HMPO: Hybrid Median-length Policy Optimization for Chain-of-Thought Compression

Anonymous EMNLP submission

Abstract

Large language models achieve remarkable performance via extended chain-of-thought (CoT) reasoning, yet this lengthy process incurs substantial inference overhead. Existing CoT compression methods struggle with inflexible manual length budgets, computationally expensive multi-stage training pipelines, and fragile scalability restricted to small models. We propose **HMPO** (**H**ybrid **M**edian-length **P**olicy **O**ptimization), a cost-effective, single-stage reinforcement learning framework. HMPO efficiently compresses CoT via three synergistic components: an *adaptive median-based budget* derived from successful rollouts to eliminate manual tuning, a *cosine-decay token reward* for smooth length penalization, and a *multiplicative reward formulation* that substantially mitigates trivial reward hacking by strictly prioritizing answer correctness. Trained exclusively on mathematical data, HMPO generalizes seamlessly across math, code, science, and instruction-following tasks. Extensive experiments scaling from 9B to 122B parameters across dense and Mixture-of-Experts (MoE) architectures demonstrate that HMPO achieves **19%–46% token compression with negligible accuracy degradation**, all while drastically reducing training costs compared to existing multi-stage baselines.

1 Introduction

Recent large language models (LLMs) like OpenAI o1 (Jaech et al., 2024), DeepSeek-R1 (Guo et al., 2025), and Qwen-3 (Yang et al., 2025) achieve remarkable reasoning performance via extended chains of thought (CoT). While powerful across math, science, and coding tasks, this "thinking" paradigm incurs prohibitive costs: models routinely generate thousands of tokens per query, leading to high latency and resource consumption (Chen et al., 2025; Sui et al., 2025). This raises a pressing question: *can we compress the reasoning process without sacrificing accuracy?*

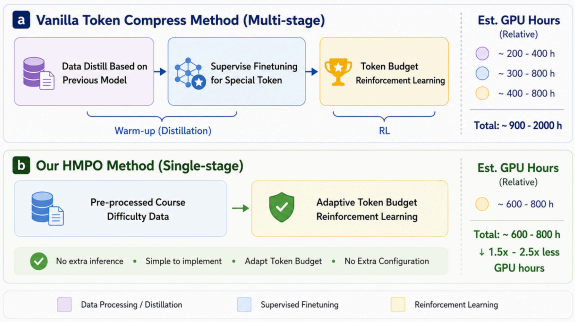

Figure 1: Comparison of training pipelines. Unlike vanilla multi-stage methods (a) that require data distillation and SFT warm-ups, our HMPO (b) achieves compression in a single RL stage, reducing GPU hours by 1.5×–2.5×.

Despite recent progress, existing CoT compression approaches share three critical limitations: *(1) Inflexible constraints and domain specificity:* Methods rely on manually tuned length budgets (Wu et al., 2026; Lyu et al., 2025) or rigid binary "think or skip" decisions (Zhang et al., 2025), limiting evaluations mostly to narrow mathematical domains. *(2) Prohibitive training costs:* Achieving effective compression typically requires computationally expensive pipelines, such as progressive multi-stage pruning (Hou et al., 2025), data distillation with Supervised Fine-Tuning (SFT) warm-ups (Fang et al., 2026), or massive pre-sampling (Luo et al., 2025). *(3) Fragile scalability:* Current evaluations are restricted to small dense models ($\leq$ 32B) (Fan et al., 2026), leaving applicability to state-of-the-art large-scale and Mixture-of-Experts (MoE) architectures unverified.

To address these limitations, we propose **HMPO** (**H**ybrid **M**edian-length **P**olicy **O**ptimization), a cost-effective, single-stage reinforcement learning framework (**Figure 1**). Built upon Group Relative Policy Optimization (GRPO) (Shao et al., 2024), HMPO elegantly compresses CoT through three synergistic components: (1) an *adaptive median-*

based budget that dynamically targets the median length of *correct* rollouts, implicitly forming a self-tightening curriculum without manual tuning; (2) a *cosine-decay token reward* applied exclusively to successful trajectories, providing a "soft landing" penalty that compresses lengths while preserving necessary reasoning steps; and (3) a *multiplicative reward formulation* ($R = R_{\text{acc}} \cdot R_{\text{token}}$) that enforces a strict "correctness-first, length-second" hierarchy, mathematically inoculating the training against reward hacking by denying efficiency gradients to incorrect responses.

As illustrated in Figure 1, by eliminating distillation and SFT phases, HMPO reduces training GPU hours by 1.5×–2.5× over multi-stage baselines. Crucially, it unlocks massive inference token savings. Trained solely on math data, HMPO generalizes seamlessly across math, code, science, and instruction-following tasks. Scaling from 9B to 122B parameters across dense and MoE architectures, it achieves **19%–46% token compression with negligible accuracy degradation**. Our main contributions are:

- **Methodological Innovation:** HMPO introduces adaptive correct-median budgeting, cosine reward shaping, and multiplicative reward factorization, jointly resolving the budget rigidity, penalty aggressiveness, and reward hacking vulnerabilities of prior RL-based compression methods.

- **Cross-domain Generalization:** We present the most extensive CoT compression evaluation to date, validating HMPO across 9B to 122B dense and MoE models, demonstrating robust cross-domain generalization despite math-only training.

- **Unprecedented Scalability to 100B+ Models:** We present the first successful CoT compression on massive MoE architectures. On the state-of-the-art 122B model, HMPO achieves highly compact reasoning while maintaining near-perfect accuracy parity. This enables our models to perform competitively against substantially larger models—matching or outperforming frontier models of much larger activated scales (e.g., 700B+ MoEs) with drastically fewer tokens.

2 Related Work

Efficient Chain-of-Thought Reasoning. While Chain-of-Thought (CoT) (Wei et al., 2022) equips reasoning models like OpenAI o1 (Jaech et al., 2024) and DeepSeek-R1 (Guo et al., 2025) with advanced capabilities, it often induces severe inference latency via "overthinking" (Chen et al., 2025; Sui et al., 2025). To mitigate this, recent efficient reasoning approaches explore supervised fine-tuning on compressed data (Munkhbat et al., 2025; Kang et al., 2025; Xia et al., 2025), post-hoc pruning (Zeng et al., 2025), and length-aware reinforcement learning (Aggarwal and Welleck, 2025; Yeo et al., 2025).

CoT Compression via RL and Adaptive Reasoning. The most closely related works fall into length-aware RL, iterative pruning, and adaptive reasoning. Length-aware RL methods (Aggarwal and Welleck, 2025; Lyu et al., 2025; Han et al., 2025; Arora and Zanette, 2026) incorporate explicit length penalties but strictly rely on manually predefined or statically estimated token budgets. Iterative pruning methods (Hou et al., 2025; Luo et al., 2025) and adaptive "think-or-skip" frameworks such as AdaptThink (Zhang et al., 2025) and Thinkless (Fang et al., 2026) conditionally reduce CoT length but necessitate computationally prohibitive multi-stage pipelines or data distillation. In contrast, HMPO seamlessly integrates fine-grained length shaping into a *single RL stage*, entirely replacing rigid manual constraints with an *on-policy adaptive median budget*.

3 Method

As illustrated in Figure 2, our proposed HMPO builds upon GRPO (Shao et al., 2024) by dynamically deriving a group-adaptive budget b from the median length of correct rollouts and applying a length-aware token reward.

