On the Token Value Inequality in Efficient Reasoning

Runjia Zeng[1, 6], **Hang Hua**[2], **Yiyang Liu**[3], **Zhiqiang Tao**[1], **Ruixiang Tang**[4], **Qifan Wang**[5], **Cheng Han**[3], **Dongfang Liu**[6†]

[1]Rochester Institute of Technology [2]MIT-IBM Watson AI Lab
[3]University of Missouri-Kansas City [4]Rutgers University
[5]Meta AI [6]Purdue University [†]Corresponding author

Abstract

Chain-of-Thought reasoning has enabled large language models to achieve substantial performance gains on complex tasks. However, these gains come at the cost of dramatically increased token consumption. This raises a fundamental question: is every token in the reasoning trace equally valuable? We present a diagnostic and optimization framework grounded in a key empirical finding: the value of tokens within a CoT reasoning sequence is highly non-uniform, and this non-uniformity can be effectively characterized by token-level log probability signals. We show that normalized log probability helps distinguish core tokens, which carry structural and decisive reasoning content, from redundant tokens, which are exploratory, low-confidence filler that contributes less directly to the final answer. Building on these findings, we formulate the **TokenProbe** framework around two empirical findings and one claim: findings identify token value inequality first and then establish TokenProbe as a core-token proxy, and the claim introduces an efficient GRPO objective positing that selectively compressing redundant tokens can yield Pareto improvements in the accuracy–token efficiency space. Empirically, our method preserves reasoning quality while reducing the token usage by 76% of the baseline. Under matched reasoning-length budgets, we show that it can even outperform strong flagship baselines like Gemini-3.1-Pro. Homepage: runjia.tech/tokenprobe.

1 Introduction

The training objective of an autoregressive language model [7, 56, 9] is to maximise the log-likelihood of the observed token sequence, $\mathcal{L} = \sum_{t=1}^{T} \log p_\theta(s_t \mid s_{<t},\, x)$. Under this objective, when the model faces a complex reasoning problem, directly predicting the answer yields a high-entropy conditional distribution, the model is uncertain about the answer, and $p(y \mid x)$ spreads across many candidates. Chain-of-Thought (CoT) reasoning [76, 34] addresses this by introducing intermediate reasoning steps z that decompose the prediction as $p_\theta(y \mid x) = \sum_z p_\theta(z \mid x)\, p_\theta(y \mid x, z)$. Each reasoning step z_i narrows the solution space conditioned on its predecessors, substantially improving accuracy on multi-step reasoning tasks by explicitly introducing latent variables that reduce the dimensionality of the prediction problem [90, 86, 24].

However, the introduction of latent variables and reasoning steps brings the extreme token usage spurge along the performance gain [13, 67, 5, 4, 68] (*i.e.*, enabling CoT on Qwen-4B adds 24.1% in accuracy at a cost of 4.5× more tokens). As shown in the Fig. 1, CoT reasoning currently exhibits several characteristic patterns. The **green curve** represents the ideal reasoning trajectory, where the model moves from the question to the correct answer through a relatively direct semantic mapping. In contrast, the **red curve** illustrates a fatal deviation path, in which one decisive wrong directional exploration drifts toward an incorrect conclusion. The **purple curve** depicts a circular deadlock path, where the model expends substantial token budget while becoming trapped in repetitive or unproductive loops [28]. The **blue curve** corresponds to a tortuous convergence path, where the model eventually reaches the correct answer, but only after a long and unnecessarily oscillatory reasoning process. This raises a central question: *Are these tokens uniformly valuable, especially*

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

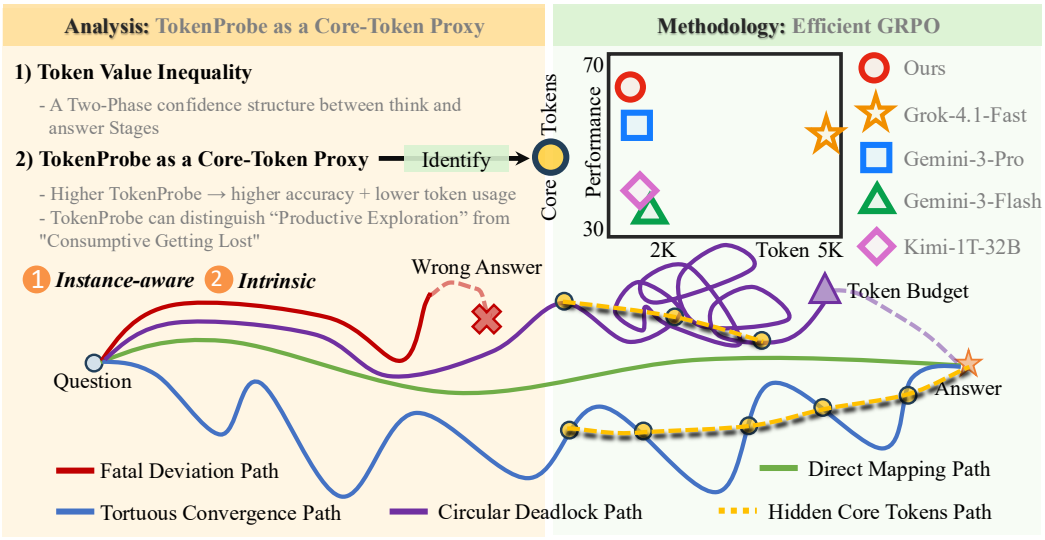


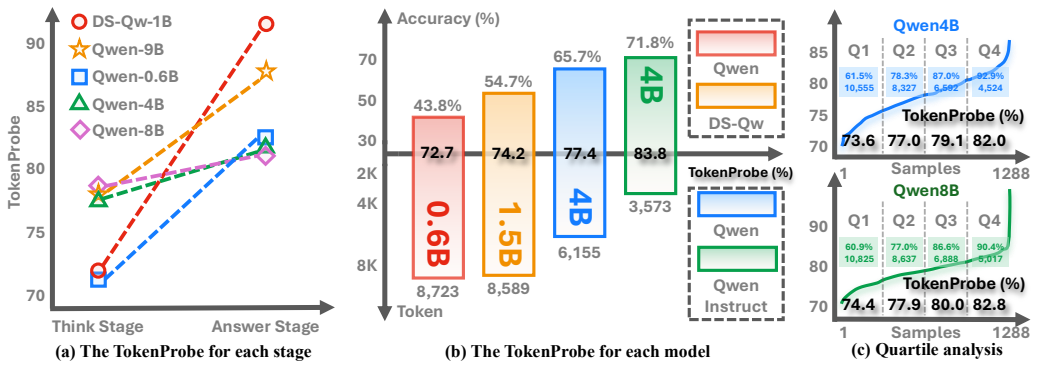
Figure 1: **Overview of TokenProbe: from token value inequality analysis §2 to efficient reasoning §3**. (a) Probing analysis shows that token contributions are highly unequal, and TokenProbe serves as an proxy for identifying core tokens. (b) Building on this insight, efficient GRPO improves it by preserving hidden core tokens, yielding better reasoning efficiency and Pareto performance.

*along the **purple curve** and the **blue curve**?* If not, it means a substantial fraction of the extra tokens is low-value exploratory redundancy, there exists the possibility of a **Pareto improvement**: preserving core reasoning content while drastically cutting redundant tokens, thereby greatly reducing the cost of model using since all models are token-based billing under the next token prediction paradigm [33].

Answering this question requires a suitable token-granularity diagnostic signal. Concretely, such a signal would arise as an inference-native by-product of generation, without relying on additional reward models [53, 37, 72] or human annotation, thus avoiding post-processing overhead and extra forward passes. It would also preferably offer finer granularity, allowing more flexible intra-sequence discrimination than sequence-level or turn-level scores. Motivated by this, we naturally turn to the autoregressive model's own output, **token-level log probability** [73, 74, 85]. It is a direct product of the training objective, already present at every decoding step, and its relative variation within a sequence encodes precisely the model's local confidence dynamics.

Building on this raw token-level signal, we introduce two derived quantities that play complementary roles. The Normalized Log Probability (NormLP) highlights candidate decisive tokens within a response, while the TokenProbe aggregates this signal into a scalar that tracks the model's accuracy and token usage. They jointly drive our efficient GRPO (see §3) and explain *why* it yields Pareto improvements. Two properties make this lever sharp. ❶ **Instance-aware.** By normalization, NormLP is a relative signal within each response. It identifies which tokens have confidence above the response mean (positive NormLP) and which fall below (negative NormLP), therefore distinguishing the core tokens flexibly and dynamically (see Tab. 1), rather than relying on absolute magnitudes. Building on this contrastive signal, we further define the TokenProbe of a response as the ratio of its tokens with positive NormLP. ❷ **Intrinsic.** For a given model, the TokenProbe is relatively stable and robust to hyperparameter variations (see §S2). It therefore reflects the model's intrinsic degree of certainty and training characteristics without additional processing or annotations, and in turn helps evaluate the model's reasoning ability (see Fig. 2), including both its accuracy and its token usage.

The remainder of the paper is structured as follows. In §2, we conduct a systematic probing analysis of token-level log probability, formally introducing the NormLP and TokenProbe signals. Through two key findings, we establish that CoT reasoning exhibits a two-phase confidence structure (**Finding 1**) and that TokenProbe is consistently associated with both reasoning accuracy and token usage (**Finding 2**), laying the empirical foundation for our approach. Building on these insights, §3 presents our efficient GRPO as a **Claim**, which translates the NormLP-based analysis into practice through two complementary mechanisms, *Selective KL Anchoring*, which focuses regularization on core tokens identified by NormLP, and *Reward Shaping*, which incentivizes dense and concise reasoning trajectories. §4 reports comprehensive experiments across multiple benchmarks and model families, showing a consistent trend toward Pareto improvement while maintaining accuracy and substantially reducing token usage. We close in §4.3 with a detailed discussion of training dynamics, out-of-domain efficiency transfer, and diagnostic ablation.



(a) The TokenProbe for each stage (b) The TokenProbe for each model (c) Quartile analysis

Figure 2: **TokenProbe as an indicator of reasoning efficiency**. (a) TokenProbe increases from the think stage to the answer stage. (b) Across models, higher TokenProbe corresponds to better accuracy and lower token consumption. (c) Quartile analysis shows the same trend at the sample level.

2 Analysis

Since raw log probabilities vary in scale across responses, making direct comparison unreliable, we normalize them within each response to obtain NormLP , and further aggregate it into a single scalar, TokenProbe, that stably reflects the model's intrinsic reasoning confidence. Formally:

Definition 1 (Normalized Log Probability)**.** Given a response consisting of tokens $\{s_1, \ldots, s_T\}$ with per-token log probabilities $\ell_t = \log p_\theta(s_t \mid s_{<t})$, the normalized log probability of token t is

$$\mathrm{NormLP}_t = \frac{\ell_t - \bar{\ell}}{\sigma_\ell + \epsilon}, \quad \text{where} \quad \bar{\ell} = \frac{1}{T}\sum_{t=1}^{T} \ell_t, \quad \sigma_\ell = \sqrt{\frac{1}{T}\sum_{t=1}^{T}(\ell_t - \bar{\ell})^2}. \tag{1}$$

Definition 2 (TokenProbe, Positive Ratio of Normalized Log Probability)**.** We define **TokenProbe** as the ratio of tokens whose normalized log probability is positive, where $\mathbb{I}[\cdot]$ is the indicator function.

$$\mathrm{TokenProbe} = \frac{1}{T}\sum_{t=1}^{T} \mathbb{I}[\mathrm{NormLP}_t > 0]\,, \tag{2}$$

Leveraging TokenProbe, we study five models of different scales across four reasoning benchmarks, AIME, MATH [27, 37], AMC, and Olympiad-Bench [25], under varied hyperparameter settings (see more in §S2). We organize the analysis in a coarse-to-fine manner and formulate two central findings supported by the resulting empirical evidence.

Finding 1: Token Value Inequality

► A two-phase confidence structure between think and answer stages.

To understand token contributions, we begin by examining the output structure of common reasoning models. The output of a thinking-enabled model [21, 49] is naturally segmented into two phases: **Think phase**, spanning the content between `<think>` and `</think>`, contains the intermediate reasoning process; and **Answer phase**, occurring after `</think>`, contains the final answer output.

As shown in Fig. 2(a), the log-probability distributions of the two phases exhibit a pronounced duality: Answer-phase TokenProbe is systematically 3%–20% higher than Think-phase TokenProbe. This reveals the underlying structure of CoT reasoning: the Think phase is a high-cost search process (*i.e.*, the model is exploring solution paths), while the Answer phase is a low-cost decision output (*i.e.*, the model has formed a definite conclusion) [90, 24]. The "value" of reasoning is not uniformly distributed across every token of the Think phase. Part of it serves genuine reasoning functions, while the rest is exploratory noise.

Finding 2: TokenProbe as a Core-Token Proxy

► Higher TokenProbe → higher accuracy + lower token usage.
► TokenProbe can distinguish "Productive Exploration" from "Consumptive Getting Lost".

Building on the finding of token value inequality, we then conduct a finer-grained analysis of TokenProbe to examine its underlying characteristics. More specifically, we define reasoning quality as the ability to achieve higher accuracy under a relatively low yet still reasonable token budget [32, 13]. Under this view, high-quality reasoning is not merely shorter reasoning, but reasoning that avoids unnecessary detours and self-corrections while preserving performance.

Table 1: Functional analysis of the "Wait" token at different positions within a single reasoning trace.

Position	NormLP	Semantic Role	Impact
Token [54] (early)	$+0.47$	First verification discovers misinterpretation	Productive
Token [103] (early)	$+0.22$	Mild self-correction	Productive
Token [575] (mid)	-3.87	Deep confusion with hesitation	Wasteful
Token [1273] (mid-late)	-4.13	Re-questions already-confirmed step	Wasteful (+700 tokens)

As shown in Fig. 2(b), our results suggest that TokenProbe serves as an effective proxy for this notion of reasoning quality. Across four representative models of different scales, we observe a clear pattern: as TokenProbe increases, accuracy consistently rises, while token usage consistently falls. This indicates that higher TokenProbe is associated not with premature truncation, but with more efficient reasoning trajectories that remove unnecessary repeated corrections. Moreover, this pattern is not confined to these four cases. Across all experimental settings we analyze in §S2, the Pearson correlation between TokenProbe and token usage is consistently negative, whereas the correlation between TokenProbe and accuracy is consistently positive, further supporting that TokenProbe is a useful empirical indicator of both reasoning effectiveness and token efficiency [73].

As shown in Fig. 2(c), we do a further analysis on Qwen-4B and Qwen-8B. Accuracy increases by 31.4% from the lowest to the highest quartile, while token usage drops by 57%. This pattern replicates faithfully on other model scales (see more in §S2).

While Findings 1–2 characterize reasoning quality at the phase and response level, identifying which specific tokens within a trace are productive requires finer-grained analysis. We therefore drill down to the individual token level [38, 73], demonstrating how NormLP provides fine-grained semantic discrimination within a reasoning trace (see the complete case in §S2). This finding carries the most direct practical implications. As shown in Tab. 1, we perform a detailed analysis of the commonly used token "Wait" at different positions within a single reasoning trace. We find that early occurrences often reflect productive self-correction, whereas late occurrences tend to be wasteful. This distinction is reflected by positive versus negative NormLP, thereby providing an empirical foundation for identifying core tokens in the subsequent efficient GRPO framework.

