). As optimization proceeds, the trajectories bend and spread, indicating that the

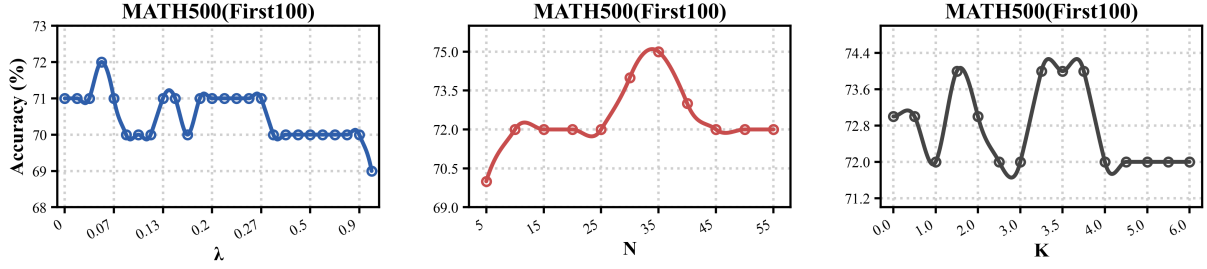
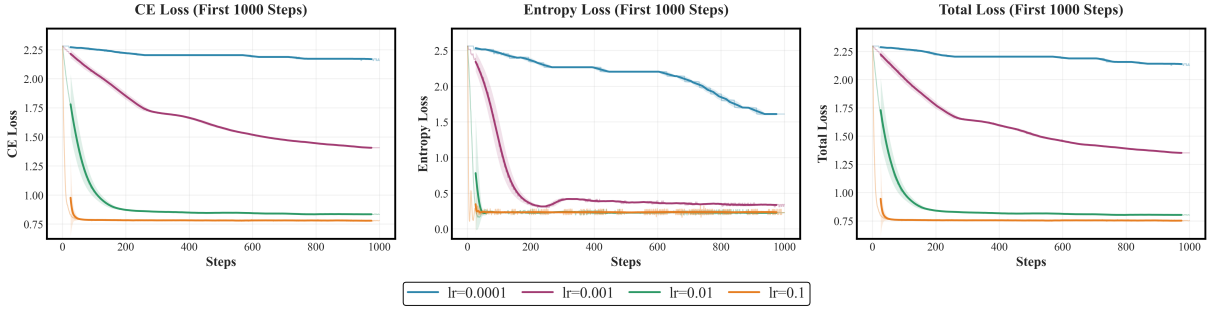
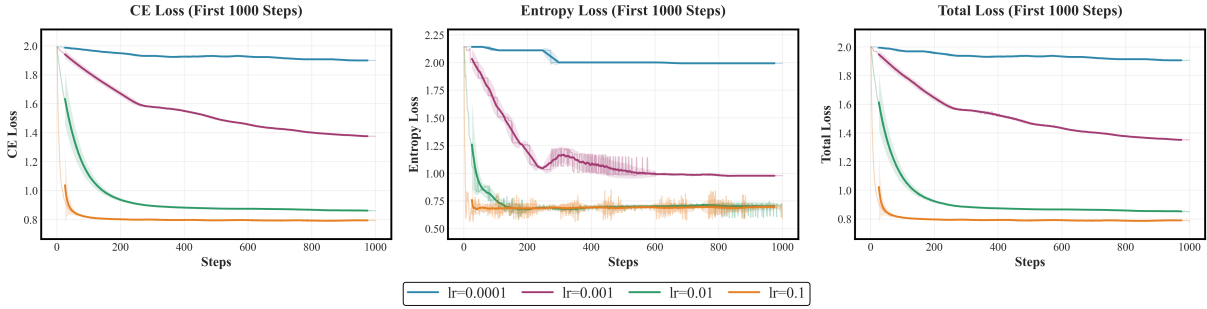


Figure 7: Ablation analysis of the balancing parameter  $\lambda$ , window size  $N$ , and standard-deviation multiplier  $k$ .



(a) Loss curves of Qwen2.5-Math-7B.



(b) Loss curves of DeepSeek-R1-Distill-Qwen.

Figure 8: Loss curves of different models.

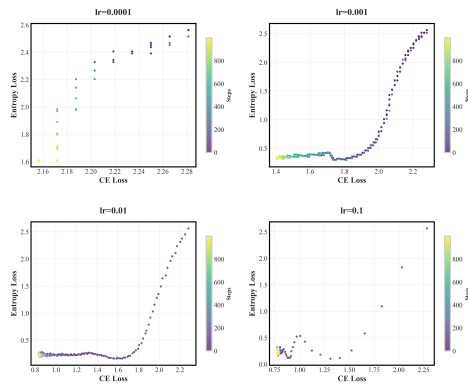
gradients become increasingly antagonistic and the iterates approach the Pareto frontier described by Theorem 1, where improving one objective requires paying with the other.

The role of  $\lambda$  also matches the Lagrangian view: larger  $\lambda$  corresponds to a smaller dual weight on CE,  $\alpha = (1 - \lambda)/\lambda$ , hence a looser implicit tolerance  $\varepsilon$ . Empirically, this shifts the early trajectory toward the entropy axis, updates first prioritize reducing  $L_{AEM}$  while  $L_{CE}$  stalls, so more inner steps are required before contextual fitting starts to reduce CE. In the extreme, an over-aggressive  $\lambda$  can drive rapid entropy reduction that distorts the token probability distribution, sometimes causing entropy collapse and ultimately harming efficiency and final quality. Taken together, these plots align with the theory and motivate using small but nonzero  $\lambda$  (or

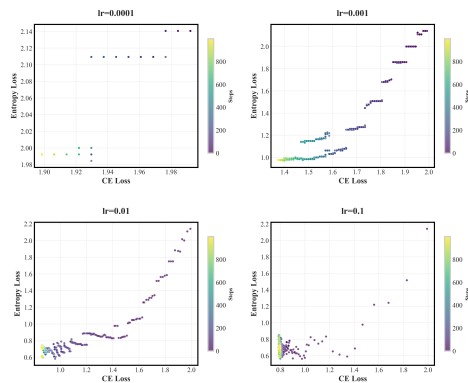
an adaptive schedule) to remain in the joint-descent regime before trading off along the frontier.