3.1 Preliminaries

For each query $(q, a) \sim \mathcal{D}$, the policy π_θ generates a response $o = (o_1, \ldots, o_{|o|})$ of $|o|$ tokens. Following GRPO, we sample G rollouts $\{o_i\}_{i=1}^{G}$ from the old policy $\pi_{\theta_{\text{old}}}(\cdot \mid q)$ to optimize the clipped surrogate objective:

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{(q,a)\sim\mathcal{D}}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\ell_{i,t}(\theta)\right] \tag{1}$$

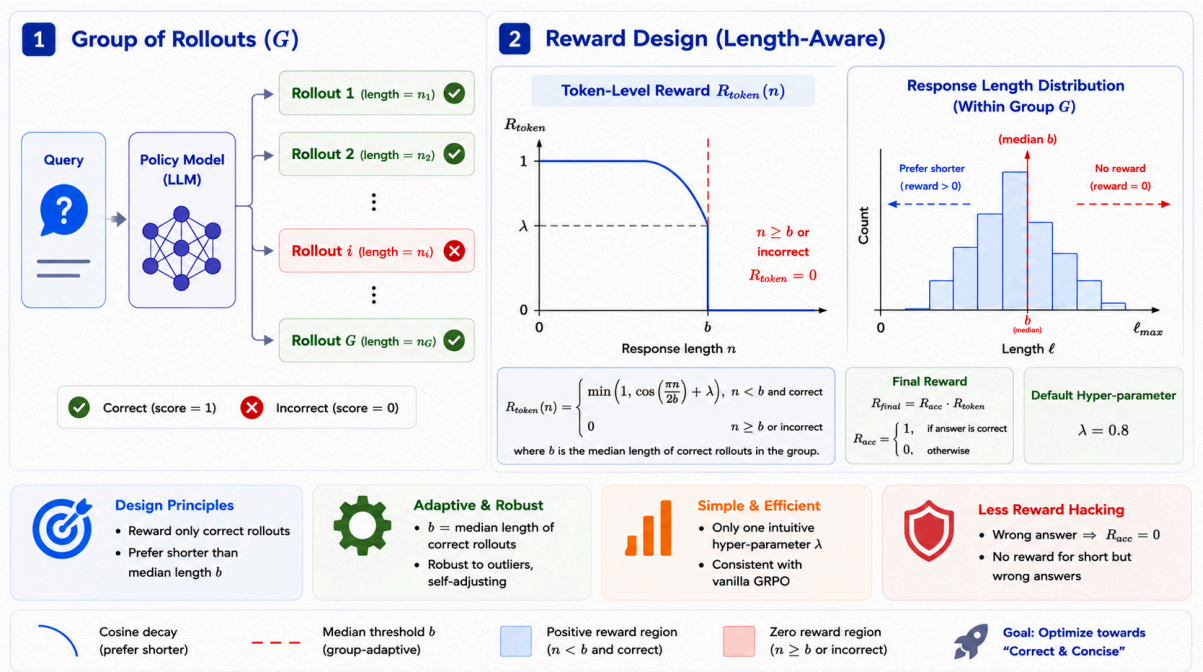

Figure 2: Overview of HMPO. **Left:** For each query, the policy samples a group of rollouts (G). **Right:** Instead of relying on a static threshold, HMPO dynamically derives an adaptive budget b from the median length of only the *correct* rollouts to construct a smooth cosine-decay token reward. **Bottom:** The final reward is combined multiplicatively to enforce a strict "correctness-first, length-second" objective, mathematically preventing reward hacking (i.e., short but incorrect answers strictly receive zero reward).

where the token-level importance ratio $s_{i,t}(\theta)$ is defined as:

$$s_{i,t}(\theta) = \frac{\pi_\theta(o_{i,t} \mid q, o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t} \mid q, o_{i,<t})} \tag{2}$$

and the token-level surrogate loss $\ell_{i,t}(\theta)$ is split across lines to fit the column width:

$$\ell_{i,t}(\theta) = \min\Big(s_{i,t}(\theta)A_i, \\ \text{clip}\big(s_{i,t}(\theta), 1-\epsilon, 1+\epsilon\big)A_i\Big) \tag{3}$$

The group-relative advantage A_i for each rollout o_i is computed as:

$$A_i = \frac{R_i - \text{mean}(\{R_j\}_{j=1}^G)}{\text{std}(\{R_j\}_{j=1}^G) + \varepsilon} \tag{4}$$

where R_i denotes the rollout-level reward for sample o_i, and ε is a small constant added for numerical stability.

Standard GRPO rewards focus on answer correctness. To mitigate the high inference costs of lengthy CoT, we must additionally optimize for reasoning efficiency. However, penalizing verbosity without degrading accuracy requires establishing an appropriate token budget. Therefore, our core modification to GRPO lies in redesigning the rollout reward R_i to jointly enforce this budget-aware efficiency and answer correctness. where R_i is the rollout-level reward for sample o_i.

3.2 Adaptive Budget Estimation

In RL-based CoT compression, efficiency is typically achieved by incorporating length-penalizing rewards, which fundamentally rely on defining a target token budget (Lyu et al., 2025; Hou et al., 2025; Wu et al., 2026). However, establishing an appropriate budget remains a critical challenge. Existing paradigms typically resort to either a dataset-wide static budget, which fails to accommodate varying problem complexities, or prior rollout-based estimation, which incurs prohibitive computational overhead and tightly couples the algorithm to pre-computed data.

To overcome these bottlenecks, HMPO introduces an **adaptive median-length budget** calculated dynamically from on-policy valid reasoning traces. Let $\{o_i\}_{i=1}^G$ be the current rollout group with respective token lengths $n_i = |o_i|$, and let $\mathcal{C} \subseteq \{1, \ldots, G\}$ denote the indices of rollouts that yield the correct answer. The group-level budget b is formally defined as the median length of these

successful attempts:

$$b = \text{median}(\{n_i \mid i \in \mathcal{C}\}) \tag{5}$$

This formulation is mathematically simple yet highly effective. Statistically, computing the median exclusively on correct rollouts isolates the length signal from noisy, failed reasoning attempts and provides strong robustness against the heavy-tailed distribution of LLM generation. Dynamically, it perfectly decouples the budget from data preprocessing while enabling a difficulty-aware constraint: complex queries intrinsically produce longer correct rollouts, naturally relaxing the budget, whereas simpler queries are subjected to tighter constraints. Furthermore, as the policy learns to reason more efficiently over time, b automatically decreases, implicitly forming a self-tightening curriculum without manual intervention.

3.3 Token-aware Reward Design

With the dynamic budget b in place, the remaining question is how to incorporate it into the optimization objective. A naive length penalty is prone to reward hacking, where the model produces overly short but incorrect responses to exploit the length signal. To prevent this, we factorize the overall reward into an accuracy term and a token-efficiency term, enforcing a strict *correctness-first, length-second* hierarchy.

Accuracy reward. R_{acc} measures whether the final prediction is correct. In our experiments, it is assigned as 1 for correct and 0 for incorrect.

Token reward. To steer the policy toward "correct-and-concise" behavior with minimal deviation from standard on-policy updates, we introduce a length preference *exclusively* for successful trajectories. Specifically, we define R_{token} to reinforce correct rollouts whose length n is below the adaptive budget b, applying a smooth cosine decay:

$$R_{\text{token}} = \begin{cases} \min\left(1,\ \cos\left(\frac{\pi n}{2b}\right) + \lambda\right), & \text{if correct and } n < b, \\ 0, & \text{otherwise,} \end{cases} \tag{6}$$

where the reward is applied only to correct trajectories with $n > 0$; all incorrect or empty ($n = 0$) responses receive $R_{\text{token}} = 0$. The scalar λ controls the magnitude of the length-induced gain. The cosine term guarantees a smooth, bounded preference: shorter correct responses receive higher rewards, with the signal decaying gently as n approaches b. The $\min(1, \cdot)$ truncation ensures the reward does not over-dominate the accuracy signal.