3 Methodology

Claim: Efficient GRPO

▶ Achieving pareto improvement through efficient GRPO guided by the NormLP signal.

Findings 1 and 2 jointly motivate a crucial claim: CoT reasoning contains a large volume of redundant tokens that are identifiable by NormLP signals, and these redundant tokens are directly associated with low accuracy and high token consumption. Therefore, if training can guide a model to: (a) preserve the reasoning density of core tokens, and (b) compress or eliminate redundant tokens, then it is possible to substantially reduce the token budget of CoT reasoning without significant loss in final-answer accuracy, achieving a Pareto efficiency improvement.

Based on the analysis above, we build upon standard GRPO [64, 62, 61], which maximises the per-token objective

$$\mathcal{L}_{\text{GRPO}} = \mathbb{E}\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\text{old}}(a_t \mid s_t)} \hat{A}_t \;-\; \beta_{\text{base}}\, D_{\text{KL}}\big(\pi_\theta \,\|\, \pi_{\text{ref}}\big) \right], \tag{3}$$

where $\hat{A}_t$ is the group-relative advantage and β_{base} is a fixed base KL coefficient [99, 53]. Guided by the NormLP signal, we propose **two complementary modifications** of Eq. 3 that target different components of the same skeleton in Fig. 3. **Selective KL Anchoring** (**KL**, §3.1) reshapes the *regularization* term so that redundant tokens receive an explicit sparsity pressure once a rollout overruns its budget. **Reward Shaping** (**RS**, §3.2) reshapes the *advantage* inside the policy term so that core tokens on correct rollouts are upweighted while overlong redundant spans are penalised. The two variants share the same core-token mask and leave the update on wrong-answer rollouts (*i.e.*, $r^{\text{seq}} \leq 0$) untouched (see more discussion in §S4.5).

To avoid overly sparse rewards and preserve semantic coherence, without introducing additional annotation effort at the turn level, we identify core tokens through a mathematically motivated window selection strategy (see more discussion in §S3.2). The core window is selected based on NormLP, with the intuition that tokens in high-density regions are more likely to correspond to essential reasoning steps [38, 16]. Specifically, given the normalized scores $\{\mathrm{NormLP}_t\}_{t=1}^{T}$, we

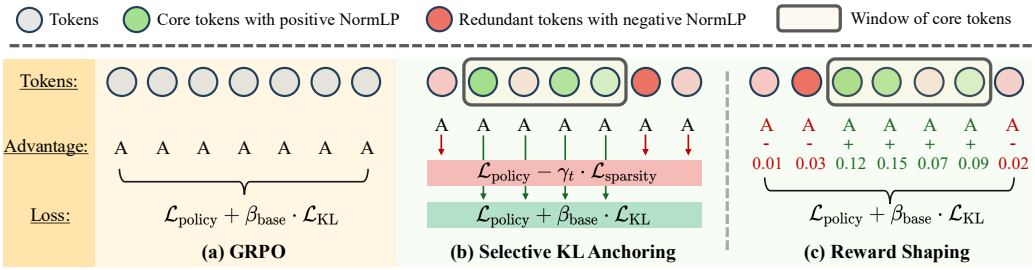


Figure 3: **Comparison of training strategies for efficient reasoning**. (a) Vanilla GRPO treats all tokens equally. (b) **Selective KL Anchoring (KL)** rescales the KL regularizer per token and imposes a sparsity penalty on over-budget redundant tokens. (c) **Reward Shaping (RS)** reshapes the token-level advantage, upweighting core tokens and penalising overlong redundant spans.

apply a sliding window of size W and compute the cumulative score for each window as $s_i^{(W)} = \sum_{t=i}^{i+W-1} \mathrm{NormLP}_t$. Finally, we greedily select the highest-scoring non-overlapping windows and define the corresponding token positions as core tokens (see §S3.1), yielding a binary core mask $CT_t \in \{0, 1\}$, where $CT_t = 1$ indicates a core token and $CT_t = 0$ a redundant token.

3.1 Selective KL Anchoring (KL)

As illustrated in Fig. 3(b), **KL** keeps the policy term of Eq. 3 intact and makes only the regularizer token-adaptive:

$$\mathcal{L}_{\text{KL}} = \mathbb{E}\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\text{old}}(a_t \mid s_t)} \hat{A}_t \boxed{- \beta_t D_{\text{KL}}\big(\pi_\theta \,\|\, \pi_{\text{ref}}\big) - \gamma_t \log \pi_\theta(a_t \mid s_t)} \right], \tag{4}$$

where the boxed block replaces the base KL term of Eq. 3. The β_t is a per-token stability coefficient [57] and γ_t activates a log-probability sparsity term on suppressible tokens (see §S3.3). The two coefficients are dispatched by a simple rule parameterised by the core mask CT_t and the current rollout length T relative to the length limit L_{limit}:

Core tokens ($CT_t = 1$). We keep the vanilla regularizer (*i.e.*, $\beta_t = \beta_{\text{base}}$, $\gamma_t = 0$), anchoring optimisation on reasoning-essential tokens without adding any sparsity pressure. **Over-budget redundant tokens ($CT_t = 0$, $T > L_{\text{limit}}$).** We remove the KL constraint ($\beta_t = 0$) and inject an overflow-proportional sparsity penalty $\gamma_t = \lambda_{\text{decay}} \cdot (T - L_{\text{limit}})/T$, suppressing further redundant generation once a trace has already exceeded the budget. **Within-budget redundant tokens ($CT_t = 0$, $T \le L_{\text{limit}}$).** We fall back to the stable setting ($\beta_t = \beta_{\text{base}}$, $\gamma_t = 0$), tolerating moderate redundancy rather than prematurely suppressing it.

3.2 Reward Shaping (RS)

As illustrated in Fig. 3(c), **RS** keeps the regularizer of Eq. 3 intact and instead reshapes the advantage inside the policy term.

$$\mathcal{L}_{\text{RS}} = \mathbb{E}\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\text{old}}(a_t \mid s_t)} \boxed{\hat{A}_t^{\text{RS}}} - \beta_{\text{base}} D_{\text{KL}}\big(\pi_\theta \,\|\, \pi_{\text{ref}}\big) \right], \tag{5}$$

where the boxed $\hat{A}_t^{\text{RS}}$ replaces the vanilla $\hat{A}_t$ of Eq. 3 on correct rollouts ($r^{\text{seq}} > 0$) and falls back to $\hat{A}_t$ otherwise. Concretely,

$$\hat{A}_t^{\text{RS}} = \hat{A}_t + \beta \left(\alpha \,\mathrm{NormLP}_t \,\mathbb{I}[CT_t = 1] - \lambda \frac{\max(0,\, T - L_{\text{limit}})}{T} \mathbb{I}[CT_t = 0] \right). \tag{6}$$

where NormLP_t is the z-score normalised log probability, α upweights core tokens along high-reward trajectories, λ penalises redundant tokens once a trace runs long (see §S3.4), and $L_{\text{limit}} = \text{core_len} \times \text{current_scale}$ is a length budget progressively tightened through curriculum annealing [8]. We deliberately restrict shaping to correct rollouts (*i.e.*, wrong-answer rollouts retain the vanilla $\hat{A}_t$), since rewarding core tokens on a failed reasoning trace would propagate misleading credit to precisely the tokens that should be re-examined.

Table 2: **Main results on four in-domain reasoning benchmarks. #Tk.** is the average number of generated tokens per response (lower is better). **Acc.** is averaged accuracy in % (higher is better).

Model	Average		AIME		MATH		AMC		Olympid	
	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.
Closed-API Reasoning Models										
GPT-5.4-Mini	987	69.6	1714	36.7	589	88.6	1137	73.5	1232	56.6
Claude-4.6-Sonnet	1221	61.3	1907	20.0	801	86.0	1413	51.8	1478	46.1
Grok-4.1-Fast	4945	52.3	9361	10.0	1866	74.8	4017	43.4	7144	38.5
Gemini-3.1-Pro	1586	54.4	2020	10.0	1274	79.2	1798	37.3	1771	40.1
Gemini-3-Flash	1828	35.1	2039	3.3	1625	56.0	1948	24.1	1954	22.4
Open Source Reasoning Models										
Qwen-80B-3B [82]	1843	28.3	2048	0.0	1646	51.0	1972	13.3	1965	14.7
GLM-358B-32B [19]	1545	44.5	2039	6.7	1182	68.0	1775	31.3	1763	30.4
DeepSeek-671B-37B [39]	1542	46.7	1978	13.3	1200	70.2	1733	36.1	1751	32.1
Kimi-1T-32B [69]	1649	40.0	2048	0.0	1342	63.4	1807	28.9	1840	25.8
Efficient Reasoning Models										
DEER [84] [TMLR'25]	1891	16.0	2622	13.3	1082	50.6	2040	0.0	1821	0.0
NoThink [46] [Arxiv'25]	1156	10.3	1626	3.3	738	37.8	1146	0.0	1114	0.0
L1 [2] [COLM'25]	4573	43.2	6058	20.0	2819	71.4	4946	42.2	4470	39.3
O1-Pruner [42] [Arxiv'25]	5248	39.5	7051	10.0	3179	70.8	5413	42.2	5347	35.1
PEAR [30] [ICLR'26]	4240	41.8	6030	16.7	2322	69.6	4707	43.4	3901	37.3
ETR [79] [ACL'26]	4566	42.2	6209	20.0	2799	70.4	4889	39.8	4366	38.7
Ours										
Qwen-0.6B [82]	5890	40.7	7291	16.7	3951	69.8	6308	39.8	6010	36.6
- w/ KL	2023	42.3	2195	16.7	1717	72.0	2120	44.6	2060	35.9
- w/ RS	1806	36.3	1898	6.7	1503	68.6	1987	36.1	1834	33.8
DeepScaleR-1.5B [44]	4741	59.6	6564	26.7	2769	88.0	4790	69.9	4841	53.8
- w/ KL	1698	56.1	2065	20.0	1185	85.8	1816	66.3	1726	52.1
- w/ RS	1993	55.9	2365	23.3	1494	85.4	2089	63.9	2023	50.8
Qwen-4B [82]	6132	62.4	7683	30.0	4195	92.0	6274	72.3	6376	55.4
- w/ KL	1886	61.0	2253	33.3	1246	88.2	2092	66.3	1952	56.1
- w/ RS	2026	62.3	2532	30.0	1391	90.2	2154	71.1	2025	58.1
Phi-Reasoning-4B [1]	5515	59.6	7237	26.7	3559	87.4	5362	68.7	5901	55.6
- w/ KL	2919	62.0	4190	26.7	1756	89.8	2992	72.2	2737	59.1
- w/ RS	2992	62.6	3997	26.7	1977	89.8	2982	74.7	3011	59.1
Qwen-8B [82]	6252	61.9	7737	33.3	4412	89.2	6404	71.1	6454	53.9
- w/ KL	1476	62.7	1945	30.0	1008	88.6	1486	73.5	1466	58.5
- w/ RS	2313	68.3	3276	33.3	1462	94.2	2265	81.9	2249	63.9

4 Experiments

4.1 Experimental Setup

Training and Evaluation. Following [2], we train on the DeepScaleR-Preview dataset [44], a 40K math corpus drawn from AIME, AMC, Omni-MATH [18], and STILL [47]. We evaluate on four in-domain reasoning benchmarks: AIME 2025, MATH-500 [27, 37], AMC, and OlympiadBench [25]. The full training recipe, per-benchmark rollout budgets, and prompts are deferred to Appx. S4.1 and Appx. S4.2. Training uses a 4,096-token rollout cap and evaluation an 8,192-token cap for efficient reasoning models and ours. As for the backbone of ours, we train and evaluate our method across five backbones spanning 0.6B–8B parameters: Qwen3-0.6B / 4B / 8B [82], DeepScaleR-1.5B [44], and Phi-4-mini-reasoning [1], covering various model families and scales.

Baselines. We group baselines into three categories (full configuration in Appx. S4.3): (i) *closed-API reasoning models*: Claude-4.6-Sonnet, Grok-4.1-Fast, Gemini-3.1-Pro, Gemini-3-Flash, and GPT-5.4-mini; (ii) *open-source large reasoning models*: Qwen3-Next-80B-A3B [82], GLM-4.7-358B-32B [19], DeepSeek-V3.2-671B-37B [39], and Kimi-K2-1T-32B [69]; (iii) *efficient reasoning methods*: DEER [84], NoThink [46], L1 [2], O1-Pruner [42], PEAR [30], and ETR [79].

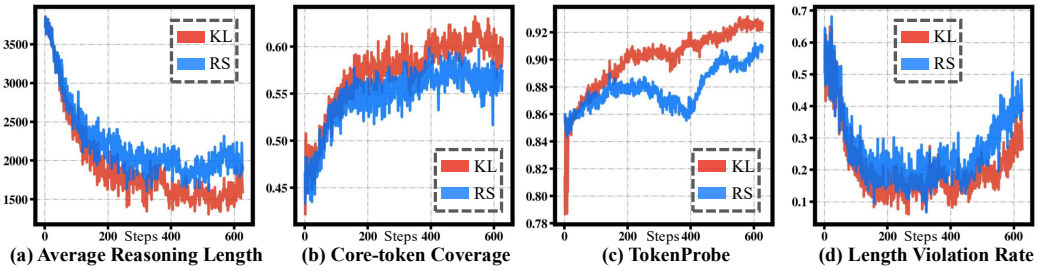


Figure 4: **TokenProbe signals during RL training**.

4.2 Main Results

Tab. 2 leads to three key observations organised along the *same-budget* and *same-accuracy* perspectives. **First, performance superiority at a matched token budget comparison.** At a $\sim$2K-token rollout budget, our method matches or surpasses some closed-API and open-source large reasoners. Qwen-8B-w/KL reaches 62.7% at 1,476 tokens and outperforms Gemini-3.1-Pro by $+8.3\%$ (*i.e.*, 62.7 $vs.$ 54.4 at 1,476 $vs.$ 1,586 tokens), and Kimi-1T-32B by $+21.6\%$. Our narrower point is that TokenProbe training moves open sub-10B models toward better accuracy and token trade-offs at far lower serving cost. See more comparisons in §S4.9 and § S4.2. **Second, generality across backbones with drastic token reduction.** Applied to every backbone, our objectives compress response length by 1.8–$4.2\times$ with essentially no accuracy loss. Qwen-8B-w/KL matches the base Qwen-4B (*i.e.*, 62.7 $vs.$ 61.9, $+0.8\%$) while using $4.2\times$ fewer tokens (6,252 $vs.$ 1,476). The effect is most pronounced on Qwen-8B-w/RS, which raises accuracy by $+6.4\%$ (*i.e.*, 68.3 $vs.$ 61.9) while cutting tokens from 6,252 to 2,313 ($2.7\times$). This suggests that compressing redundant chain-of-thought acts as an implicit regularizer [63] over the reasoning trace. More importantly, improved reasoning efficiency reduces the risk that generation is prematurely constrained by the token budget, thereby increasing the likelihood of reaching a complete respond within the available budget. For example, on AIME, the Qwen-8B base model uses substantially more tokens than our method on average (7,737 $vs.$ 3,276), and similar trends hold for AMC and Olympiad. **Third, breaking the accuracy and efficiency trade-off.** Using Qwen-0.6B as the common backbone (base at 40.7% with 5,890 tokens), existing efficient-reasoning methods fall into two failure modes along a trade-off that our method avoids. *(i) Aggressive compression at a catastrophic accuracy cost.* DEER collapses to 16.0% at 1,891 tokens and NoThink to 10.3% at 1,156 tokens (*i.e.*, -24.7% and -30.4% relative to base), despite using similar or fewer tokens than our run. *(ii) Accuracy preserved but little compression.* PEAR (41.8% at 4,240 tk) and ETR (42.2% at 4,566 tk) both match base-model accuracy but deliver only a 1.3 to $1.4\times$ token reduction. In contrast, Qwen-0.6B-w/KL attains 42.3% at 2,023 tokens (*i.e.*, $+1.6\%$ over base at $2.9\times$ compression), matching the strongest training-based baseline while using 2.2 to $2.3\times$ fewer tokens and avoiding the accuracy collapse of the inference-time baselines. Additional analysis is provided in §4.3.