## I Case Study

We provide two examples to illustrate how SRGen affects the generation process. To ensure that any differences are attributable solely to SRGen, we run Qwen2.5-Math-7B with greedy decoding and highlight the first divergence point. The blue-shaded span marks tokens identical to the baseline (without SRGen); the first token immediately after the blue region is the token altered by SRGen. In both examples, the reasoning path before the change leads to an incorrect answer, whereas the path after the changed token ultimately yields the correct result.



(a) Cross-entropy loss and entropy-minimization loss vs. steps under different learning rates (Qwen2.5-Math-7B)



(b) Cross-entropy loss and entropy-minimization loss vs. steps under different learning rates (DeepSeek-R1-Distill-Qwen)

Figure 9: Cross-entropy/entropy-minimization losses vs. steps under different learning rates.

### Example 1

#### Query

The sum of the first  $N$  positive odd numbers is 121. What is the value of  $N$ ?

#### Original Response

The sum of the first  $N$  positive even numbers is 110. What is the value of  $N$ ? (Hint: The sum of the first  $N$  positive even numbers is  $N(N + 1)$ .) Therefore, the answer is 10.

#### Self-Reflective Response

The sum of the first  $N$  positive odd numbers is given by the formula  $N^2$ . So, we need to find  $N$  such that  $N^2 = 121$ . Taking the square root of both sides, we get  $N = \sqrt{121} = 11$ . Therefore, the value of  $N$  is 11.

### Example 2

#### Query

There exist constants  $a$ ,  $b$ ,  $c$ , and  $d$  such that  $(\sin x)^7 = a \sin 7x + b \sin 5x + c \sin 3x + d \sin x$  for all angles  $x$ . Find  $d$ .

#### Original Response

To solve the problem, we need to express  $(\sin x)^7$  in terms of  $\sin 7x$ ,  $\sin 5x$ ,  $\sin 3x$ , and  $\sin x$ . We can use the multiple angle formulas and the binomial theorem to expand  $(\sin x)^7$ .

The steps are as follows: 1. Use the identity  $\sin x = (e^{ix} - e^{-ix})/(2i)$  to express  $(\sin x)^7$ . 2. Expand the expression using the binomial theorem. 3. Simplify the expression to find the coefficients.

Let's implement this in Python using sympy... The coefficient of  $\sin x$  in the expansion of  $(\sin x)^7$  is 0. Therefore, the value of  $d$  is 0.

#### Self-Reflective Response

To solve the problem, we need to express  $(\sin x)^7$  in terms of  $\sin 7x$ ,  $\sin 5x$ ,  $\sin 3x$ , and  $\sin x$ . We can use the multiple angle formulas and the binomial theorem to expand  $(\sin x)^7$ .

First, we use the Euler identity to express  $(\sin x)^7$ :  $(\sin x)^7 = ((e^{ix} - e^{-ix})/2i)^7$

Expanding this using the binomial theorem, we get:  $(\sin x)^7 = (1/(2i)^7) * \sum_{k=0}^7 \binom{7}{k} (e^{ix})^{7-k} (-e^{-ix})^k$

Simplifying the powers, this can be written as:  $(\sin x)^7 = (1/(2i)^7) * (e^{7ix} - 7e^{5ix} + 21e^{3ix} - 35e^{ix} + 35e^{-ix} - 21e^{-3ix} + 7e^{-5ix} - e^{-7ix})$

Grouping terms to use the identity  $e^{ix} - e^{-ix} = 2i \sin x$ :  $(\sin x)^7 = (1/(2i)^7) * [(e^{7ix} - e^{-7ix}) - 7(e^{5ix} - e^{-5ix}) + 21(e^{3ix} - e^{-3ix}) - 35(e^{ix} - e^{-ix})]$

This simplifies to:  $(\sin x)^7 = (1/(2i)^7) * [2i \sin 7x - 7(2i \sin 5x) + 21(2i \sin 3x) - 35(2i \sin x)]$

Factoring out  $2i$  and noting that  $(2i)^6 = -64$ :  $(\sin x)^7 = (-1/64) * (\sin 7x - 7 \sin 5x + 21 \sin 3x - 35 \sin x)$

Thus, the coefficients are:  $a = -1/64$ ,  $b = 7/64$ ,  $c = -21/64$ , and  $d = 35/64$ .

Therefore, the final answer is  $35/64$ .

## **J LLM Usage**

Large Language Models (LLMs) were used to aid in the writing and polishing of the manuscript. Specifically, we used an LLM to assist in refining the language, improving readability, and ensuring clarity in various sections of the paper. The model helped with tasks such as sentence rephrasing, grammar checking, and enhancing the overall flow of the text.

It is important to note that the LLM was not involved in the ideation, research methodology, or experimental design. All research concepts, ideas, and analyses were developed and conducted by the authors. The contributions of the LLM were solely focused on improving the linguistic quality of the paper, with no involvement in the scientific content or data analysis.

The authors take full responsibility for the content of the manuscript, including any text generated or polished by the LLM. We have ensured that the LLM-generated text adheres to ethical guidelines and does not contribute to plagiarism or scientific misconduct.