Final reward. We combine the accuracy and token efficiency signals multiplicatively:

$$R_{\text{final}} = R_{\text{acc}} \cdot R_{\text{token}} \tag{7}$$

Consider four trajectory categories: *correct-short*, *correct-long*, *incorrect-short*, *incorrect-long*. Under this formulation, only correct-and-concise trajectories receive high reward; incorrect and correct-but-beyond-budget ones are zeroed out ($R_{\text{final}} = 0$). In contrast, additive combination ($R_{\text{acc}} + R_{\text{token}} \in [0, 2]$) still grants correct-but-long trajectories a base reward of $R_{\text{acc}} = 1$ regardless of verbosity, making the optimization signal less direct. The multiplicative form provides a sharper target and reduces reward misalignment—this motivates our default choice.

3.4 Training Procedure

Algorithm 1 summarizes the overall HMPO training loop. For each input query, we sample a group of responses from the old policy. Crucially, we first compute the accuracy reward for each rollout to identify the correct subset $\mathcal{C}$. We then estimate the adaptive budget b using the median length of these valid traces. Finally, we calculate R_{token} and the multiplicative R_{final} to construct normalized advantages, which are subsequently plugged into the GRPO objective for policy optimization.

4 Experiments

4.1 Experimental Setup

Models, Datasets, and Baselines. To evaluate scalability across architectures and model sizes, we utilize Qwen3.5-9B (Qwen Team, 2026) as our dense model, alongside Qwen3.5-35B-A3B (Qwen Team, 2026) and Qwen3.5-122B-A10B (Qwen Team, 2026) as Mixture-of-Experts (MoE) models. We train on 6.5K challenging instances (levels 5–9) sampled from DeepMath-103K (He et al., 2025). We compare HMPO against the original Base Models and three CoT compression baselines: **AdaptThink** (Zhang et al., 2025) (constrained binary mode selection), **Thinkless** (Fang et al., 2026) (two-stage SFT and decoupled RL), and **ThinkPrune** (Hou et al., 2025) (multi-round iterative length pruning). To ensure fair comparisons, all baselines are rigorously reproduced using their original training pipelines and hyperparameters.Detailed reproduction procedures for all

Algorithm 1 HMPO Training

Require: Dataset $\mathcal{D}$, policy π_θ, old policy $\pi_{\theta_{\text{old}}}$, group size G, penalty coefficient λ

while not converged **do**
  Sample a batch of queries $\{q\} \sim \mathcal{D}$
  for each query q **do**
    Sample G rollouts $\{o_i\}_{i=1}^{G} \sim \pi_{\theta_{\text{old}}}(\cdot \mid q)$
    for $i = 1$ to G **do**
      Compute length $n_i = |o_i|$ and accuracy reward $R_{\text{acc}}^{(i)}$
    end for
    Identify correct subset $\mathcal{C} = \{i \mid R_{\text{acc}}^{(i)} > 0\}$
    if $|\mathcal{C}| > 0$ **then**
      Estimate adaptive budget $b \leftarrow \text{median}(\{n_i \mid i \in \mathcal{C}\})$
    else
      $b \leftarrow \infty$ {Fallback; R_{final} will be 0 regardless}
    end if
    for $i = 1$ to G **do**
      Compute token reward $R_{\text{token}}^{(i)}$ using Eq. 6
      Compute final reward $R_i \leftarrow R_{\text{acc}}^{(i)} \cdot R_{\text{token}}^{(i)}$
    end for
    Normalize $\{R_i\}_{i=1}^{G}$ into advantages $\{A_i\}_{i=1}^{G}$
  end for
  Update π_θ using the GRPO objective in Eq. 1
end while

baselines are provided in Appendix B. All artifacts (models, datasets, and codebases) utilized in this work are publicly available and employed in strict compliance with their open-source licenses (e.g., Apache 2.0, MIT). We will release our code, training scripts, and configurations upon acceptance.

Evaluation Metrics. We evaluate cross-domain performance on five benchmarks: AIME 2025 (Zhang and Math-AI, 2025) and AIME 2026 (Zhang and Math-AI, 2026) (math), GPQA-Diamond (Rein et al., 2023) (science), IFEval (Zhou et al., 2023) (instruction following), and LiveCodeBench-V6 (Jain et al., 2025) (code). For each benchmark, we report accuracy (Acc), average token length (Len), and per-benchmark compression ratio (ρ). We also report the overall cross-benchmark compression ratio $\bar{\rho}$, defined as:

$$\bar{\rho} = \frac{\sum_{i=1}^{5} \text{Len}_{\text{base}}^{(i)} - \sum_{i=1}^{5} \text{Len}_{\text{method}}^{(i)}}{\sum_{i=1}^{5} \text{Len}_{\text{base}}^{(i)}}. \quad (8)$$

To maximally preserve the models' inherent capabilities and ensure robust performance under token compression, inference is conducted via SGLang (Zheng et al., 2024) using official recommended settings (temperature 1.0, top-p=0.95, top-k=20) and a full 256K maximum context limit. To reduce variance, AIME results are averaged over 16 sampled responses per problem, and other benchmarks are evaluated twice.

Implementation Details. We implement HMPO using the VERL framework (Sheng et al., 2025), sampling G=10 rollouts per prompt with λ=0.8. Training converges within 40–60 steps, reaching the optimal accuracy-compression trade-off around steps 16–24. Full hyperparameters and infrastructure details are deferred to Appendix A.

4.2 Main Results

Table 1 presents the overall comparison across three Qwen3.5-series models. Detailed metric layouts are described in the table caption. Notably, due to prohibitive computational costs, we omit baseline reproductions for the 122B configuration.

HMPO achieves the best accuracy-compression trade-off. HMPO consistently delivers substantial compression while preserving Base Model accuracy. On Qwen3.5-9B, HMPO nearly halves output tokens ($\bar{\rho}$=46%) with a negligible 0.02 percentage points(pp) average accuracy drop. On Qwen3.5-35B-A3B, it compresses 42% while maintaining 85.52% average accuracy, even improving IFEval (94.09% vs. 92.79%). In contrast, ThinkPrune achieves higher compression (64–72%) but suffers severe accuracy degradation (11–22pp AIME drops). Thinkless exhibits catastrophic failure on 9B (<15% accuracy), likely due to SFT-induced mode collapse during the subsequent RL phase. AdaptThink provides marginal gains but surprisingly inflates 35B IFEval lengths by 33%. Thus, HMPO uniquely occupies the Pareto frontier where both accuracy and efficiency are jointly optimized.

Consistent in-domain retention and out-of-domain generalization. Trained solely on math data, HMPO transfers its concise reasoning patterns seamlessly. On in-domain **AIME 2025/2026**, it secures 23–37% compression, even boosting 122B accuracy on AIME 2025 (89.58%→90.21%). Crucially, on *out-of-domain* tasks, HMPO achieves 16–61% compression with accuracy parity on **LiveCodeBench(LCB)-V6** (e.g., exactly matching the 35B Base Model's 73.66% while cutting outputs by 49%). It further yields 17–34% compression on **GPQA-Diamond** without accuracy loss, and 16–61% reduction on **IFEval** while maintaining or improving accuracy. This proves HMPO generalizes robustly without task-specific tuning.Compression case studies are presented in Appendix C.