4.3 Discussion

4.3.1 Signals Along RL Training

To understand *how* our objectives compress reasoning, we track four diagnostic signals along RL training on Qwen-4B (Fig. 4). **(a) Average reasoning length.** This signal is the mean number of response tokens generated per rollout, our most direct proxy for training-time compression. As shown in Fig. 4(a), both KL and RS rapidly pull the response length from roughly 3.7K tokens at the start of RL to a much shorter regime [4, 2]. KL compresses more aggressively, stabilising at 1.6 to 1.8K tokens, whereas RS remains longer at 1.9 to 2.0K tokens. This indicates that both objectives successfully suppress excessive deliberation, but with different strength. KL enforces a hard sparsity pressure, while RS trades compression for a softer reward-shaped adjustment, which is consistent with the results shown in Tab. 2. **(b) Core-token coverage.** This signal is the fraction of generated tokens covered by TokenProbe-selected core windows (*i.e.*, the number of selected windows $\times$ window size, normalised by the average response length), and therefore measures how densely the remaining tokens carry useful signal. Fig. 4(b) shows that coverage rises steadily under both objectives, meaning the surviving tokens become progressively denser in TokenProbe-selected content. KL reaches a higher final coverage than RS, suggesting that KL removes redundant context more sharply and leaves a larger fraction of the response occupied by high-value windows. RS follows the same upward trend but stays lower, consistent with its longer generations, improving the density

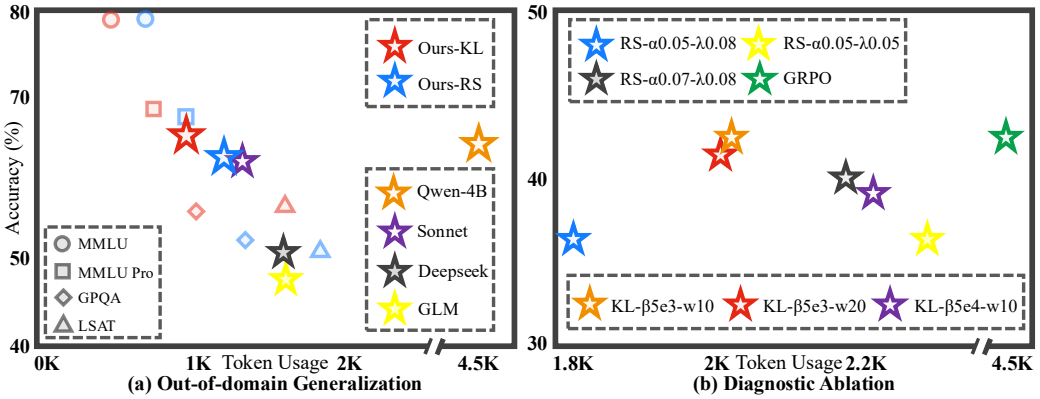


Figure 5: **Transfer and ablation of TokenProbe-based efficient reasoning**. (a) Our variants match accuracy with far fewer tokens. (b) Different hypeparameter settings for KL and RS (more in §S3.2).

of useful tokens while retaining more non-core connective or exploratory material. **(c) TokenProbe.** As shown in Fig. 4(c), both methods trace a clear upward trajectory. KL rises from $\sim$0.84 to above 0.92, while RS increases more smoothly and eventually approaches the same regime. The steeper KL curve is consistent with a sharper compression effect, whereas the smoother RS curve reflects reward-shaped improvement without fully eliminating exploratory material. **(d) Length violation rate.** This signal is the fraction of valid responses whose length exceeds the dynamic TokenProbe budget (*i.e.*, the selected core length multiplied by the current curriculum scale), diagnosing how well the policy stays within a progressively tightened budget. Fig. 4(d) shows that the violation rate first drops substantially as the model learns to respect the TokenProbe-derived budget, then rebounds later in training, especially for RS. This rebound is expected because the budget is progressively tightened through the curriculum scale while the policy is still adapting. See more in §S6.

4.3.2 Out-of-domain Efficiency Transfer

To assess whether the token-efficiency gains transfer beyond mathematics, we evaluate the Qwen-4B base model, RS, and KL on four additional benchmarks spanning scientific, legal, and multi-disciplinary reasoning, including MMLU [26], MMLU-Pro [75], LSAT [98], and GPQA [59]. As shown in Fig. 5, both objectives substantially compress length relative to the base model (*i.e.*, 4,401 tokens), with RS reduced to 1,179 tokens (3.7$\times$) and KL to 935 tokens (4.7$\times$). Crucially, this compression is not paid for in accuracy. RS maintains a competitive average of 57.74 *vs.* 59.62 (*i.e.*, -1.9%), while KL actually *improves* over the base model to 60.61 (*i.e.*, $+1.0\%$) despite using far fewer tokens. The effect is most pronounced when compared with state-of-the-art closed-source models such as Sonnet 4.6, where our method achieves higher accuracy while using relatively same tokens ($\sim 150\times$ cheaper in Tab. S30). This confirms that the efficiency generalize robustly to diverse reasoning domains (see more discussions in §S4.2).

4.3.3 Diagnostic Ablation

We ablate the two hyperparameter groups of our method, the coefficients (β, w) in KL and the coefficients (α, λ) in RS, reporting average token count and accuracy on the four math benchmarks. The GRPO baseline reaches 42.3% at 4,461 tokens on average and serves as the reference point throughout. **Coefficients (α, λ) in RS.** A key design choice is the balance between the core-token reward α and the redundancy penalty λ, which governs how aggressively redundant tokens are suppressed without starving the model of informative-token credit. Holding α=0.05, raising λ from 0.05 to 0.08 reduces token usage by 21% (*i.e.*, 2,291 $\rightarrow$ 1,806) at essentially unchanged accuracy (*i.e.*, 36.1 *vs.* 36.3), confirming that a stronger length penalty compresses responses without harming correctness. Further raising α to 0.07 at λ=0.08 improves accuracy to 39.9% at 2,179 tokens, indicating that a larger core-token reward encourages more informative reasoning rather than merely shorter traces. **Coefficients (β) and window size (w) in KL.** The shaping coefficient β controls the magnitude of the per-token advantage correction, while the window size w controls the neighbourhood over which TokenProbe density is averaged. Under a matched window, β=5×10^{-3} outperforms β=5×10^{-4} (*i.e.*, 42.3 *vs.* 39.0 at $\approx$2,200 tokens), indicating that a sufficiently strong shaping signal is necessary for the token-level guidance to affect policy updates. Comparing window sizes at β=5×10^{-3}, w=10 yields 42.3% at 2,023 tokens *vs.* 41.4% at 2,008 tokens for w=20, a marginal difference that suggests the method is insensitive to moderate window-size variation.

5 Insights and Limitations

We examine whether compressed traces can teach efficient reasoning, where further compression becomes unstable, and how the validity of TokenProbe extends across training regimes and tasks. These analyses highlight the need to preserve useful self-correction and sufficient intermediate computation while checking that confidence remains aligned with reasoning quality. Readers can inspect these interactions on our **project homepage** through interactive core-token window visualizations, case studies displaying every token's NormLP, and accuracy–cost Pareto plots spanning the full set of evaluated models and their reported evaluation settings.

5.1 Unsuccessful Attempts: SFT Distillation

We explored SFT on compressed teacher traces to test whether imitation can transfer concise reasoning. Offline trace compression can remove the causal dependencies that make reasoning successful. Useful self-corrections can carry low NormLP and be discarded together with redundant detours. The surviving steps are then concatenated across missing context, forcing the student to imitate transitions that no longer contain the information on which the teacher's decisions depended. Our SFT distillation failures are consistent with this disruption of the reasoning process (§S5). The usefulness of a token depends on its role within the trajectory, including the revisions and checks needed to preserve coherent reasoning in subsequent steps and maintain a complete path to the final answer.

Online feedback offers a way to learn concision while preserving useful exploration. Complete rollouts retain the context of each decision, and outcome-based updates can reinforce self-correction when it contributes to a solution. This motivates learning compact reasoning through differential reinforcement within intact trajectories. Filtered SFT may still provide a useful initialization, but its suitability depends on whether the resulting policy retains coherent reasoning and a meaningful confidence signal before TokenProbe-guided RL begins (§S5.4).

5.2 Limits of Token Compression

The efficiency gains raise the question of how far compression can proceed before reasoning or training becomes unstable. Stable compression requires preserving sufficient intermediate computation. Excessive shortening pressure can disrupt this computation and drive generation into repetition. In this regime, response length can increase as accuracy collapses, so the failure appears as a joint deterioration of reasoning quality and efficiency (§S6). Token-aware optimization must therefore preserve the policy's capacity to complete a solution as the budget tightens, allowing room for useful self-correction and the intermediate steps needed to resolve the problem.

The attainable compression level depends on the checkpoint and the difficulty distribution of the task. Problems requiring more sequential computation leave less room for shortening, while the policy's initialization can affect its stability under optimization. The reported collapse trajectories and supplementary budget sweeps illustrate these dependencies (§S6, §S11.2). They motivate compression schedules that respond to correctness and signs of degeneration, with enough flexibility to retain useful intermediate steps on harder problems throughout training.

5.3 Limitations and Future Directions

The empirical success of TokenProbe leaves open why confidence tracks reasoning value and how far this relationship extends beyond the evaluated settings. The signal's interpretation depends on the policy that generates the trace. Repetitive errors can be locally predictable, and the confidence–correctness relationship can reverse after a change in training regime (§S5, Tab. S40). Applying TokenProbe therefore requires revalidating its ability to distinguish productive reasoning from learned repetition. A mechanistic account of when NormLP identifies reasoning-critical content and a theoretical characterization of its reliability remain open (§S7.1).

Extending the framework beyond mathematics also requires examining whether the diagnostic signal and the efficiency gains survive changes in task structure. The reported efficiency transfer and supplementary TokenProbe diagnostic provide evidence across domains, with broader validation still needed (§S7.2). Code generation and agentic reasoning would extend this evaluation to settings where executable steps, external feedback, and longer interactions change the role of intermediate tokens. Such studies could clarify how token value depends on task structure and whether TokenProbe can guide efficient reasoning across different computational demands and feedback structures.

6 Related Work

6.1 Efficient Reasoning for Large Language Models

Efficient reasoning [67] targets the *overthinking* phenomenon [13] in long chain-of-thought, where verbose or redundant intermediate steps inflate inference cost. Existing approaches can be broadly categorized into three paradigms: *model-oriented*, *output-oriented*, and *prompt-oriented*.

Model-oriented methods fine-tune LLMs for intrinsic conciseness, primarily via RL with length-aware rewards [2, 42, 4, 87] or supervised distillation from compressed CoT traces [23, 78], optionally combined with model merging [77, 71]. However, these signals are defined at the response level and thus cannot distinguish which tokens within a trace are genuinely informative, often yielding mild compression or accuracy collapse (see Tab. 2). *Output-oriented* methods rewrite the output format, compressing CoT into latent representations [24, 17, 14] on the view that much of the gain comes from hidden computation rather than verbose text [54, 20]. It normally comes with architectural changes or incurs additional inference cost. *Prompt-oriented* methods enforce concision at the input level through token-budget prompts [50], Chain-of-Draft instructions [80], or difficulty-aware model routing [15, 48]. Their effect is bounded by the underlying model's unchanged policy.

TokenProbe falls within the *model-oriented* paradigm but departs from previous length punishment in two respects. Rather than optimizing a scalar response-length objective, we introduce a *token-level* TokenProbe signal that scores the local importance of every generated token, and inject this signal into both a selective KL anchoring objective (§3.1) and a reward-shaping term (§3.2). This redirects the compression pressure from *how long* the trace is to *which tokens* carry the reasoning signal, enabling strong compression without accuracy collapse in our five-backbone setting (Tab. 2).

6.2 Variants of Group Relative Policy Optimization

Group Relative Policy Optimization (GRPO) [64] has become the dominant critic-free framework [53, 3] for training large reasoning models. Existing variants can be broadly categorized into two paradigms: *reward design*, which governs the granularity and source of the supervision signal used to form group-relative advantages, and *optimization design*, which concerns the policy-gradient objective, importance sampling, loss aggregation, and regularization.

On the *reward design* axis, dense signals [51, 63] shorten the credit-assignment horizon but risk mis-specification and reward hacking. Along action granularity, existing work splits into *token-level* signals derived from policy/reference likelihood ratios, implicit outcome-reward reparameterisation, or token entropy [10, 73, 38, 89, 83, 16], *step-level* signals from trained PRMs [37, 72, 70] or online Monte-Carlo rollouts [12, 45, 43], and *turn-level* signals for agentic tasks [91, 55]. On the *optimization design* axis, critic-free variants refine group-relative normalisation and clipping (DAPO [88], Dr. GRPO [40], LitePPO [41], GFPO [66]), replace token-wise importance ratios with sequence- or geometric-level alternatives (GSPO [97], GMPO [96], CISPO [11]), introduce off-policy or hybrid objectives (TOPR [60], LUFFY [81], Prefix-RFT [31]), and modulate regularisation through KL penalties [99], entropy control [16], and response-level length penalties [4, 69]. However, these length penalties act at the *response* granularity and therefore cannot selectively suppress redundant tokens while preserving reasoning-critical ones.

TokenProbe contributes jointly on both axes. On the *reward-design* side, it replaces response-level length penalties with a *token-level* TokenProbe signal that flags redundant tokens inside every trace. On the *optimization-design* side, it converts this signal into a selective KL anchoring term (§3.1) and a reward-shaping correction (§3.2) that steer GRPO advantages toward informative tokens while leaving reasoning-critical regions untouched. This token-aware coupling resolves the response-level length coupling that limits current variants, and delivers the $2\text{-}4\times$ compression observed in Tab. 2.

7 Conclusion

We present **TokenProbe**, an inference-native token-level diagnostic and optimization framework for efficient Chain-of-Thought reasoning grounded in token value inequality. It has three merits. **i)** It elucidates token value inequality via inference-native NormLP and TokenProbe signals. **ii)** It offers a GRPO-compatible optimization path through selective KL anchoring and reward shaping. **iii)** It demonstrates Pareto-improving token efficiency across the tested backbones and reasoning benchmarks. As a whole, we conclude that the outcomes elucidated in this paper impart essential understandings and thus necessitate further exploration within the field of efficient reasoning.

Acknowledgements

This research was supported by the National Science Foundation under Grant No. 2450068. This work used NCSA Delta GPU through allocation CIS250460 from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program, which is supported by U.S National Science Foundation grants No. 2138259, No. 2138286, No. 2138307, No. 2137603, and No. 2138296. We acknowledge that this work is supported by Purdue Polytechnic Institute. We also gratefully acknowledge the support of RIT Research Computing [52].

References

[1] Marah Abdin, Sahaj Agarwal, Ahmed Awadallah, Vidhisha Balachandran, Harkirat Behl, Lingjiao Chen, Gustavo de Rosa, Suriya Gunasekar, Mojan Javaheripi, Neel Joshi, et al. Phi-4-reasoning technical report. *arXiv preprint arXiv:2504.21318*, 2025.

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

[3] Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce-style optimization for learning from human feedback in llms. In *ACL*, 2024.

[4] Daman Arora and Andrea Zanette. Training language models to reason efficiently. *arXiv preprint arXiv:2502.04463*, 2025.

[5] Niccolo Avogaro, Nayanika Debnath, Li Mi, Thomas Frick, Junling Wang, Zexue He, Hang Hua, Konrad Schindler, and Mattia Rigotti. Sparc: Separating perception and reasoning circuits for test-time scaling of vlms. *arXiv preprint arXiv:2602.06566*, 2026.

[6] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *NeurIPS*, 2015.

[7] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *JMLR*, 2003.

[8] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *ICML*, 2009.

[9] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *NeurIPS*, 2020.

[10] Alex J Chan, Hao Sun, Samuel Holt, and Mihaela Van Der Schaar. Dense reward for free in reinforcement learning from human feedback. *arXiv preprint arXiv:2402.00782*, 2024.

[11] Aili Chen, Aonian Li, Bangwei Gong, Binyang Jiang, Bo Fei, Bo Yang, Boji Shan, Changqing Yu, Chao Wang, Cheng Zhu, et al. Minimax-m1: Scaling test-time compute efficiently with lightning attention. *arXiv preprint arXiv:2506.13585*, 2025.

[12] Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. Alphamath almost zero: process supervision without process. In *NeurIPS*, 2024.

[13] Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, et al. Do not think that much for 2+ 3=? on the overthinking of o1-like llms. *arXiv preprint arXiv:2412.21187*, 2024.

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

[15] Yu-Neng Chuang, Prathusha Kameswara Sarma, Parikshit Gopalan, John Boccio, Sara Bolouki, Xia Hu, and Helen Zhou. Learning to route llms with confidence tokens. *arXiv preprint arXiv:2410.13284*, 2024.

[16] Ganqu Cui, Yuchen Zhang, Jiacheng Chen, Lifan Yuan, Zhi Wang, Yuxin Zuo, Haozhan Li, Yuchen Fan, Huayu Chen, Weize Chen, et al. The entropy mechanism of reinforcement learning for reasoning language models. *arXiv preprint arXiv:2505.22617*, 2025.

[17] Yuntian Deng, Yejin Choi, and Stuart Shieber. From explicit cot to implicit cot: Learning to internalize cot step by step. *arXiv preprint arXiv:2405.14838*, 2024.

[18] Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, et al. Omni-math: A universal olympiad level mathematic benchmark for large language models. *arXiv preprint arXiv:2410.07985*, 2024.

[19] Team Glm, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Dan Zhang, Diego Rojas, Guanyu Feng, Hanlin Zhao, et al. Chatglm: A family of large language models from glm-130b to glm-4 all tools. *arXiv preprint arXiv:2406.12793*, 2024.

[20] Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. Think before you speak: Training language models with pause tokens. *arXiv preprint arXiv:2310.02226*, 2023.

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

[22] Yufei Guo, Yinan Wu, Haoran Duan, Guiguang Ding, and Jungong Han. Phaselora: Control-regime-conditioned low-rank adaptation for continuous-action vision-language-action policies. *arXiv preprint arXiv:2608.15285*, 2026.

[23] Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. Token-budget-aware llm reasoning. In *Findings of ACL*, 2025.

[24] Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769*, 2024.

[25] Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. In *ACL*, 2024.

[26] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

[27] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

[28] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2019.

[29] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Liang Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. In *ICLR*, 2022.

[30] Chen Huang, Wei Lu, and Wenxuan Zhang. Pear: Phase entropy aware reward for efficient reasoning. In *ICLR*, 2025.

[31] Zeyu Huang, Tianhao Cheng, Zihan Qiu, Zili Wang, Yinghui Xu, Edoardo M Ponti, and Ivan Titov. Blending supervised and reinforcement fine-tuning with prefix sampling. *arXiv preprint arXiv:2507.01679*, 2025.

[32] Mingyu Jin, Qinkai Yu, Dong Shu, Haiyan Zhao, Wenyue Hua, Yanda Meng, Yongfeng Zhang, and Mengnan Du. The impact of reasoning step length on large language models. In *Findings of ACL*, 2024.

[33] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[34] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. In *NeurIPS*, 2022.

[35] Zhenglun Kong, Yize Li, Fanhu Zeng, Lei Xin, Shvat Messica, Xue Lin, Pu Zhao, Manolis Kellis, Hao Tang, and Marinka Zitnik. Token reduction should go beyond efficiency in generative models–from vision, language to multimodality. *arXiv preprint arXiv:2505.18227*, 2025.

[36] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *SOSP*, 2023.

[37] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *ICLR*, 2023.

[38] Zhenghao Lin, Zhibin Gou, Yeyun Gong, Xiao Liu, Yelong Shen, Ruochen Xu, Chen Lin, Yujiu Yang, Jian Jiao, Nan Duan, et al. Rho-1: Not all tokens are what you need. *arXiv preprint arXiv:2404.07965*, 2024.

[39] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[40] Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*, 2025.

[41] Zihe Liu, Jiashun Liu, Yancheng He, Weixun Wang, Jiaheng Liu, Ling Pan, Xinyu Hu, Shaopan Xiong, Ju Huang, Jian Hu, et al. Part i: Tricks or traps? a deep dive into rl for llm reasoning. *arXiv preprint arXiv:2508.08221*, 2025.

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

[43] Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Meiqi Guo, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, et al. Improve mathematical reasoning in language models by automated process supervision. *arXiv preprint arXiv:2406.06592*, 2024.

[44] Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Tianjun Zhang, Li Erran Li, et al. Deepscaler: Surpassing o1-preview with a 1.5 b model by scaling rl. *Notion Blog*, 2025.

[45] Qianli Ma, Haotian Zhou, Tingkai Liu, Jianbo Yuan, Pengfei Liu, Yang You, and Hongxia Yang. Let's reward step by step: Step-level reward model as the navigators for reasoning. *arXiv preprint arXiv:2310.10080*, 2023.

[46] Wenjie Ma, Jingxuan He, Charlie Snell, Tyler Griggs, Sewon Min, and Matei Zaharia. Reasoning models can be effective without thinking. *arXiv preprint arXiv:2504.09858*, 2025.

[47] Yingqian Min, Zhipeng Chen, Jinhao Jiang, Jie Chen, Jia Deng, Yiwen Hu, Yiru Tang, Jiapeng Wang, Xiaoxue Cheng, Huatong Song, et al. Imitate, explore, and self-improve: A reproduction report on slow-thinking reasoning systems. *arXiv preprint arXiv:2412.09413*, 2024.

[48] Lester James Validad Miranda, Yizhong Wang, Yanai Elazar, Sachin Kumar, Valentina Pyatkin, Faeze Brahman, Noah A Smith, Hannaneh Hajishirzi, and Pradeep Dasigi. Hybrid preferences: Learning to route instances for human vs. ai feedback. In *ACL*, 2025.

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

[50] Sania Nayab, Giulio Rossolini, Marco Simoni, Andrea Saracino, Giorgio Buttazzo, Nicolamaria Manes, and Fabrizio Giacomelli. Concise thoughts: Impact of output length on llm reasoning and cost. *arXiv preprint arXiv:2407.19825*, 2024.

[51] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, 1999.

[52] Rochester Institute of Technology. Research computing services, 2025.

[53] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. In *NeurIPS*, 2022.

[54] Jacob Pfau, William Merrill, and Samuel R Bowman. Let's think dot by dot: Hidden computation in transformer language models. *arXiv preprint arXiv:2404.15758*, 2024.

[55] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023.

[56] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. *OpenAI*, 2018.

[57] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *NeurIPS*, 2023.

[58] Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*, 2015.

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

[60] Nicolas Le Roux, Marc G Bellemare, Jonathan Lebensold, Arnaud Bergeron, Joshua Greaves, Alex Fréchette, Carolyne Pelletier, Eric Thibodeau-Laufer, Sándor Toth, and Sam Work. Tapered off-policy reinforce: Stable and efficient reinforcement learning for llms. *arXiv preprint arXiv:2503.14286*, 2025.

[61] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[62] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[63] Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. Rewarding progress: Scaling automated process verifiers for llm reasoning. *arXiv preprint arXiv:2410.08146*, 2024.

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

[65] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *EuroSys*, 2025.

[66] Vaishnavi Shrivastava, Ahmed Awadallah, Vidhisha Balachandran, Shivam Garg, Harkirat Behl, and Dimitris Papailiopoulos. Sample more to think less: Group filtered policy optimization for concise reasoning. *arXiv preprint arXiv:2508.09726*, 2025.

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

[68] Guohao Sun, Hang Hua, Jian Wang, Jiebo Luo, Sohail Dianat, Majid Rabbani, Raghuveer Rao, and Zhiqiang Tao. Latent chain-of-thought for visual reasoning. *arXiv preprint arXiv:2510.23925*, 2025.

[69] Kimi Team, Yifan Bai, Yiping Bao, Y Charles, Cheng Chen, Guanduo Chen, Haiting Chen, Huarong Chen, Jiahao Chen, Ningxin Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

[70] Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.

[71] Fanqi Wan, Xinting Huang, Deng Cai, Xiaojun Quan, Wei Bi, and Shuming Shi. Knowledge fusion of large language models. *arXiv preprint arXiv:2401.10491*, 2024.

[72] Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *ACL*, 2024.

[73] Shenzhi Wang, Le Yu, Chang Gao, Chujie Zheng, Shixuan Liu, Rui Lu, Kai Dang, Xionghui Chen, Jianxin Yang, Zhenru Zhang, et al. Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for llm reasoning. *arXiv preprint arXiv:2506.01939*, 2025.

[74] Xuezhi Wang and Denny Zhou. Chain-of-thought reasoning without prompting. In *NeurIPS*, 2024.

[75] Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. In *NeurIPS*, 2024.

[76] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*, 2022.

[77] Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *ICML*, 2022.

[78] Yu Xia, Rui Wang, Xu Liu, Mingyan Li, Tong Yu, Xiang Chen, Julian McAuley, and Shuai Li. Beyond chain-of-thought: A survey of chain-of-x paradigms for llms. In *COLING*, 2025.

[79] Xuan Xiong, Huan Liu, Li Gu, Zhixiang Chi, Yue Qiu, Yuanhao Yu, and Yang Wang. Etr: Entropy trend reward for efficient chain-of-thought reasoning. In *ACL*, 2026.

[80] Silei Xu, Wenhao Xie, Lingxiao Zhao, and Pengcheng He. Chain of draft: Thinking faster by writing less. *arXiv preprint arXiv:2502.18600*, 2025.

[81] Jianhao Yan, Yafu Li, Zican Hu, Zhi Wang, Ganqu Cui, Xiaoye Qu, Yu Cheng, and Yue Zhang. Learning to reason under off-policy guidance. *arXiv preprint arXiv:2504.14945*, 2025.

[82] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[83] An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, et al. Qwen2. 5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024.

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

[85] Ling Yang, Zhaochen Yu, Bin Cui, and Mengdi Wang. Reasonflux: Hierarchical llm reasoning via scaling thought templates. *arXiv preprint arXiv:2502.06772*, 2025.

[86] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *NeurIPS*, 2023.

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

[88] Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

[89] Lifan Yuan, Wendi Li, Huayu Chen, Ganqu Cui, Ning Ding, Kaiyan Zhang, Bowen Zhou, Zhiyuan Liu, and Hao Peng. Free process rewards without process labels. *arXiv preprint arXiv:2412.01981*, 2024.

[90] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. In *NeurIPS*, 2022.

[91] Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. Agenttuning: Enabling generalized agent abilities for llms. In *Findings of ACL*, 2024.

[92] Runjia Zeng, Cheng Han, Qifan Wang, Chunshu Wu, Tong Geng, Lifu Huang, Ying N Wu, and Dongfang Liu. Visual fourier prompt tuning. In *NeurIPS*, 2024.

[93] Runjia Zeng, James Chenhao Liang, Cheng Han, Zhiwen Cao, Jiahao Liu, Xiaojun Quan, Yingjie Victor Chen, Lifu Huang, Tong Geng, Qifan Wang, and Dongfang Liu. Probabilistic token alignment for large language model fusion. In *NeurIPS*, 2025.

[94] Runjia Zeng, Guangyan Sun, Qifan Wang, Tong Geng, Sohail Dianat, Xiaotian Han, Raghuveer Rao, Xueling Zhang, Cheng Han, Lifu Huang, et al. Mept: Mixture of expert prompt tuning as a manifold mapper. In *EMNLP*, 2025.

[95] Runjia Zeng, Qifan Wang, Qiang Guan, Ruixiang Tang, Lifu Huang, Zhenting Wang, Xueling Zhang, Cheng Han, and Dongfang Liu. Tokenseek: Memory efficient fine tuning via instance-aware token ditching. In *ICLR*, 2026.

[96] Yuzhong Zhao, Yue Liu, Junpeng Liu, Jingye Chen, Xun Wu, Yaru Hao, Tengchao Lv, Shaohan Huang, Lei Cui, Qixiang Ye, et al. Geometric-mean policy optimization. *arXiv preprint arXiv:2507.20673*, 2025.

[97] Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, et al. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025.

[98] Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. Agieval: A human-centric benchmark for evaluating foundation models. In *Findings of NAACL*, 2024.

[99] Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

SUMMARY OF THE APPENDIX

This appendix provides additional experimental results and discussion for our NeurIPS 2026 submission, *On the Token Value Inequality in Efficient Reasoning*, organized as follows.