Table 1: Main results on three Qwen3.5-series model configurations. Each method occupies two rows: accuracy (Acc, %) and output token length (Len). Green/red annotations indicate improvement/degradation relative to the Base Model. The Avg column shows average accuracy and average length (with overall compression ratio $\bar{\rho}$). **Bold**: HMPO results in the Avg column. †: baselines not reproduced due to training cost.

Method		AIME'25	AIME'26	LCB-V6	GPQA	IFEval	Avg.
Qwen3.5-9B							
Base Model	Acc	85.21	89.90	65.60	81.57	91.82	82.82
	Len	24,947	24,871	49,227	10,586	5,692	23,065
AdaptThink	Acc	81.25	87.29	66.79	82.58	92.42	82.07 -0.75
	Len	19,797 -21%	20,548 -17%	35,577 -28%	8,366 -21%	4,468 -22%	17,751 -23%
Thinkless	Acc	13.75	8.33	8.78	1.26	20.24	10.47 -72.35
	Len	560 -98%	604 -98%	604 -99%	548 -95%	1,239 -78%	711 -97%
ThinkPrune	Acc	73.75	77.71	56.11	80.81	89.65	75.61 -7.21
	Len	12,171 -51%	10,586 -57%	12,592 -74%	4,484 -58%	2,071 -64%	8,381 -64%
HMPO	Acc	84.58	89.79	65.27	82.32	92.05	**82.80 -0.02**
	Len	17,087 -32%	15,564 -37%	19,283 -61%	6,934 -34%	3,366 -41%	**12,447 -46%**
Qwen3.5-35B-A3B							
Base Model	Acc	89.58	91.87	73.66	83.08	92.79	86.20
	Len	18,900	19,920	31,024	7,107	7,365	16,863
AdaptThink	Acc	86.67	86.25	68.70	84.09	92.33	83.61 -2.59
	Len	12,897 -32%	13,470 -32%	25,851 -17%	6,708 -6%	9,761 +33%	13,737 -19%
Thinkless	Acc	60.42	67.92	52.29	77.27	69.13	65.41 -20.79
	Len	7,672 -59%	7,433 -63%	9,343 -70%	4,345 -39%	2,713 -63%	6,301 -63%
ThinkPrune	Acc	67.71	78.54	60.31	81.57	92.98	76.22 -9.98
	Len	5,798 -69%	5,295 -73%	7,815 -75%	2,806 -61%	1,664 -77%	4,676 -72%
HMPO	Acc	86.46	90.83	73.66	82.58	94.09	**85.52 -0.68**
	Len	12,454 -34%	12,457 -37%	15,762 -49%	4,981 -30%	2,836 -61%	**9,698 -42%**
Qwen3.5-122B-A10B†							
Base Model	Acc	89.58	92.71	78.90	86.60	93.40	88.24
	Len	17,666	17,072	23,774	6,871	4,016	13,880
HMPO	Acc	90.21	92.50	79.01	85.61	93.35	**88.14 -0.10**
	Len	13,675 -23%	13,136 -23%	20,077 -16%	5,671 -17%	3,377 -16%	**11,187 -19%**

Robustness across model scales and architectures. HMPO remains effective scaling from 9B dense to 122B MoE models. On the 122B model, it yields 19% compression with a mere 0.10pp average accuracy degradation. Crucially, HMPO requires *no hyperparameter tuning*: the identical λ=0.8 setting works across all scales. By automatically calibrating compression pressure via the adaptive median budget, HMPO serves as a highly practical, drop-in optimization for deployment.

Token efficiency vs. frontier models. To contextualize HMPO's practical value, Figure 3 plots accuracy versus average tokens per trajectory on AIME 2026. First, compared to its unoptimized base model, HMPO-Qwen3.5-122B-A10B slashes average token consumption by 23% (17.1K $\rightarrow$ 13.1K) with a negligible 0.21% accuracy variance (92.71% vs. 92.50%). Beyond self-improvement, HMPO fundamentally reshapes the Pareto frontier against substantially larger frontier models. It decisively outperforms Nemotron-3-Super-120B-A12B (89.90% at 13.4K) and delivers performance highly competitive with massive models like Kimi-K2.5-1T-A32B (93.30% at 16.6K). This demonstrates that HMPO enables reasoning models to punch significantly above their weight class, delivering top-tier mathematical performance at a fraction of the frontier inference cost.

4.3 Ablation Study

We ablate HMPO's components on Qwen3.5-9B (Table 2, Figure 4). *Note on Validation:* We monitor dynamics on AIME 2024. Since early-step outputs often exceed the 24K limit and truncate, initial accuracy is artificially deflated. As compression progresses, truncation ceases, revealing genuine accuracy before over-compression causes

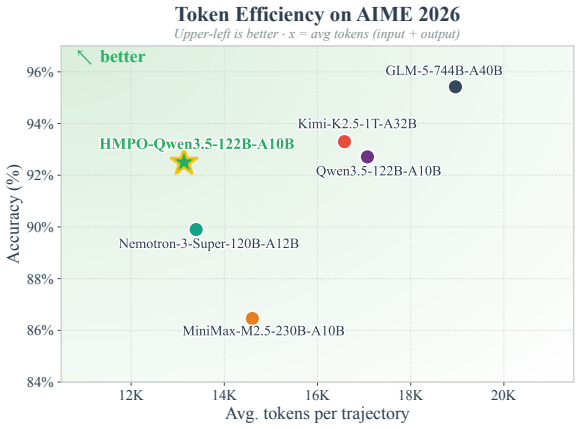

Figure 3: Token efficiency on AIME 2026. Each point represents a model's accuracy vs. average tokens per trajectory (input + output). Upper-left is better. HMPO-trained models (stars) achieve competitive or superior accuracy with significantly fewer tokens than frontier models of much larger activated scale.

degradation. Thus, we emphasize relative convergence trends over absolute peaks.

Table 2: Ablation study on Qwen3.5-9B, evaluated on all five benchmarks. We vary one component at a time from the default. All variants at step 16 except fixed-budget (step 40, due to slower convergence).

Variant	Avg Acc (%)	$\bar{\rho}$ (%)
Default (λ=0.8, med., mult.)	82.80	46.0
Compression strength		
λ=0.4, med., mult.	79.79	41.6
λ=1.0, med., mult.	81.15	42.5
Budget strategy		
λ=1.0, fixed 12K, mult.	82.12	12.4
Reward composition		
λ=0.8, med., additive	80.83	41.7

Adaptive median budget vs. fixed budget. We compare our adaptive median against a meticulously calibrated fixed budget (b=12K, set at 75% of the Base Model's ∼16K mean training-time response length, with λ=1.0). As shown in Figure 4(a), the fixed budget fails to drive meaningful compression (stalling at ∼14K), as many responses already satisfy the threshold and receive no optimization signal. Conversely, the adaptive median dynamically self-calibrates to the batch's length distribution, forming a robust self-tightening curriculum that continuously compresses outputs (16K → 4K). This avoids the sluggish convergence or premature collapse inherent to static manual tuning.

Multiplicative vs. additive reward. Under identical settings (λ=0.8, median budget), both multiplicative and additive formulations achieve comparable compression trajectories (Figure 4(a) and (b)). The accuracy curves (Figure 4(c)) also follow similar trends, though the multiplicative variant exhibits slightly less oscillation and achieves modestly higher final performance (82.80% vs. 80.83% in Table 2). We attribute this to the tighter objective coupling: the multiplicative form exclusively rewards the desired *correct-and-concise* category, whereas additive combination permits correct-but-verbose trajectories to accumulate positive reward, introducing noise into the optimization landscape.