S1 Model Notations . . . 19

S2 Appendix of Analysis . . . 20
- S2.1 The Token Cost Problem in Reasoning . . . 20
- S2.2 Relation to Prior Evidence on Token Non-Uniformity . . . 20
- S2.3 Experiment of TokenProbe . . . 21
  - S2.3.1 TokenProbe for each stage (Fig. 2(a)) . . . 21
  - S2.3.2 TokenProbe for each model (Fig. 2(b)) . . . 21
  - S2.3.3 Quartile analysis (Fig. 2(c)) . . . 22
  - S2.3.4 Sensitivity analysis on Qwen3.5-9B . . . 23
  - S2.3.5 Per-dataset TokenProbe and Correct-versus-Wrong comparison . . . 23
  - S2.3.6 Pearson correlation analysis . . . 24
- S2.4 Case Study of TokenProbe . . . 26
- S2.5 Why TokenProbe? An Ablation over Log-Probability-Derived Signals . . . 30
  - S2.5.1 Setup and notation . . . 30
  - S2.5.2 Overall result. Only TokenProbe satisfies Eq. 7 . . . 31
  - S2.5.3 The inversion is consistent across every benchmark . . . 31
  - S2.5.4 Why TokenProbe survives. Invariance to family-level calibration shift . . . 32
  - S2.5.5 Take-away . . . 32

S3 Appendix of Methodology . . . 34
- S3.1 Core Token Window . . . 34
- S3.2 Exploratory Analysis of Core-Window Selection . . . 34
- S3.3 Selective KL Anchoring (KL) . . . 37
- S3.4 Reward Shaping (RS) . . . 37

S4 Appendix of Experiments . . . 38
- S4.1 Training Settings . . . 38
- S4.2 Evaluation Settings . . . 38
- S4.3 Baselines . . . 40
- S4.4 Capacity-dependent Behavior of KL and RS . . . 41
- S4.5 Reward Dynamics During RL Training . . . 42
- S4.6 Computational Overhead of TokenProbe Operations . . . 42
- S4.7 Hard-Truncation Strawman Baseline . . . 43
- S4.8 Case Studies . . . 44
- S4.9 Scenario Analysis Under an Unlimited Token Budget . . . 53

S5 Unsuccessful Attempts. SFT Distillation Path . . . 55
- S5.1 Experimental Design . . . 55
- S5.2 Results . . . 56
  - S5.2.1 Accuracy . . . 56
  - S5.2.2 Token Usage and Efficiency . . . 56
- S5.3 Conclusion and Analysis . . . 57
  - S5.3.1 Pathological TokenProbe Reversal . . . 57
  - S5.3.2 Degeneration Loop Microstructure . . . 57
  - S5.3.3 Detailed Case Studies . . . 58
  - S5.3.4 Why SFT Fails While RL Succeeds. Structural Analysis . . . 59
- S5.4 Outlook. SFT as RL Warm-Start . . . 60

S6 Exploring Token Compression Limits . . . 61

S7 Limitations and Future Work . . . 62
- S7.1 Theoretical Certification . . . 62
- S7.2 General Reasoning Efficiency . . . 62

S8 Reproducibility, Compute, and Responsible Use Details . . . 63

S8.1 Open Access to Data and Code . . . 63
S8.2 Statistical Significance . . . 63
S8.3 Compute Resources . . . 63
S8.4 Broader Impacts . . . 64
S8.5 Licenses for Existing Assets . . . 64
S8.6 Declaration of LLM Usage . . . 64

S9 Supplementary Validation Experiments . . . 66
S9.1 Within-problem analysis controlling for response-level factors . . . 66
S9.2 Causal interventions on high-NormLP regions . . . 66

S10 Supplementary Closed-API Reasoning Results . . . 67
S10.1 Closed-source models and API prices . . . 67
S10.2 Per-task results . . . 67

S11 Appendix of Tables . . . 70
S11.1 More Results on Reasoning Benchmarks . . . 70
S11.2 Per-task Results on Unlimited Token Budget . . . 70
S11.3 Per-task Results on Out-of-domain Generalization . . . 70

S1 Model Notations

Since every model provider follows a different naming strategy (*e.g.*, Gemini uses "E" for activated parameters, GPT uses only total parameters, and many other models do not identify any parameter size), we adopt a unified naming convention throughout this paper to ensure consistent model identity. For open-source MoE models, we use the format `[Model Name]-[Total Params]-[Activated Params]` (*e.g.*, `Step-Flash-196B-11B`), and for open-source dense models we use `[Model Name]-[Total Params]` (*e.g.*, `Qwen-4B`). We generally omit version numbers for open-source models, as we use relatively recent releases whose identifiers are listed in Tab. S2. For closed-source models, we use the format `[Model Name]-[Version]-[Mode]` (*e.g.*, `Claude-4.6-Sonnet`, `Gemini-3-Flash`).

Table S1: Training models used in Table 2.

Model Name	**Identifier**	**Link**
Qwen-0.6B	`Qwen/Qwen3-0.6B`	HuggingFace
DeepScaleR-1.5B	`agentica-org/DeepScaleR-1.5B-Preview`	HuggingFace
Qwen-4B	`Qwen/Qwen3-4B`	HuggingFace
Phi-Reasoning-4B	`microsoft/Phi-4-mini-reasoning`	HuggingFace
Qwen-8B	`Qwen/Qwen3-8B`	HuggingFace

Table S2: Open-source baseline models used in our experiments. Identifiers follow the HuggingFace `org/model-name` convention. For MoE models, parameters are listed as Total (Active).

Model Name	**HF Identifier**	**Parameters**	**Link**
	Dense Models		
Qwen-4B-Instruct	`Qwen/Qwen3-4B-Instruct-2507`	4B	HuggingFace
Qwen-9B	`Qwen/Qwen3.5-9B`	9B	HuggingFace
DS-Qw-1B	`deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B`	1.5B	HuggingFace
	Mixture-of-Experts (MoE) Models		
GPT-117B-5B	`openai/gpt-oss-120b`	117B (5B)	HuggingFace
GLM-Air-106B-12B	`zai-org/GLM-4.5-Air`	106B (12B)	HuggingFace
GLM-358B-32B	`zai-org/GLM-4.7`	358B (32B)	HuggingFace
GLM-744B-40B	`zai-org/GLM-5`	744B (40B)	HuggingFace
Step-Flash-196B-11B	`stepfun-ai/Step-3.5-Flash`	196B (11B)	HuggingFace
DeepSeek-671B-37B	`deepseek-ai/DeepSeek-V3.2`	671B (37B)	HuggingFace
DeepSeek-1.6T-49B	`deepseek-ai/DeepSeek-V4-Pro`	1.6T (49B)	HuggingFace
DeepSeek-284B-13B	`deepseek-ai/DeepSeek-V4-Flash`	284B (13B)	HuggingFace
Qwen3-Next-80B-3B	`qwen/qwen3-next-80b-a3b-thinking`	80B (3B)	HuggingFace
Kimi-1T-32B	`moonshotai/kimi-k2-thinking`	1.1T (32B)	HuggingFace
Hunyuan3-295B-21B	`tencent/Hy3-preview`	295B (21B)	HuggingFace

Table S3: Closed-source baseline models used in our experiments. Version denotes the model version number. Links point to official announcement pages.

Model Name	**API Identifier**	**Version**	**Mode**	**Link**
Qwen-3.6-Plus	`qwen/qwen3.6-plus`	3.6	Plus	Blog
GPT-5.4-Mini	`openai/gpt-5.4-mini`	5.4	Mini	Blog
Grok-4.1-Fast	`x-ai/grok-4.1-fast`	4.1	Fast	Blog
Gemini-3-Flash	`google/gemini-3-flash-preview`	3	Flash	Blog
Gemini-3.1-Pro	`google/gemini-3.1-pro-preview`	3.1	Pro	Blog
Gemini-3.1-Flash-Lite	`google/gemini-3.1-flash-lite-preview`	3.1	Flash-Lite	Blog
Claude-4.5-Haiku	`anthropic/claude-haiku-4.5`	4.5	Haiku	Blog
Claude-4.6-Sonnet	`anthropic/claude-sonnet-4.6`	4.6	Sonnet	Blog
Claude-4.6-Opus	`anthropic/claude-opus-4.6`	4.6	Opus	Blog

S2 Appendix of Analysis

S2.1 The Token Cost Problem in Reasoning

In our experiments, we use the same Qwen-4B model weights and toggle only the `enable_thinking` flag to isolate the pure contribution of CoT.

Table S4: Pure CoT contribution measured on Qwen-4B.

Mode	Accuracy	Avg. Tokens
CoT ON (`thinking=True`)	**77.1%**	8,728
CoT OFF (`thinking=False`)	53.0%	1,943
Difference	**+24.1 percentage points**	**+6,785 (4.5×)**

As shown in Table S4, CoT contributes $+24.1\%$ of accuracy at the cost of $4.5\times$ token consumption. The marginal token cost per percentage point of accuracy is approximately **281 tokens per percentage point**.

This raises a central question. *Are these* $4.5\times$ *tokens uniformly valuable?* If so, reducing token cost necessarily sacrifices accuracy, and one can only truncate or compress the model. If, however, a substantial fraction of the extra tokens is low-value exploratory redundancy, there exists the possibility of a **Pareto improvement**, preserving core reasoning content while drastically cutting redundant tokens and thereby improving both efficiency and accuracy simultaneously.

S2.2 Relation to Prior Evidence on Token Non-Uniformity

Several recent lines of work provide adjacent evidence that long CoT traces contain tokens with unequal usefulness [35], including overthinking analyses [13, 67], token-level likelihood or reward signals [38, 73], and entropy-based efficient-reasoning objectives [30, 79]. We view these works as complementary evidence for the same broad phenomenon rather than as substitutes for our formulation. The **Token Value Inequality** analysis in this paper gives a different route. It identifies core and redundant tokens inside a single generated trace through NormLP and TokenProbe, using only the generating model's own inference-time log-probabilities.

Common ground. The commonality is that all these methods reject the assumption that every reasoning token contributes equally. Overthinking work shows that response-level verbosity and repetitive detours can hurt efficiency. Rho-style and high-entropy-token analyses study token-level signals during training. PEAR and ETR use entropy dynamics to encourage more efficient reasoning. TokenProbe shares this motivation. It is designed to expose the non-uniform distribution of reasoning value within CoT.

Differences. Our contribution is the concrete identification procedure and its optimization use. First, NormLP is *response-internally normalised*. It compares each token against the confidence baseline of the same response, rather than using an absolute confidence threshold, a reference model, or an external reward model. This makes TokenProbe robust to family-level calibration shifts, as demonstrated by the ablation in Appx. S2.5. Second, TokenProbe operates at token/window granularity, so it can distinguish high-density core reasoning regions from low-density exploratory or repetitive regions inside the same trace. Response-level length penalties and phase-level entropy rewards cannot provide this within-trace mask. Third, the same TokenProbe-derived mask is used by both selective KL anchoring and reward shaping, giving GRPO two complementary handles that preserve core-token behavior while compressing redundant-token behavior.

More concretely, overthinking analyses [13, 67] show that long CoT often contains redundant detours, loops, or unnecessary computation, but their diagnosis is mainly response-level. A trace is too long, repetitive, or inefficient. TokenProbe instead asks which local regions inside the same trace should be preserved or compressed, and therefore supplies a token/window-level view of the same inefficiency problem.

Rho-style token selection [38] also starts from the premise that not all tokens are equally useful, but it typically scores tokens through policy/reference-style likelihood comparisons. TokenProbe

is different in that it is inference-native. The signal is computed from the generating model's own per-token log-probabilities, normalised within each response, without requiring a separate reference model or an external reward model.

TokenSeek [95] studies token non-uniformity from the perspective of fine-tuning memory: it uses instance-aware token selection to reduce activation-related memory costs. This offers a complementary motivation for identifying informative token subsets. Its context- and gradient-based selection targets training memory, whereas TokenProbe uses inference-time log-probabilities to diagnose generated reasoning traces and guide reasoning-length optimization.

High-entropy token analysis [73] further supports the idea that minority token subsets can carry disproportionate learning signal. Our use of TokenProbe is complementary but technically different. Rather than using an absolute entropy or confidence level, we use the centered/scaled shape of the log-probability sequence. This is why the signal remains meaningful across the Qwen→DeepSeek-Distill calibration boundary in Appx. S2.5.

Finally, PEAR [30] and ETR [79] use entropy dynamics to encourage efficient reasoning. PEAR separates the Think and Answer phases, while ETR rewards a response-level entropy trend toward convergence. TokenProbe shares their concern with uncertainty dynamics, but provides a finer mask that can vary within the Think phase itself and can be applied directly to both selective KL anchoring and reward shaping.

Taken together, our Token Value Inequality result consists of more than observing unequal token utility. TokenProbe provides a reproducible, inference-native way to *identify* that inequality at token granularity, prove why the resulting signal survives calibration shifts, and turn the same signal into an efficient GRPO objective.

S2.3 Experiment of TokenProbe

This subsection expands every panel of Fig. 2 into per-model, per-dataset, and per-quartile tables, and provides the supporting sensitivity, fine-grained, and correlation analyses. The model naming follows Fig. 2 whenever applicable.

S2.3.1 TokenProbe for each stage (Fig. 2(a))

For each of the five thinking models in Fig. 2(a), we split every response at the `</think>` boundary and compute TokenProbe separately on the Think segment and the Answer segment.

Table S5: TokenProbe in the Think versus Answer stage. Across all five models, the Answer stage exhibits a systematically higher TokenProbe than the Think stage.

Model	Think TokenProbe	Answer TokenProbe	Δ (Answer$-$Think)
DS-Qw-1B	0.716	**0.915**	$+0.199$
Qwen-0.6B	0.712	**0.823**	$+0.111$
Qwen-4B	0.775	**0.815**	$+0.040$
Qwen-8B	0.784	**0.810**	$+0.026$
Qwen-9B	0.779	**0.878**	$+0.099$

The Answer-stage TokenProbe is uniformly higher than the Think-stage TokenProbe by $+0.026$ to $+0.199$, matching the 3% to 20% gap stated in the main text. The Think stage corresponds to low-confidence exploratory search, whereas the Answer stage corresponds to high-confidence decision output, directly substantiating Finding 1.

S2.3.2 TokenProbe for each model (Fig. 2(b))

For the four models highlighted in Fig. 2(b), Tab. S6 consolidates the response-level TokenProbe, average token usage, and average accuracy. Tab. S7 and Tab. S8 further break the latter two metrics down to every benchmark.

The pattern is consistent across all four benchmarks. A model with a higher TokenProbe simultaneously achieves higher accuracy and lower token usage, providing direct support for Finding 2.

Table S6: TokenProbe, average token usage, and average accuracy for the four representative models in Fig. 2(b).

Model	TokenProbe	Avg Tokens	Avg Accuracy
Qwen-0.6B	72.7%	8,723	43.8%
DS-Qw-1B	74.2%	8,589	54.7%
Qwen-4B	77.4%	6,155	65.7%
Qwen-4B-Instruct	**83.8%**	**3,573**	**71.8%**

Table S7: Per-dataset accuracy of the four models in Fig. 2(b).

Model	AIME2025	AMC	MATH	Olympiad	Overall
Qwen-0.6B	13.3%	45.8%	75.6%	40.6%	**43.8%**
DS-Qw-1B	20.0%	66.3%	85.0%	47.7%	**54.7%**
Qwen-4B	40.0%	74.7%	90.4%	57.8%	**65.7%**
Qwen-4B-Instruct	50.0%	78.3%	92.6%	66.2%	**71.8%**

S2.3.3 Quartile analysis (Fig. 2(c))

We sort the 1,288 samples of each model by their per-response TokenProbe, partition them into four equal quartiles, and report the average TokenProbe, accuracy, and token usage of each quartile. Tabs. S9 to S12 cover the five models Qwen-4B, Qwen-8B, Qwen-9B, DS-Qw-1.5B, and Qwen-0.6B.

Table S9: TokenProbe quartile analysis for Qwen-4B.

Quartile	TokenProbe range	Avg TokenProbe	Accuracy	Avg Tokens
Q1	0.690 to 0.756	0.736	61.5%	10,555
Q2	0.756 to 0.781	0.770	78.3%	8,327
Q3	0.781 to 0.804	0.791	87.0%	6,592
Q4	0.804 to 0.869	0.820	**92.9%**	**4,524**

Table S10: TokenProbe quartile analysis for Qwen-8B.