Effect of λ. We compare $\lambda \in \{0.4, 0.8, 1.0\}$ (all with median budget, multiplicative reward). As shown in Figure 4(a), all three produce nearly identical compression in the first 10 steps, after which higher λ drives faster length reduction. In accuracy (Figure 4(c)), λ=0.8 peaks earliest (step 20, ∼75%) while λ=1.0 and λ=0.4 peak later (step 24 and step 32, at ∼70–72%) with more oscillation. The slightly lower peak of λ=0.4 reflects its weaker compression pressure, causing the model to converge more slowly; λ=1.0 compresses aggressively but sacrifices some accuracy headroom. We adopt λ=0.8 as a balanced default that achieves the highest accuracy at a moderate compression rate.

4.4 Analysis

Training dynamics and checkpoint selection. We use AIME 2024 to monitor training progress. As noted in §4.3, early-step validation accuracy is artificially deflated by 24K training truncation. However, we empirically find that the validation compression ratio reliably predicts downstream test compression. To further characterize this trade-off, we evaluate every 4th checkpoint of HMPO on Qwen3.5-9B using full-length inference on two representative benchmarks. As shown in Figure 5, several key patterns emerge: **(1) Graceful degradation with a clear inflection point:** Accuracy on AIME2025 (red solid line) remains stable through step 16 (85.21%→84.58%), then drops sharply beyond step 20 as essential reasoning steps are excessively pruned. **(2) Disproportionate compression on verbose tasks:** LiveCodeBench output length (blue dashed line) plummets from 49K to 19K tokens (61% compression) by step 16 with negligible accuracy loss (65.60% → 65.27%), suggesting substantial redundancy in the Base Model's code

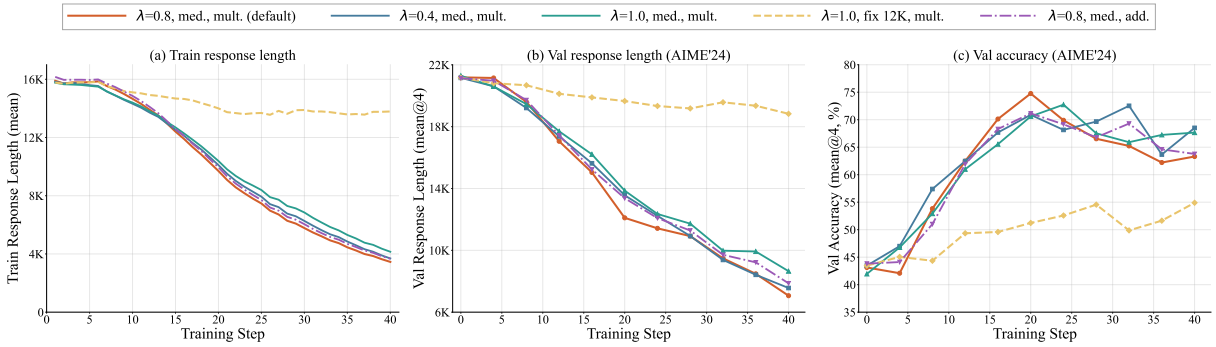

Figure 4: Ablation training dynamics on Qwen3.5-9B. Five configurations are compared: varying λ (0.4, 0.8, 1.0), budget strategy (median vs. fixed 12K), and reward composition (multiplicative vs. additive). The default (λ=0.8, median, mult.) achieves the best accuracy-compression trade-off.

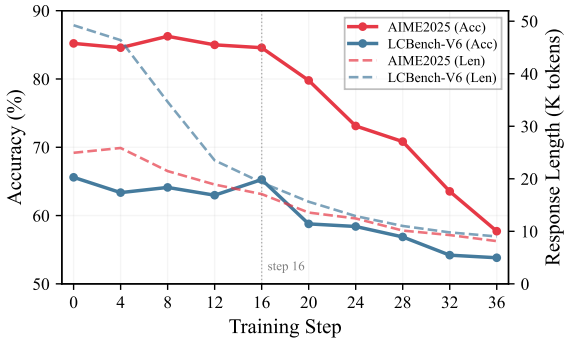

Figure 5: Training dynamics of HMPO (adaptive median) on Qwen3.5-9B. Solid lines (left): accuracy on AIME2025 and LiveCodeBench-V6, stable until step 16. Dashed lines (right): response length drops monotonically (LiveCodeBench reduces 49K→19K by step 16). Dotted line: selected checkpoint.

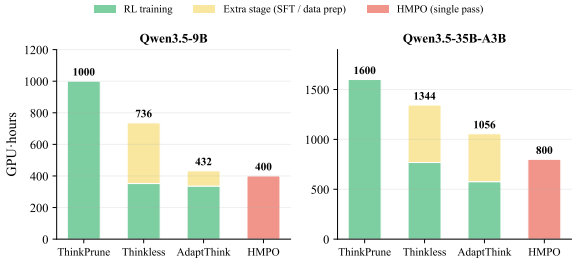

Figure 6: Training cost (GPU·hours) comparison. Stacked bars show main training (teal) and extra stages such as SFT or data preprocessing (gold). HMPO (orange) requires only a single-pass GRPO run with no additional stages.

reasoning. **(3) Monotonic length decrease:** Response lengths (dashed lines, right axis) decrease smoothly across both benchmarks, confirming that the adaptive median budget forms a stable compression curriculum without sudden collapses. These dynamics dictate our early-stopping strategy (typically around step 16–20), securing optimal compression before accuracy degrades.

Training cost comparison with baselines. Figure 6 compares total GPU·hours for each method. ThinkPrune requires 2.5× HMPO's cost on 9B (1000 vs. 400) due to multi-round iterative pruning. Thinkless incurs 1.8× overhead from its SFT distillation stage. AdaptThink has comparable RL cost but adds significant data preprocessing time (96–480 GPU·h). HMPO achieves competitive or superior compression in a single training pass with no auxiliary stages, making it the most cost-effective approach at both model scales.

5 Conclusion

In this paper, we introduced HMPO, a cost-effective, single-stage reinforcement learning framework designed to compress verbose Chain-of-Thought (CoT) reasoning. By synergizing a dynamic correct-median budget with a multiplicative cosine-decay reward, HMPO eliminates the need for tedious manual budget tuning and mathematically inoculates the training process against reward hacking. Extensive evaluations demonstrate that HMPO achieves near-optimal accuracy-compression trade-offs. Trained exclusively on mathematical data, the learned concise reasoning patterns generalize robustly across diverse out-of-domain tasks (code, science, instruction following) and scale seamlessly from 9B dense models up to 122B MoE architectures. By completely removing the reliance on computationally prohibitive multi-stage pipelines, SFT warm-ups, and task-specific tuning, HMPO provides a highly practical, drop-in optimization strategy for the efficient real-world deployment of advanced reasoning LLMs.

Limitations

While HMPO achieves consistent compression with negligible accuracy loss across multiple benchmarks and model scales, our study has several limitations that we leave for future work.

Multi-turn reasoning. Our evaluation focuses on *single-turn* reasoning settings, where the model produces one self-contained chain-of-thought per query. We do not study *multi-turn* scenarios such as interactive problem solving, follow-up clarification, or dialogue-based tutoring, in which reasoning unfolds progressively across turns and earlier thoughts may be reused or revised. "Concise reasoning" is more nuanced in such settings: the model must avoid overthinking within each turn while also avoiding redundant re-derivations across turns. Extending HMPO to multi-turn reasoning, e.g. by computing the adaptive budget over a turn-aware aggregation of rollouts, is a promising direction we leave for future work.

Agentic tasks. A related but distinct setting concerns agentic tasks, where the model interacts with external tools or environments through tool calls, observations, and intermediate state. We perform only a preliminary investigation in such settings, and the design of HMPO is not specifically tailored to them. In agentic settings, reasoning length must be balanced against tool-call planning, intermediate state tracking, and recovery from failed actions; a budget that is appropriate for self-contained reasoning may be too tight for plans that legitimately require longer deliberation. A systematic study of HMPO under agentic workflows, including budgets that are conditioned on tool usage or trajectory phase, remains an important direction.

References

Pranjal Aggarwal and Sean Welleck. 2025. L1: Controlling how long a reasoning model thinks with reinforcement learning. *arXiv preprint arXiv:2503.04697*.

Daman Arora and Andrea Zanette. 2026. Training language models to reason efficiently. *Advances in Neural Information Processing Systems*, 38:60770–60808.

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, and 1 others. 2025. Do not think that much for 2+ 3=? on the overthinking of long reasoning models. In *Forty-second International Conference on Machine Learning*.

Zhenxuan Fan, Jie Cao, Yang Dai, Zheqi Lv, Wenqiao Zhang, Zhongle Xie, Peng LU, and Beng Chin Ooi. 2026. Ctrlcot: Dual-granularity chain-of-thought compression for controllable reasoning. *arXiv preprint arXiv:2601.20467*.

Gongfan Fang, Xinyin Ma, and Xinchao Wang. 2026. Thinkless: Llm learns when to think. *Advances in neural information processing systems*, 38:151268–151295.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. 2025. Token-budget-aware llm reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 24842–24855.

Zhiwei He, Tian Liang, Jiahao Xu, Qiuzhi Liu, Xingyu Chen, Yue Wang, Linfeng Song, Dian Yu, Zhenwen Liang, Wenxuan Wang, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. 2025. Deepmath-103k: A large-scale, challenging, decontaminated, and verifiable mathematical dataset for advancing reasoning. *Preprint*, arXiv:2504.11456.

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

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, and 1 others. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.

Naman Jain, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2025. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, volume 2025, pages 58791–58831.

Yu Kang, Xianghui Sun, Liangyu Chen, and Wei Zou. 2025. C3ot: Generating shorter chain-of-thought without compromising effectiveness. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 24312–24320.

Haotian Luo, Li Shen, Haiying He, Yibo Wang, Shiwei Liu, Wei Li, Naiqiang Tan, Xiaochun Cao, and Dacheng Tao. 2025. O1-pruner: Length-harmonizing fine-tuning for o1-like reasoning pruning. *arXiv preprint arXiv:2501.12570*.

Shangke Lyu, Linjuan Wu, Yuchen Yan, Xingyu Wu, Hao Li, Yongliang Shen, Peisheng Jiang, Weiming

Lu, Jun Xiao, and Yueting Zhuang. 2025. Hierarchical budget policy optimization for adaptive reasoning. *arXiv preprint arXiv:2507.15844*.

Tergel Munkhbat, Namgyu Ho, Seo Hyun Kim, Yongjin Yang, Yujin Kim, and Se-Young Yun. 2025. Self-training elicits concise reasoning in large language models. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 25127–25152.

Qwen Team. 2026. Qwen3.5: Towards native multimodal agents.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. 2023. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2025. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pages 1279–1297.

Yang Sui, Yu-Neng Chuang, Guanchu Wang, Jiamu Zhang, Tianyi Zhang, Jiayi Yuan, Hongyi Liu, Andrew Wen, Shaochen Zhong, Na Zou, and 1 others. 2025. Stop overthinking: A survey on efficient reasoning for large language models. *arXiv preprint arXiv:2503.16419*.

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.

Taiqiang Wu, Zenan Xu, Bo Zhou, and Ngai Wong. 2026. The art of efficient reasoning: Data, reward, and optimization. *arXiv preprint arXiv:2602.20945*.

Heming Xia, Chak Tou Leong, Wenjie Wang, Yongqi Li, and Wenjie Li. 2025. Tokenskip: Controllable chain-of-thought compression in llms. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 3351–3363.

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*.

Edward Yeo, Yuxuan Tong, Morry Niu, Graham Neubig, and Xiang Yue. 2025. Demystifying long chain-of-thought reasoning in llms. *arXiv preprint arXiv:2502.03373*.

Wenhao Zeng, Yaoning Wang, Chao Hu, Yuling Shi, Chengcheng Wan, Hongyu Zhang, and Xiaodong Gu. 2025. Pruning the unsurprising: Efficient code reasoning via first-token surprisal. *arXiv preprint arXiv:2508.05988*.

Jiajie Zhang, Nianyi Lin, Lei Hou, Ling Feng, and Juanzi Li. 2025. AdaptThink: Reasoning models can learn when to think. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 3716–3730, Suzhou, China. Association for Computational Linguistics.

Yifan Zhang and Team Math-AI. 2025. American invitational mathematics examination (aime) 2025.

Yifan Zhang and Team Math-AI. 2026. American invitational mathematics examination (aime) 2026.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, and 1 others. 2024. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023. Instruction-following evaluation for large language models. *Preprint*, arXiv:2311.07911.

Appendix

A Training Hyperparameters

Table 3 lists the shared training configuration across all models. Table 4 details per-model parallelism settings.

B Baseline Reproduction Details

We provide detailed notes on our reproduction of each baseline method. All baselines are trained using Megatron (for SFT stages) and VERL (for RL stages) on their respective original training and validation datasets, strictly following each paper's prescribed training procedure and the hyperparameter configurations specified in their official open-source codebases.

AdaptThink. AdaptThink relies on pre-computed per-instance reference statistics from the target model itself. Because reasoning behavior (e.g.,chain-of-thought length and per-instance accuracy) varies substantially across base models of different sizes, these statistics must be re-computed whenever the base model is changed. Specifically, before training each target model, we sample $K=16$ rollout responses per training instance and compute per-instance mean accuracy, which serves

Table 3: Shared HMPO training hyperparameters.

Hyperparameter	Value
RL Algorithm	
Base algorithm	GRPO
Rollout group size G	10
Clip ratio ϵ (low / high)	0.2 / 0.2
Clip ratio c	10.0
KL coeff. (reward)	0.001
KL loss coeff.	0.001
KL loss type	low-var KL
Entropy coeff.	0
Loss aggregation	token-mean
Total epochs	1
HMPO Reward	
Budget strategy	adaptive median
λ	0.8
Composition	multiplicative
Optimizer	
Optimizer	AdamW
Learning rate	5×10^{-7}
Min LR	5×10^{-8}
Schedule	cosine decay
Warmup	5%
Data	
Training data	DeepMath-103K (lv.5–9)
Samples	6.5K
Validation	AIME'24 (30 problems.)
Max prompt	4,096 tok.
Checkpointing	
Save / val freq.	every 4 steps
Val samples	4 (do_sample)

Table 4: Per-model parallelism and batch settings.