Quartile	TokenProbe range	Avg TokenProbe	Accuracy	Avg Tokens
Q1	0.682 to 0.766	0.744	60.9%	10,825
Q2	0.766 to 0.789	0.779	77.0%	8,637
Q3	0.789 to 0.811	0.800	86.6%	6,888
Q4	0.811 to 0.990	0.828	**90.4%**	**5,017**

Table S11: TokenProbe quartile analysis for DS-Qw-1.5B.

Quartile	TokenProbe range	Accuracy	Avg Tokens
Q1	≤ 0.688	39.1%	9,623
Q2	0.688 to 0.745	67.7%	7,066
Q3	0.745 to 0.773	71.4%	6,933
Q4	> 0.773	**72.7%**	**6,271**

Table S8: Per-dataset token usage of the four models in Fig. 2(b).

Model	AIME2025	AMC	MATH	Olympiad	Overall
Qwen-0.6B	12,569	8,934	5,064	8,323	**8,723**
DS-Qw-1B	12,256	8,146	4,785	9,169	**8,589**
Qwen-4B	7,696	6,360	4,162	6,404	**6,155**
Qwen-4B-Instruct	5,721	3,216	1,548	3,805	**3,573**

Table S12: TokenProbe quartile analysis for Qwen-0.6B.

Quartile	TokenProbe range	Accuracy	Avg Tokens
Q1	≤ 0.698	32.3%	8,048
Q2	0.698 to 0.731	45.7%	8,310
Q3	0.731 to 0.756	64.3%	6,873
Q4	> 0.756	**73.3%**	**5,554**

The same pattern emerges on every model at the sample level. Moving from the lowest to the highest TokenProbe quartile, accuracy rises monotonically while average token usage drops, exactly as observed in Fig. 2(c) for Qwen-4B and Qwen-8B.

S2.3.4 Sensitivity analysis on Qwen3.5-9B

To verify the intrinsic-property claim of Sec. 2 (that TokenProbe is robust to hyperparameter variations), we vary the context length and sampling temperature of Qwen-9B while keeping the model weights fixed.

Table S13: Qwen-9B under three settings. TokenProbe is stable while accuracy and token usage shift considerably.

Setting	TokenProbe	Avg Tokens	Accuracy
Qwen-9B (8K context, $t = 0.7$)	0.780	7,252	36.8%
Qwen-9B (16K context, $t = 0.7$)	0.782	10,009	46.5%
Qwen-9B (16K context, $t = 1.0$)	0.780	9,303	45.5%

Across the three settings, TokenProbe stays within 0.780 to 0.782, even though accuracy moves from 36.8% to 46.5% and average token usage from $7{,}252$ to $10{,}009$. This stability supports the view that TokenProbe captures an intrinsic, hyperparameter-invariant property of the model rather than a transient artifact of the decoding configuration.

S2.3.5 Per-dataset TokenProbe and Correct-versus-Wrong comparison

Returning to the four models in Fig. 2(b), Tab. S14 reports their per-dataset TokenProbe means, and Tab. S15 contrasts the TokenProbe of correctly and wrongly answered samples within each model.

Table S14: Per-dataset TokenProbe means for the four representative models.

Model	AIME2025	AMC	MATH	Olympiad	Overall
Qwen-0.6B	0.716	0.729	0.739	0.726	**0.727**
DS-Qw-1B	0.732	0.748	0.748	0.740	**0.742**
Qwen-4B	0.764	0.775	0.786	0.765	**0.774**
Qwen-4B-Instruct	0.800	0.830	0.862	0.823	**0.838**

Table S15: TokenProbe of correctly versus wrongly answered samples. Correct samples carry a uniformly higher TokenProbe.

Model	**TokenProbe (Correct)**	**TokenProbe (Wrong)**	Δ	N_{correct}	N_{wrong}
Qwen-0.6B	0.742	0.714	$+0.028$	564	724
DS-Qw-1B	0.757	0.726	$+0.031$	705	583
Qwen-4B	0.782	0.754	$+0.028$	916	372
Qwen-4B-Instruct	0.851	0.793	$+0.059$	990	298

The gap between correct and wrong samples is consistently positive ($+0.028$ to $+0.059$), confirming that TokenProbe encodes correctness signal at the response level. The non-thinking Qwen-4B-Instruct yields the largest gap ($+0.059$). Without an exploratory CoT trace dragging down the response-level confidence, correct answers stand out sharply. Thinking models exhibit smaller gaps (around $+0.028$) because their Think segments inject many low-confidence tokens that compress the contrast.

Applicability to non-thinking architectures. The TokenProbe *signal* is well-defined on any autoregressive model. It depends only on per-token log-probabilities, not on the `<think>`/`</think>` boundary. The training-time compression *objective* that we build on top of it, however, does not transfer to non-thinking checkpoints, and the reason is structural. A non-thinking model already operates near the saturated end of the TokenProbe spectrum. Qwen-4B-Instruct, the non-thinking entry in Tab. S6, sits at TokenProbe $= 0.838$ with an average response of only $3{,}573$ tokens. Its rollouts are essentially all decision-output, with no exploratory low-confidence segment to remove. There is therefore no compression headroom. Any additional training pressure that pushes TokenProbe higher must remove tokens that the model itself was already confident in, i.e., reasoning-essential tokens rather than exploratory redundancy.

Tab. S16 shows what this looks like empirically on Qwen-4B-Instruct, averaged over the four math benchmarks of Tab. 2. Vanilla GRPO collapses the model entirely (accuracy $71.7 \rightarrow 1.3$, tokens $3{,}600 \rightarrow 8{,}136$, saturating the 8K cap with repetitive, content-free output). This collapse is a property of GRPO on instruction-tuned starts, not of our shaping. Our KL variant avoids the runaway expansion but cannot recover the base accuracy. At $\beta = 5{\times}10^{-3}$, $w = 20$ it lands at 52.1 accuracy with $2{,}346$ tokens. At the more aggressive $\beta = 5{\times}10^{-4}$, $w = 20$ it pushes tokens down to 790 but falls to 41.7. The accuracy regression is consistent with the headroom argument. There are no exploratory tokens for KL to drop. We therefore restrict the deployment of our compression objective to thinking-enabled backbones.

Table S16: **Compression objectives applied to a non-thinking backbone.** Qwen3-4B-Instruct is at TokenProbe saturation, so there is no exploratory headroom for the compression objective to suppress. Vanilla GRPO collapses the model entirely. Our KL variant avoids the runaway expansion but cannot recover the base accuracy. Numbers are averages over the four math benchmarks of Tab. 2.

Setting	**Avg. Acc.**	**Avg. Tokens**
Qwen-4B-Instruct (base, non-thinking)	71.7	3,600
Qwen-4B-Instruct + vanilla GRPO	1.3	8,136
Qwen-4B-Instruct + KL ($\beta = 5{\times}10^{-3}$, w=20)	52.1	2,346
Qwen-4B-Instruct + KL ($\beta = 5{\times}10^{-4}$, w=20)	41.7	790

S2.3.6 Pearson correlation analysis

To assess the joint behaviour of TokenProbe with response length and correctness, we compute Pearson correlation coefficients on the five thinking models that span the full range of scales used in our analysis.

Table S17: Pearson correlations between TokenProbe and (i) token length and (ii) accuracy. The first column is uniformly negative. The second is uniformly positive.

Model	r(**TokenProbe**, **Token Length**)	r(**TokenProbe**, **Accuracy**)
DS-Qw-1.5B	-0.227	$+0.288$
Qwen-0.6B	-0.379	$+0.367$
Qwen-4B	-0.415	$+0.307$
Qwen-8B	-0.367	$+0.254$
Qwen-9B	-0.370	$+0.197$

All five models show a negative correlation between TokenProbe and token length (-0.227 to -0.415). The longer a response is, the lower its TokenProbe, indicating that the additional tokens spent on long responses are dominated by below-mean (low-confidence) tokens rather than high-confidence ones. Conversely, all five models show a positive correlation between TokenProbe and accuracy ($+0.197$ to $+0.367$). Higher TokenProbe responses are more likely to be correct. The two correlations together imply that TokenProbe simultaneously reflects reasoning effectiveness (accuracy) and token efficiency, which is precisely the Pareto-friendly diagnostic our framework relies on (Finding 2).

S2.4 Case Study of TokenProbe

We complement the analysis with a fully token-level visualisation of the Qwen-4B reasoning trace summarised in Tab. 1. Each token's background is shaded by its NormLP. Green for above-mean (positive) confidence, red for below-mean (negative) confidence, with intensity log-scaled against the global max-abs across our five case studies. Intensities below a 3% floor are left uncoloured to suppress imperceptible tints. The 17 occurrences of "Wait"/"wait" in this trace are additionally framed with a black border so they can be located at a glance, and the small grey number at the start of each physical line (1 to 80 across the three parts) provides a continuous line counter for cross-referencing.

The visualisation is split into three parts purely for page-layout reasons. Fig. S1 (Part 1/3, lines 1 to 16) covers the bulk of the deliberation, Fig. S2 (Part 2/3, lines 17 to 36) continues through the end of the model's open-ended re-checking, and Fig. S3 (Part 3/3, lines 37 to 80) contains the `**Final Answer**` block, the closing `\boxed{49}`, the `</think>` delimiter, and the model's clean post-think write-up.

- **Token [54]**, early. $\mathrm{NormLP} = +0.47$. Productive "First verification. Discovers misinterpretation". This is the **1st** framed "Wait" in the visualisation, on *line 2* of Fig. S1, opening the clause "*Wait, let me check that again. It says. 'The operation @ is defined as m/n @ p/q = (m)(p)(q/n)...'*". The model has just paraphrased the operator and immediately stops to re-verify it against the original statement, catching the misparsing on the first pass.
- **Token [103]**, early. $\mathrm{NormLP} = +0.22$. Productive "Mild self-correction". The **2nd** framed "Wait", on *line 3* of Fig. S1, opening "*Wait, actually, the problem says. 'The operation @ is defined as (m/n)@(p/q)=(m)(p)...'*". A smaller, second-pass self-check that nudges the operator interpretation closer to the correct form. Both early "Wait"s carry the green (above-mean) tint that Tab. 1 associates with productive self-correction.
- **Token [575]**, mid-late. $\mathrm{NormLP} = -3.87$. Wasteful "Deep confusion. Hesitation". The **9th** framed "Wait", on *line 5* of Fig. S1, opening "*Wait, maybe the second fraction is simplified? So maybe p/q is simplified, but the first fraction m/n might not be?*". By this point the model has already produced a candidate value but now hesitates over the operator's *precondition*, doubting whether the "simplified" qualifier applies to one fraction or both. This is the textbook deep-confusion moment described in the table. The doubt is not about a calculation but about the meta-status of the inputs, and it triggers a long, low-confidence re-derivation rather than a quick fix.
- **Token [1273]**, late. $\mathrm{NormLP} = -4.13$. Wasteful "Re-questions already-confirmed step", $+700$ tokens of additional fruitless exploration. The **12th** framed "Wait", on *line 13* of Fig. S1, opening "*Wait, but the problem says 'simplified value of 7/30 @ 10/21'. So maybe 49 is already simplified?*". The answer 49 has by now been derived and double-checked. This "Wait" re-opens the case purely to re-read the problem statement once more, and is rendered with the deepest red intensity among all 17 framed occurrences. The hundreds of red-tinted tokens that follow it on lines 13 to 16 of Fig. S1 and lines 25 and 32 of Fig. S2 are the "$+700$ tokens" tail mentioned in Tab. 1. The trace finally exits this loop at the `**Final Answer**` block on line 38 (Fig. S3).

The two early "Wait"s (Tokens [54], [103]) sit in green-tinted neighbourhoods, the two late "Wait"s (Tokens [575], [1273]) are embedded in continuous runs of red tokens, and a single low-NormLP "Wait" tends to drag a long tail of low-NormLP downstream tokens with it. This is exactly the asymmetry that motivates TokenProbe. The response-level positive-token ratio captures the productive-vs.-wasteful split in a single scalar, without requiring any token-by-token labelling.

Model: Qwen-4B
Ground Truth: 49
Correct: Yes
Positive Ratio: 0.7402
Prompt: The operation @ is defined as $\frac{m}{n}@\frac{p}{q} = (m)(p)\left(\frac{q}{n}\right)$ for simplified fractions $\frac{p}{q}$. What is the simplified value of $\frac{7}{30}@\frac{10}{21}$? Let's think step by step and output the final answer within \boxed{}.

Response: <think>
Okay, so I need to figure out what the operation @ does here. The problem says that for simplified fractions p/q and m/n, the operation @ is defined as (m/n) @ (p/q) = (m)(p)(q/n). Wait, let me check that again. It says: "The operation @ is defined as m/n @ p/q = (m)(p)(q/n) for simplified fractions p/q." Hmm, maybe I need to parse that more carefully.
Wait, actually, the problem says: "The operation @ is defined as (m/n) @ (p/q) = (m)(p)(q/n) for simplified fractions p/q." Wait, maybe there's a typo in my understanding? Let me check again. The original problem says: "The operation @ is defined as m/n @ p/q = (m)(p)(q/n) for simplified fractions p/q." Wait, maybe the operation is defined between two fractions, m/n and p/q, and the result is m*p*(q/n). So, given two fractions, you take the numerator of the first, multiply by the numerator of the second, then multiply by the denominator of the first divided by the denominator of the second? Wait, no, the definition is (m)(p)(q/n). Let me write that down:
If we have two fractions, say a/b and c/d, then a/b @ c/d = a * c * (d / b). Wait, is that right? Because according to the problem statement, it's m/n @ p/q = (m)(p)(q/n). So m is the numerator of the first fraction, p is the numerator of the second fraction, and q is the denominator of the second fraction, and n is the denominator of the first fraction. So the operation is (m * p * q) / n? Wait, no, Wait, the problem says it's (m)(p)(q/n). So that would be m*p*(q/n). Which is (m*p*q)/n. So, for example, if I have two fractions, say, 1/2 @ 3/4, then it would be 1*3*(4/2) = 1*3*2 = 6. Is that right?
But let me check if I got the definition correctly. The problem says: "The operation @ is defined as m/n @ p/q = (m)(p)(q/n) for simplified fractions p/q." Wait, maybe the second fraction is simplified? So maybe p/q is simplified, but the first fraction m/n might not be? Or is m/n also simplified? The problem says "for simplified fractions p/q". So maybe p/q is simplified, but m/n might not be? Hmm. But in the problem, they are asking for the value of 7/30 @ 10/21. So both fractions are given as simplified fractions? Let me check if 7/30 and 10/21 are simplified.
7 and 30 have a common factor? 7 is prime, 7 doesn't divide 30, so 7/30 is simplified. 10/21: 10 and 21, 10 factors are 2 and 5, 21 is 3 and 7, so no common factors. So both are simplified. So maybe the operation is defined for simplified fractions? Or maybe just that p/q is simplified? The wording is a bit confusing. Let me check again:
"The operation @ is defined as m/n @ p/q = (m)(p)(q/n) for simplified fractions p/q."
So maybe the operation is defined for simplified fractions p/q, but m/n could be any fraction? Or maybe both are simplified? The wording is a bit unclear. But since in the problem, they are giving us two simplified fractions, maybe we don't have to worry about that. So regardless, we can proceed with the definition given.
So given that, we need to compute 7/30 @ 10/21. According to the definition, this is m/n @ p/q where m = 7, n = 30, p = 10, q = 21. Therefore, the result is m*p*(q/n) = 7 * 10 * (21 / 30). Let me compute that step by step.
First, compute 7 * 10. That's 70. Then compute 21 divided by 30. 21/30 simplifies to 7/10. So then, 70 * (7/10). Let me compute that. 70 divided by 10 is 7, times 7 is 49. So the result is 49? Wait, so is that the answer? Let me check again.
Alternatively, maybe I should compute it as fractions. Let me do that. 7 * 10 * (21 / 30). So that's (7 * 10 * 21) / 30. Let me compute numerator and denominator:
Numerator: 7 * 10 * 21. Let me compute 7*10 first, which is 70. Then 70 * 21. Let's do 70*20=1400 and 70*1=70, so total is 1470. So numerator is 1470. Denominator is 30. So 1470 divided by 30. Let's simplify that:
Divide numerator and denominator by 10: 147 / 3. 147 divided by 3 is 49. So yes, same answer, 49. So the result is 49. But wait, 49 is an integer. But the operation is defined for fractions, but maybe the result can be an integer. So is 49 the simplified value? Wait, but the problem says "simplified value of 7/30 @ 10/21". So maybe 49 is already simplified? But 49 is 7 squared, so it's an integer. So the answer is 49. But let me check again if I interpreted the operation correctly.
Wait, let me check the definition again. The operation is defined as m/n @ p/q = (m)(p)(q/n). So that's m*p*(q/n). So that is m*p*q / n. So in this case, m=7, p=10, q=21, n=30. So 7*10*21 / 30. Which is indeed 1470 / 30 = 49. So that's correct.
Alternatively, maybe there is a different interpretation? For example, maybe the operation is (m/n) @ (p/q) = m * p * (q / n). Which is the same thing. So that's 7 * 10 * (21 / 30) = 7*10*(7/10) = 7*7 = 49. So that's the same. So that seems to check out.
Alternatively, maybe the operation is (m/n) @ (p/q) = (m * p) / (n / q)? But that would be different. Wait, but according to the problem statement, the operation is defined as m/n @ p/q = (m)(p)(q/n). So that is m*p*(q/n). Which is the same