	9B	35B	122B
Nodes×GPUs	2×8	4×8	8×8
TP	2	1	1
PP	4	8	8
CP	1	1	1
EP	1	4	4
ETP	1	1	1
Gen TP	2	8	8
Max resp.	24K	24K	24K
Batch	64	64	128
Mini-batch	16	32	64
Mem util.	0.6	0.6	0.6
Offload	✓	✓	✓
Precision	bf16	bf16	bf16
35B & 122B MoE settings:			
Attn. backend: auto			
Recompute: uniform, full, 1 layer			
MoE aux loss: 0.01; z-loss: 0.001			

as the reward baseline during RL to normalize advantages across problems of varying difficulty.

Another consequence of switching base models is that their problem-solving capacity within a fixed token budget differs: a smaller model may struggle to produce correct answers within a tight no-think budget, causing the no-think branch to yield near-zero accuracy and thus preventing the policy from learning meaningful think/no-think decisions. To mitigate this, we adjust the no-think length budget according to model capacity: we use 4K tokens for Qwen3.5-9B, and increase it to 16K tokens for Qwen3.5-35B-A3B. This calibration ensures that the no-think branch remains a viable competing trajectory during RL, so that the policy genuinely learns *when* to skip extended reasoning rather than being implicitly forced to think due to insufficient generation length.

Thinkless. Thinkless employs a two-stage pipeline: SFT distillation followed by Decoupled GRPO. We observe that the SFT stage—which trains the model on paired long/short responses with control tokens (`<think>` and `<short>`)—causes substantial degradation of the base model's reasoning capability, a limitation acknowledged in the original paper.

In the subsequent RL stage (trained for over 100 steps), this degradation manifests as severe mode collapse: on Qwen3.5-9B, the model converges almost entirely to short mode (`<short>`) within the first 10 steps, while on Qwen3.5-35B-A3B it collapses to think mode (`<think>`) within the same window. We extensively explored the Decoupled GRPO hyperparameters, including the control token update weight $\alpha \in \{0.001, 0.01, 0.5\}$ and the reward coefficient $\gamma \in \{0.5, 0.8\}$, but were unable to prevent the collapse under any configuration. This underscores a fundamental limitation of the approach: its reliance on high-quality SFT data and sensitivity to RL-stage hyperparameters make it fragile when applied to new base models.

ThinkPrune. ThinkPrune employs a multi-round iterative RL procedure that progressively tightens the maximum response length across stages (e.g., 4K→3K→2K in the original paper). A critical design decision is checkpoint selection: the original paper prescribes choosing the checkpoint with the shortest output length that maintains less than 10% accuracy degradation on the validation set (evaluated every 20 steps).

During reproduction, we encountered a practical challenge: when using the original paper's prescribed `max_response_length` values (4K, 3K, 2K), model outputs are almost always truncated at these limits, causing validation accuracy to collapse to near zero. This renders the checkpoint selection criterion inapplicable, as no checkpoint satisfies the $<10\%$ degradation threshold.

To resolve this, we decouple the training truncation from evaluation: we set `max_response_length` to 32K during validation (to allow full-length generation for accurate accuracy measurement), while retaining the target length budget as a truncation threshold

only during training rollouts. This engineering modification faithfully reproduces the intended training dynamics while enabling meaningful checkpoint selection.

Even with this fix, we find that ThinkPrune's stop criterion is difficult to satisfy precisely in practice—the method tends to over-compress before the accuracy degradation signal becomes detectable at the 20-step evaluation granularity, resulting in checkpoints that have already crossed the accuracy cliff. This observation indirectly validates the advantage of HMPO's continuous monitoring approach (§4.4), which enables finer-grained early stopping at the optimal accuracy-compression inflection point.

C Case Study

As illustrated in Figure 7 and Figure 8 We present qualitative examples comparing Base Model and HMPO outputs to illustrate how HMPO achieves token compression while preserving correctness. Both examples are drawn from Qwen3.5-9B.

C.1 AIME 2026: Mathematical Reasoning

Statistics.

- **Base Model**: thinking = 41,349 chars, response = 2,198 chars, total = 43,547 chars.
- **HMPO**: thinking = 7,483 chars, response = 1,560 chars, total = 9,043 chars.
- **Compression**: 79.2%. Both produce the correct answer $\boxed{178}$.

Analysis. The Base Model's reasoning trace exhibits several inefficiencies:

1. **Redundant verification of obvious constraints**: The Base Model spends ∼800 characters debating whether d must be an integer, considering fractional d values and ruling them out—a conclusion that follows trivially from "arithmetic sequences of integers."
2. **Repeated self-questioning**: The thinking trace revisits whether d can be negative multiple times ("Is there any possibility that d is negative and 24, 34 are terms before 4?"), each time reaching the same conclusion that $d > 0$.
3. **Over-elaboration of basic steps**: Computing $\gcd(20, 30) = 10$ and listing its divisors is expanded across ∼500 characters with intermediate prime factorization steps.

In contrast, HMPO's reasoning is structured as a concise step-by-step outline:

1. Identify constraints → formulate $(k_1-1)d = 20$, $(k_2-1)d = 30$.
2. Determine $d \mid \gcd(20, 30) = 10$, positive divisors $\{1, 2, 5, 10\}$.
3. Compute $T_{10} = 4 + 9d$ for each d, sum them.
4. Brief arithmetic verification: $13+22+49+94 = 178$.

HMPO eliminates the redundant self-debate while retaining all logically necessary steps. The final response is also more concise (1,560 vs. 2,198 chars) while containing identical mathematical content.

C.2 LiveCodeBench: Code Generation

Statistics.

- **Base Model**: thinking = 136,169 chars, response = 13,488 chars, total = 149,657 chars.
- **HMPO**: thinking = 44,109 chars, response = 3,595 chars, total = 47,704 chars.
- **Compression**: 68.1%. Both pass all test cases.

Analysis. Both models arrive at the same algorithmic solution: $O(N^2)$ DP with convex hull trick (CHT) optimization. However, their reasoning processes differ substantially:

1. **Problem reformulation**: Both correctly identify that the cost formula uses a prefix sum of `nums`. The Base Model spends ∼5K chars re-reading the problem statement and second-guessing its interpretation before reaching this conclusion; HMPO verifies it against examples in ∼1K chars.
2. **DP formulation**: The Base Model explores multiple DP formulations ($DP[i]$, $DP[i][m]$, back to $DP[i][m]$ with restructuring) before settling on the correct one. HMPO directly formulates the 2D DP and proceeds to optimization.
3. **Convex hull trick derivation**: The Base Model derives CHT from scratch with extensive algebraic manipulation (∼20K chars), while HMPO recognizes the standard pattern (minimizing $mx + b$ with monotone queries) and applies it in ∼5K chars.
4. **Implementation**: The Base Model's response includes both a lengthy explanation of the approach (10K chars) *and* the code (3.5K chars), with the explanation repeating much of the thinking. HMPO produces a clean code-only response (3.6K chars) without redundant explanation.

The key insight is that HMPO does not sacrifice algorithmic sophistication—it still discovers and

AIME 2026: Mathematical Reasoning Case (Answer: 178)

Question. Find the sum of the 10th terms of all arithmetic sequences of integers that have first term equal to 4 and include both 24 and 34 as terms.

Base Model — Thinking (35,965 chars).

"Define $a_n = 4 + (n-1)d$. Since the terms are integers and $a_1 = 4$, d must be an integer. From $24 = 4 + (m-1)d$ and $34 = 4 + (k-1)d$, we get $d \mid 20$ and $d \mid 30$, so $d \mid \gcd(20, 30) = 10$.

[...]