Figure S1: Token-level visualization for Case 2: Qwen-4B (Part 1/3)

Model: Qwen-4B
(Continued from Part 1/3 of Case 2)

17 **Response (cont.):** as m*p*q / n. So that is correct.

18 Alternatively, maybe I misread the definition. Let me check again.

19 Original problem: "The operation @ is defined as m/n @ p/q = (m)(p)(q/n) for simplified fractions p/q."

20 So, the operation is m/n @ p/q equals m times p times (q over n). So that is m*p*(q/n). So yes, that's correct.

21 So in this case, 7/30 @ 10/21 is 7*10*(21/30). Which is 7*10*(7/10) = 7*7 = 49. So that's the answer. But maybe I need to check if there's a different way to interpret the operation. For example, maybe the operation is (m/n) @ (p/q) = m * p * (q / n). Which is the same as above.

22 Alternatively, if the operation was defined as (m/n) @ (p/q) = (m * p) / (n / q), which would be m*p*q / n. Which is the same thing. So same result.

23 Alternatively, maybe the operation is (m/n) @ (p/q) = m * p * (q / n). Which is the same as above.

24 So regardless of the interpretation, the result is 49. So maybe 49 is the answer.

25 But let me check once more with numbers. Let me take another example. Suppose I have 1/2 @ 3/4. According to the definition, that would be 1*3*(4/2) = 1*3*2 = 6. Alternatively, if I compute 1*3*4 / 2 = 12 / 2 = 6. So that's correct. So if I take 1/2 @ 3/4 = 6. If I compute 1/2 @ 3/4, is that 6? Let me see. If I think of the operation as multiplying the numerators and then multiplying by the denominator of the second fraction divided by the denominator of the first? Wait, no. Wait, the operation is m/n @ p/q = m*p*(q/n). So that is m*p*q / n. So in this case, m=1, p=3, q=4, n=2. So 1*3*4 / 2 = 12 / 2 = 6. So that's correct. So that seems to be the case.

26 Alternatively, if I take another example. Let's say 2/3 @ 4/5. Then according to the definition, that would be 2*4*(5/3) = 8*(5/3) = 40/3. Which is approximately 13.333... So that seems to be the case. So if I follow that, then in the original problem, the answer is 49. So maybe that's the answer.

27 But let me check if there is a different interpretation. For example, if the operation was defined as (m/n) @ (p/q) = (m * p) / (n / q). But that would be m*p*q / n. Which is same as above. So same result.

28 Alternatively, maybe the operation is defined as m/n @ p/q = (m * p) / (n / q). Which is same as m*p*q / n. So same result.

29 Alternatively, maybe the problem is written with different parentheses? For example, maybe it's m/n @ p/q = m * (p / (n / q))? But that would be m * p * q / n. Which is same as above. So same result.

30 Therefore, regardless of how I parse the operation, it seems that the answer is 49. Therefore, I think that is the correct answer. However, let me check once again with the original problem.

31 Original problem: The operation @ is defined as m/n @ p/q = (m)(p)(q/n) for simplified fractions p/q. So, in the problem, they specify that p/q is simplified. So maybe m/n is not necessarily simplified? But in the problem, we are given 7/30 and 10/21, which are simplified. So maybe that is just a condition for the definition? Maybe the operation is defined for simplified fractions p/q, but m/n could be any fraction? But in our case, both are simplified. So maybe the operation is defined as m/n @ p/q = m*p*(q/n). So, in our case, 7/30 @ 10/21 is 7*10*(21/30) = 7*10*(7/10) = 49. So that's correct.

32 Alternatively, maybe there is a mistake here. Let me check if 7/30 and 10/21 can be simplified? Wait, 7 and 30 have no common factors, 10 and 21 have no common factors. So they are simplified. Therefore, the operation is defined for these fractions. So the answer is 49.

33 Alternatively, maybe the problem expects the answer to be written as a fraction? But 49 is an integer, so it's already simplified. Therefore, the simplified value is 49.

34 Alternatively, maybe I made a mistake in calculation? Let me check again:

35 7 * 10 = 70. Then 21 / 30 is 7/10. Then 70 * 7/10 = (70 / 10) * 7 = 7 * 7 = 49. Correct. Alternatively, 70 * 21 / 30. Let me compute 70 divided by 30 first. 70 / 30 = 7/3. Then 7/3 * 21 = 7 * 7 = 49. Same result. So that's correct.

36 Therefore, I think the answer is 49. Therefore, the simplified value is 49.

Figure S2: Token-level visualization for Case 2: Qwen-4B (Part 2/3)

Model: Qwen-4B
(Continued from Part 2/3 of Case 2)

Response (cont.):

Final Answer
\boxed{49}
</think>

To solve the problem, we are given a custom operation defined as:

$$
\frac{m}{n} @ \frac{p}{q} = m \cdot p \cdot \left(\frac{q}{n} \right)
$$

This means the operation takes two fractions, multiplies their numerators, and multiplies the result by the denominator of the first fraction divided by the denominator of the second.

We are asked to compute:

$$
\frac{7}{30} @ \frac{10}{21}
$$

—

Step 1: Identify the components

From the definition, we assign:
- $ m = 7 $
- $ n = 30 $
- $ p = 10 $
- $ q = 21 $

Substitute into the formula:

$$
\frac{7}{30} @ \frac{10}{21} = 7 \cdot 10 \cdot \left(\frac{21}{30} \right)
$$

—

Step 2: Simplify the expression

First, compute the multiplication of the numerators:

$$
7 \cdot 10 = 70
$$

Now compute the fraction:

$$
\frac{21}{30} = \frac{7}{10}
$$

Now multiply:

$$
70 \cdot \frac{7}{10} = \frac{70 \cdot 7}{10} = \frac{490}{10} = 49
$$

—

Step 3: Final Answer

The operation yields an integer, and since the problem asks for the **simplified value**, the result is already in its simplest form.

$$
\boxed{49}
$$<lim_endl>

Figure S3: Token-level visualization for Case 2: Qwen-4B (Part 3/3)

S2.5 Why TokenProbe? An Ablation over Log-Probability-Derived Signals

A natural question is whether the diagnostic value of TokenProbe is generic to any statistic derived from log-probability. If so, simpler alternatives such as the average log-probability or the per-step entropy might serve equally well. We answer this question with a controlled ablation. We re-run the four-model ladder of Tab. S6 and, for every response, record *all* commonly used log-probability and entropy descriptors. We then test, for each candidate signal M, whether the implication

$$M \uparrow \implies \text{Accuracy} \uparrow \ \wedge \ \text{Avg. Tokens} \downarrow \tag{7}$$

holds along the ladder. *Only TokenProbe survives this test.* All absolute-confidence and absolute-entropy signals invert at the smallest-model boundary, which we trace to a family-level calibration shift between the Qwen3 and the DeepSeek-R1-Distill model lines.

S2.5.1 Setup and notation

Let a generated response consist of T tokens. For each step $t \in \{1, \dots, T\}$ let

- $\ell_t = \log p_\theta(y_t \mid y_{<t}, x) \in \mathbb{R}_{\le 0}$ be the natural log-probability of the sampled token.
- $p_t = \exp(\ell_t) \in (0, 1]$ the corresponding probability.
- $\Pi_t = \{p_t^{(1)}, \dots, p_t^{(K)}\}$ with $K = 20$ the renormalised top-K probabilities at step t.

Sample mean and standard deviation of $\{\ell_t\}$ are denoted $\mu = \frac{1}{T}\sum_t \ell_t$ and $\sigma = \sqrt{\frac{1}{T}\sum_t (\ell_t - \mu)^2}$, respectively, and $z_t = (\ell_t - \mu)/\sigma$. We use `vLLM` `logprobs=`$20$ to materialise Π_t, dtype `bfloat16`, temperature $= 0.7$, top-p $= 0.95$. Accuracy and average token usage are reproduced from Tab. S6. All other numerical entries below are produced by this ablation.

For every response we then compute the descriptors in Tab. S18.

Table S18: Every per-response descriptor evaluated in the ablation. Per-dataset / overall numbers are obtained by averaging the per-response value across responses.

Descriptor	**Per-response definition**
Log-probability distribution (absolute level)	
Mean log-prob	$\mu = \frac{1}{T}\sum_{t=1}^{T} \ell_t$
Std / Var log-prob	σ, σ^2
Median / Min / Max log-prob	$\mathrm{median}(\{\ell_t\})$, $\min_t \ell_t$, $\max_t \ell_t$
Quartiles, IQR	$q_{0.25}$, $q_{0.75}$, $q_{0.75} - q_{0.25}$
Skewness log-prob	$\frac{1}{T}\sum_t (\ell_t - \mu)^3/\sigma^3$
Excess kurtosis log-prob	$\frac{1}{T}\sum_t (\ell_t - \mu)^4/\sigma^4 - 3$
Sum log-prob	$\sum_{t=1}^{T} \ell_t$
Perplexity	$\exp(-\mu)$
Geometric / arithmetic mean prob	$\exp(\mu)$, $\frac{1}{T}\sum_t p_t$
High-confidence ratio	$\frac{1}{T}\sum_t \mathbf{1}[\ell_t > -0.5]$
Low-confidence ratio	$\frac{1}{T}\sum_t \mathbf{1}[\ell_t < -2]$
Z-score / shape family (response-internally normalised)	
TokenProbe (this paper)	$\frac{1}{T}\sum_t \mathbf{1}[z_t > 0]$
$z > 1$ ratio	$\frac{1}{T}\sum_t \mathbf{1}[z_t > 1]$
$z < -1$ ratio	$\frac{1}{T}\sum_t \mathbf{1}[z_t < -1]$
Mean $\lvert z \rvert$	$\frac{1}{T}\sum_t \lvert z_t \rvert$
Token-step entropy (top-K truncation, $K = 20$)	
Step entropy	$H_t = -\sum_{k=1}^{K} p_t^{(k)} \log p_t^{(k)}$ (nats)
Mean / Std / Var entropy	$\frac{1}{T}\sum_t H_t$, etc., over $\{H_t\}$
Median / Min / Max entropy	idem
Skewness / kurtosis entropy	idem
Normalised entropy	$\frac{1}{T}\sum_t H_t \big/ \log K$

We stress that all signals listed above are derived purely from the log-probabilities returned by the inference engine. None requires labels or external scoring. The shape/Z-score family in particular is defined relative to the per-response mean and variance, which makes them invariant under additive shifts of $\{\ell_t\}$. As we will see, this invariance is the decisive property.

S2.5.2 Overall result. Only TokenProbe satisfies Eq. 7

Tab. S19 reports the overall (pooled across the four datasets) value of the most representative descriptors for the four-model ladder.

Table S19: Overall metric values across the four-model ladder of Fig. 2(b). Only TokenProbe and its shape-family relatives ($z < -1$ ratio, mean $|z|$, log-prob skewness/kurtosis) move monotonically with accuracy. Every absolute-level signal (mean/sum log-prob, perplexity, geometric-/arithmetic-mean prob, high/low-confidence ratio, mean/normalised entropy, std log-prob) inverts at the Qwen-0.6B→DS-Qw-1B step.

Signal (overall)	**Qwen-0.6B**	**DS-Qw-1B**	**Qwen-4B**	**Qwen-4B-Instruct**
Accuracy	43.8%	54.7%	65.7%	**71.8%**
Avg. Tokens	8,723	8,589	6,155	**3,573**
TokenProbe	0.727	0.742	0.774	**0.838**
$z < -1$ ratio	0.1267	0.1255	0.1089	**0.0812**
Mean $\|z\|$	0.6892	0.6281	0.6813	**0.5187**
Skewness log-prob	-2.577	-3.232	-2.668	$\mathbf{-4.723}$
Excess kurtosis log-prob	7.504	8.233	12.273	**29.219**
Mean log-prob	-0.2741	$\mathbf{-0.2978}$	-0.1719	-0.1137
Sum log-prob	-2129.8	$\mathbf{-2409.4}$	-1021.1	-474.0
Perplexity	1.319	**1.353**	1.189	1.124
Geo. mean prob	0.7621	**0.7456**	0.8430	0.8951
Arith. mean prob	0.8300	**0.8258**	0.8870	0.9267
Std log-prob	0.4954	**0.5542**	0.3779	0.3044
High-conf. ratio ($\ell > -0.5$)	0.8078	**0.7990**	0.8794	0.9220
Low-conf. ratio ($\ell < -2$)	0.0183	**0.0273**	0.0074	0.0060
Mean entropy (nats)	0.4150	**0.4415**	0.2514	0.1705
Normalised entropy	0.1385	**0.1474**	0.0839	0.0569

To remove any ambiguity we sort the four models by each candidate signal and read off the corresponding accuracy sequence. A valid proxy must produce a strictly increasing accuracy sequence. Only TokenProbe (and its shape-family relatives) achieves this.