Wait, I should pause and check if 'arithmetic sequences of integers' implies something else. [...] *Could d be non-integer rational?* [...] *Is it possible that d is negative?* [...] If $d < 0$, the terms decrease... but $24, 34 > 4$, so d must be positive. *Wait, what if 24 was before a_1?* No, a_1 is the *first* term. [...]

Wait, what if the indices are not $1, 2, \ldots$ but can be negative? [...] *Wait, the sequence must contain these terms in that order, or not necessarily contiguous?* [...] *What if the sequence wraps around modulo something?* [...] *Wait, I might have missed a case. What if the arithmetic sequence is not defined by $a_n = a_1 + (n-1)d$?* [...]

So $a_{10} = 4 + 9d$, giving $\{13, 22, 49, 94\}$. Sum $= 13 + 22 + 49 + 94 = 178$."

Base Model — Response.

The arithmetic sequences are defined by $a_1 = 4$ and the common difference d. Since the terms are integers, d must be an integer. [...] For $24, 34$ to be terms, d must be a common divisor of 20 and 30, giving $d \in \{1, 2, 5, 10\}$. The 10th term is $a_{10} = 4 + 9d \in \{13, 22, 49, 94\}$. Sum $= 13 + 22 + 49 + 94 = 178$. $\boxed{178}$

HMPO — Thinking (8,631 chars).

"The n-th term is $a_n = a_1 + (n-1)d$ with $a_1 = 4$. Since $a_2 = 4 + d$ must be an integer, $d \in \mathbb{Z}$.

From $24 = 4 + (m-1)d$ and $34 = 4 + (k-1)d$, subtracting yields $d \cdot (k - m) = 10$. The terms are increasing, so $d > 0$. [...] d is a positive divisor of 10, i.e., $d \in \{1, 2, 5, 10\}$.

[...]

For each d, $a_{10} = 4 + 9d \in \{13, 22, 49, 94\}$. Sum $= 13 + 22 + 49 + 94 = 178$."

HMPO — Response.

Let $a_n = a_1 + (n-1)d$ with $a_1 = 4$. Since the terms are integers, $d \in \mathbb{Z}$. For $24, 34$ to be terms, there exist positive integers m, k with $(m-1)d = 20$ and $(k-1)d = 30$, so d is a common divisor of 20 and 30. Since $24 > 4$, $d > 0$, and $\gcd(20, 30) = 10$ gives $d \in \{1, 2, 5, 10\}$. The 10th term is $a_{10} = 4 + 9d \in \{13, 22, 49, 94\}$, whose sum is $\boxed{178}$.

Figure 7: Case study on AIME 2026.

implements the optimal $O(N^2)$ CHT solution. It simply avoids the exploration dead-ends, repeated self-correction, and verbose re-explanation that inflate the Base Model's output.

C.3 Summary of Observations

Across both domains, HMPO's compression manifests through three consistent mechanisms:

1. **Elimination of redundant verification**: The Base Model repeatedly re-checks conclusions it has already established. HMPO verifies once and moves forward.
2. **Direct problem-solving**: HMPO avoids exploring and discarding suboptimal approaches. It converges on the correct method faster, suggesting the RL training has taught the model to recognize solution patterns more efficiently.
3. **Concise final responses**: HMPO produces shorter final answers/code without redundant re-explanation of the reasoning already performed in the thinking block.

These patterns explain why moderate compression (42–46%) can preserve or even *improve* accuracy: by removing self-contradicting explorations and repetitive verification, the model reduces opportunities for error accumulation while retaining all essential reasoning steps.

LiveCodeBench: Code Generation Case

Problem. Divide nums ($n \leq 1000$) into subarrays to minimize the total cost, where the i-th subarray $[l, r]$ contributes $(\texttt{nums}[0]+\cdots+\texttt{nums}[r] + k \cdot i) \times (\texttt{cost}[l]+\cdots+\texttt{cost}[r])$.

Base Model — Thinking (136,169 chars). *Direct excerpts; red marks repeated re-derivations and self-doubt.*

> "We use dynamic programming. Let $DP[i][m]$ be the minimum cost to partition the prefix of length i into m subarrays. The transition is $DP[i][m] = \min_{j<i} \big(DP[j][m-1] + (P_N[i] + k\,m)(P_C[i] - P_C[j])\big)$."
>
> *"Wait, if we define $DP[i]$ as the min cost for prefix i ending with* some *number of subarrays, that equation is wrong because the new subarray needs to know m."* [...] *"But wait, does m matter?"* [...] *"Wait, $P_C[i]$ and $P_N[i]$ are fixed for a specific i."* [...] *"Wait, $P_N[i]$ and $P_C[i]$ are sequences."* [...] *"Wait, this is greedy, which might not work."* [...] *"Wait, look at the constraints again. $N \leq 1000$."* [...]
>
> *[Re-verifies Example 1 multiple times, recomputing $(3+1+1) \times (4+6) = 50$ and $(3+1+4+2) \times 6 = 60$ to confirm the output is 110.]* [...]
>
> "The term inside the minimum is of the form $S_j - C_j \cdot X$, where $S_j = DP[j][m-1]$ and $X = P_N[i] + k\,m$. Slopes $-P_C[j]$ are monotone and queries X are monotone, so we apply the convex hull trick (CHT) for $O(N^2)$ overall."

Base Model — Response.

> [...long mathematical exposition (∼10K chars) before code, duplicating the derivation in the thinking trace, including a full re-statement of the DP recurrence, the convex hull rewrite, slope monotonicity, and complexity analysis.]

```
class Solution:
  def minimumCost(self, nums, cost, k):
    n = len(nums)
    P_N, P_C = [0]*(n+1), [0]*(n+1)
    for i in range(n):
      P_N[i+1] = P_N[i] + nums[i]
      P_C[i+1] = P_C[i] + cost[i]
    # . . . CHT with deque, ∼120 lines . . .
```

HMPO — Thinking (44,109 chars). *Direct excerpts; the trace is largely linear.*

> "Let $P[i]$ and $C[i]$ be the prefix sums of nums and cost. The i-th subarray contributes $(P[r] + k \cdot i)(C[r] - C[l-1])$.
>
> [...verifies Example 1 once: subarrays $[3, 1]$ and $[4]$ give $50 + 60 = 110$.]
>
> Let $DP[i]$ be the minimum cost for the prefix of length i. Since the cost depends on the subarray index c, we add c as an outer dimension and compute $DP[i][c]$ from $DP[\cdot][c-1]$.
>
> The inner minimization is $\min_{j<i} \big(DP[j][c-1] - (P[i] + k\,c) \cdot C[j]\big)$, which is a line query of the form $-C[j]\,x + DP[j][c-1]$ with $x = P[i] + k\,c$. Since slopes $-C[j]$ and queries x are both monotone, the convex hull trick gives amortized $O(1)$ per transition, hence $O(N^2)$ overall."
>
> [...]
>
> *"Final check on time complexity. $O(N^2)$ with $N = 1000$ is 10^6 operations, well within limits."*

HMPO — Response.

```
class Solution:
  def minimumCost(self, nums, cost, k):
    n = len(nums)
    prefix_nums = [0]*(n+1)
    prefix_cost = [0]*(n+1)
    for i in range(n):
      prefix_nums[i+1] = prefix_nums[i] + nums[i]
      prefix_cost[i+1] = prefix_cost[i] + cost[i]
    dp = [float('inf')]*(n+1); dp[0] = 0
    # . . . CHT with deque, ∼70 lines . . .
```

Figure 8: Case study on LiveCodeBench (LeetCode Hard)