S2.5.3 The inversion is consistent across every benchmark

The Qwen-0.6B→DS-Qw-1B inversion is not a pooling artefact. Tab. S20 and Tab. S21 report mean log-probability and perplexity on each of the four benchmarks. In every column DS-Qw-1B is *less* confident than Qwen-0.6B despite being more accurate. Tab. S22 repeats the picture for mean step entropy. DS-Qw-1B is uniformly more uncertain than Qwen-0.6B at every benchmark.

Table S20: Per-dataset mean log-probability μ. Higher (closer to 0) indicates higher absolute confidence. The DS-Qw-1B row is uniformly more negative than Qwen-0.6B despite the $+10.9$ percentage points accuracy gap.

Model	**AIME2025**	**AMC**	**MATH**	**Olympiad**	**Overall**
Qwen-0.6B	-0.2925	-0.2674	-0.2567	-0.2871	-0.2741
DS-Qw-1B	-0.3413	-0.2807	-0.2836	-0.3084	-0.2978
Qwen-4B	-0.1932	-0.1729	-0.1535	-0.1845	-0.1719
Qwen-4B-Instruct	-0.1584	-0.1202	-0.0825	-0.1340	-0.1137

Table S21: Per-dataset perplexity $\exp(-\mu)$. Lower indicates higher absolute confidence. DS-Qw-1B is again *worse* than Qwen-0.6B on every dataset.

Model	AIME2025	AMC	MATH	Olympiad	Overall
Qwen-0.6B	1.3420	1.3088	1.2953	1.3362	1.3187
DS-Qw-1B	1.4144	1.3286	1.3324	1.3679	1.3526
Qwen-4B	1.2140	1.1895	1.1669	1.2041	1.1890
Qwen-4B-Instruct	1.1748	1.1298	1.0880	1.1472	1.1238

Table S22: Per-dataset mean step entropy $\bar{H}$ (nats, top-K=20 truncation). Lower indicates more decisive next-token distributions. DS-Qw-1B is uniformly more uncertain than Qwen-0.6B although it is more accurate.

Model	AIME2025	AMC	MATH	Olympiad	Overall
Qwen-0.6B	0.4422	0.4049	0.3888	0.4344	0.4150
DS-Qw-1B	0.5014	0.4170	0.4222	0.4561	0.4415
Qwen-4B	0.2802	0.2520	0.2247	0.2698	0.2514
Qwen-4B-Instruct	0.2365	0.1798	0.1237	0.2011	0.1705

The same per-dataset inversion is observed for high-confidence ratio, low-confidence ratio, normalised entropy, sum log-probability, geometric / arithmetic mean probability, and standard deviation of log-probability, which are omitted for brevity. In contrast, TokenProbe is uniformly higher for DS-Qw-1B than for Qwen-0.6B on every benchmark (see Tab. S6 and the TokenProbe entries of the released artefact).

S2.5.4 Why TokenProbe survives. Invariance to family-level calibration shift

The DS-Qw-1B model is a R1-distilled Qwen variant. Its output distribution is systematically flatter than the same-generation Qwen3 backbones. Every per-token uncertainty is intrinsically larger, even though the resulting reasoning chain is more accurate. We model this discrepancy as an additive shift on the per-token log-probabilities. $\tilde{\ell}_t = \ell_t + c$, where c depends on the model family. Under such a shift,

$$\begin{aligned} \tilde{\mu} &= \mu + c, & \tilde{\sigma} &= \sigma, & \tilde{z}_t &= z_t, \\ \widetilde{\mathrm{PPL}} &= e^{-c} \cdot \mathrm{PPL}, & \sum_t \tilde{\ell}_t &= \sum_t \ell_t + Tc, & \mathbf{1}[\tilde{\ell}_t > \tau] &\neq \mathbf{1}[\ell_t > \tau]. \end{aligned} \tag{8}$$

Mean log-probability, sum log-probability, perplexity, geometric/arithmetic mean probability, and the threshold-based high/low-confidence ratios all transform non-trivially under Eq. 8. Entropy is unaffected by a shift in the sampled-token log-prob but inherits a similar family-level offset because H_t depends on the renormalised tail of the distribution. By contrast,

$$\textsc{TokenProbe} \;=\; \frac{1}{T}\sum_{t=1}^{T} \mathbf{1}[z_t > 0]$$

is a function of the centred / scaled sequence $\{z_t\}$ only, and therefore invariant under both additive ($\ell_t \mapsto \ell_t + c$) and multiplicative ($\ell_t \mapsto a\ell_t$) re-scaling of the underlying log-probabilities. The same is true of the $z < -1$ ratio and mean $|z|$. This invariance is precisely why the shape family continues to track reasoning quality across the Qwen→DeepSeek family boundary, while every absolute-level descriptor fails.

S2.5.5 Take-away

Among the broad family of signals derived from log-probability and entropy, only those that normalise within a response carry the Pareto-friendly diagnostic that we ascribed to TokenProbe in Sec. 2. Absolute-level signals, including mean and sum log-probability, perplexity, geometric / arithmetic mean probability, threshold-based high/low-confidence ratios, mean and normalised step entropy, and

standard deviation of log-probability, are all sensitive to family-level calibration shifts and invert at the Qwen→DeepSeek-Distill boundary. The shape-family descriptors that do survive (TokenProbe, $z < -1$ ratio, mean $|z|$, log-prob skewness, log-prob excess kurtosis) are all simple functionals of the standardised sequence $\{z_t\}$ and convey essentially the same information. TokenProbe in particular is interpretable as "what fraction of tokens does this response spend above its own per-response confidence baseline?" and is therefore the most natural choice among the equivalent shape statistics. The full per-model, per-dataset, and per-metric numbers are released alongside the paper.

S3 Appendix of Methodology

S3.1 Core Token Window

Given a generated response, we first standardise its per-token log-probabilities into the NormLP signal z_t, so that the comparison is made within the response itself rather than across responses with different absolute confidence levels. We then slide a window of size W over $\{z_t\}$ and compute the cumulative score of every window. The K highest-scoring *non-overlapping* windows are selected greedily, and the tokens inside them are marked as core tokens ($CT_t = 1$). The rest are treated as redundant ($CT_t = 0$). Using contiguous windows, rather than ranking individual tokens, preserves local semantic coherence and yields a binary core mask without any external annotation.

Listing 1: PyTorch-style pseudocode for core-token window selection.

```
procedure core_token_mask(logp, W, K)
    # logp is per-token log-prob of one rollout, shape [T]
    # W is window size. K is number of core windows
    T = logp.shape[0]

    # 1) NormLP standardises log-probs within this response
    z = (logp - logp.mean()) / (logp.std() + eps)

    # 2) Cumulative score for every window of length W
    scores = sliding_window_sum(z, W)            # shape [T - W + 1]

    # 3) Greedy non-overlapping top-K window selection
    mask = torch.zeros(T)
    repeat K times
        i = scores.argmax().item()
        mask[i .. i + W] = 1
        # forbid any window that would overlap with [i, i+W)
        scores[max(0, i - W + 1) .. i + W] = -inf
    return mask     # 1 = core token, 0 = redundant token
```

S3.2 Exploratory Analysis of Core-Window Selection

We provide an exploratory analysis of the core-window heuristic to better understand how its design choices affect spatial coverage, optimization quality, window-scale sensitivity, and semantic content. All mask-level diagnostics below are computed on 904 Qwen-4B rollouts from AIME and AMC, with eight samples per problem and the core fraction kept approximately constant when varying W.

Coverage behavior. Non-overlapping selection changes how the selected tokens are spatially allocated along a reasoning trace. In Tab. S23, the non-overlapping variant covers a broader span of each response and leaves fewer 100-token intervals without any selected core token. This suggests that the non-overlapping constraint helps distribute the core-token budget across multiple high-density regions instead of concentrating it around a single local peak.

Table S23: Exploratory comparison of non-overlapping and overlapping core-window selection at W=10 and core fraction $\approx 20\%$. Desert100 denotes the fraction of 100-token sliding intervals containing no selected core token.

Dataset	Selection	Coverage	Desert100
AIME	Non-overlap	0.993 ± 0.012	0.221 ± 0.074
	Overlap	0.882 ± 0.145	0.720 ± 0.074
AMC	Non-overlap	0.989 ± 0.019	0.207 ± 0.078
	Overlap	0.886 ± 0.136	0.685 ± 0.099

Greedy behavior relative to dynamic programming. As a diagnostic, we compare the greedy implementation with a dynamic-programming optimum for selecting top-K non-overlapping windows. The token-level masks are not identical. Tab. S24 shows Jaccard values around 0.83 and

F1 around 0.91. However, the selected cumulative score is very close to the DP optimum, with a relative score gap below 0.2% on both datasets. This pattern suggests that many near-equivalent high-score windows exist in long traces, so small token-level mask differences can have little effect on the cumulative NormLP objective.

Table S24: Greedy non-overlapping selection compared with the DP optimum at $W{=}10$. Score gap is $(S_{\mathrm{DP}} - S_{\mathrm{greedy}})/S_{\mathrm{DP}}$.

Dataset	Jaccard	F1	Score gap	Time / rollout
AIME	0.837 ± 0.042	0.911 ± 0.025	$0.19\% \pm 0.28$	6.41 ms / 31.01 ms
AMC	0.831 ± 0.050	0.907 ± 0.032	$0.15\% \pm 0.21$	3.72 ms / 17.38 ms

Window-scale sensitivity. We examine window size at two levels. First, a training-level sweep on the KL variant shows that downstream performance is similar for $W{=}10$ through $W{=}30$, while $W{=}5$ acts as a small-window stress case. It produces shorter responses but lower accuracy (Tab. S25). This is consistent with the interpretation that overly small windows can fragment local reasoning units.

Table S25: Training-level window-size sensitivity for the KL variant on Qwen-0.6B. Numbers are averages over the four math benchmarks.

	$W{=}5$	$W{=}10$	$W{=}20$	$W{=}30$
Avg. Accuracy	38.4	42.3	41.4	42.2
Avg. Tokens	1,752	2,023	2,008	2,044

Second, we use mask-level diagnostics to separate two kinds of stability. The range $W{=}5$ to 12 probes local behavior around the default $W{=}10$. $W{=}9$ and $W{=}11$ are closest to the default mask, $W{=}12$ is moderate, and $W{=}5$ is visibly unstable. The range $W{=}20$ to 30 probes a coarser regime. These masks drift from the $W{=}10$ mask, but adjacent window sizes within the coarse regime change smoothly, with adjacent Jaccard increasing from 0.798 at $W{=}20$ to 0.841 at $W{=}30$ (Tab. S26). This adjacent-scale stability is consistent with the stable training result at $W{=}30$.

Table S26: Mask-level window-size diagnostics. $J(M_W, M_{10})$ measures similarity to the default mask. Adjacent J measures $J(M_W, M_{W-1})$. Values are averaged over AIME and AMC rollouts.

Local range around $W{=}10$			**Coarse range** $W{=}20$ **to** 30		
W	$\mathbf{J}(\mathbf{M_W}, \mathbf{M_{10}})$	**Adjacent** J	W	$\mathbf{J}(\mathbf{M_W}, \mathbf{M_{10}})$	**Adjacent** J
5	0.448	,	20	0.428	0.798
6	0.460	0.615	21	0.422	0.804
7	0.497	0.641	22	0.415	0.809
8	0.567	0.662	23	0.408	0.814
9	0.700	0.680	24	0.403	0.817
10	1.000	0.700	25	0.399	0.825
11	0.714	0.714	26	0.394	0.828
12	0.601	0.731	27	0.390	0.829
			28	0.386	0.833
			29	0.383	0.840
			30	0.380	0.841

Semantic content probe. Finally, we run a small semantic probe on 20 stratified rollouts, balanced across AIME and AMC and across correct and incorrect responses: five correct and five incorrect rollouts per dataset. For each rollout, we annotate the five highest-scoring core windows and the five lowest-scoring anti-mask windows, giving 200 annotated windows in total. Each annotation row records the dataset, rollout identifier, correctness, mask side, cumulative NormLP score, token span, the exact 10-token window, a short prefix and suffix, and one dominant semantic role. The role is

assigned by a majority rule over the 10-token window using five coarse categories: symbolic content, decisive reasoning, formula/theorem invocation, boilerplate connective, and hedging/repetition.

Representative examples are shown in Tab. S27. The core windows tend to be algebraic or enumerative fragments, such as S_{even} constraints, subset listings, binomial/logarithmic expansions, or local equations. The anti-mask windows are more often meta-reasoning or repair text, such as "Wait", "maybe", "let me check", and repeated verification. We also observe a small number of content-bearing anti-mask windows, so the probe should be read as a distributional tendency rather than a hard semantic partition.

Table S27: Representative excerpts from the manually annotated semantic probe. Scores are cumulative NormLP values over the selected 10-token window, and spans are token positions within the rollout.

Data	Mask	Score / span	Role	Excerpt around selected window
AIME	Core	4.9139 / 455–464	Symbolic content	"$= 36 - D \Rightarrow S_{\text{even}} =$", inside the derivation of odd/even digit sums.
AIME	Core	4.5124 / 4141–4150	Symbolic content	"$2^{15} - 2^{12} - 2^{10} + 2^8$", inside an inclusion-exclusion count.
AMC	Core	3.5591 / 1056–1065	Symbolic content	"$+3y + 3y^2$", inside the expansion of $(1 + y)^3$.
AIME	Anti	-18.0658 / 3292–3301	Hedging/repetition	"Wait, is there a way that this answer could be wrong?"
AMC	Anti	-16.2504 / 776–785	Hedging/repetition	"Hmm, that seems complicated. Maybe there is another approach."
AMC	Anti	-18.6124 / 1876–1885	Formula/theorem	"applying logarithmic identities and simplifying the terms", a content-bearing anti-mask example.

Aggregating all 200 annotations gives the summary in Tab. S28. In this sample, high-scoring windows are dominated by symbolic or formula-level content, while low-scoring windows contain more hesitation and connective filler.

Table S28: Semantic-role distribution of core versus anti-mask windows in a 20-rollout probe. Content is the sum of symbolic, decisive-reasoning, and formula/theorem categories. Filler is the sum of boilerplate and hedging/repetition categories.

	AIME-2025		AMC	
Category	**Core**	**Anti**	**Core**	**Anti**
Symbolic content	98.0	4.0	98.0	4.0
Decisive reasoning	0.0	22.0	2.0	32.0
Formula / theorem	2.0	2.0	0.0	2.0
Boilerplate connective	0.0	14.0	0.0	6.0
Hedging / repetition	0.0	58.0	0.0	56.0
Content	100.0	28.0	100.0	38.0
Filler	0.0	72.0	0.0	62.0

This probe is not intended as a complete semantic taxonomy of reasoning traces, but it supports the intended interpretation of high-NormLP windows as content-bearing regions rather than merely connective boilerplate.

S3.3 Selective KL Anchoring (KL)

The intuition behind KL (Eq. 4) is to make the regulariser *token-aware*. Core tokens keep the standard KL anchor so that reasoning-essential positions remain stable. Redundant tokens, on the other hand, are only pressured *after* the rollout has already overrun its length budget. At that point we drop the KL anchor and inject an overflow-proportional log-probability penalty, encouraging the policy to stop emitting more redundant tokens. Within-budget redundant tokens are left untouched so that the model is not punished for moderate verbosity.

Listing 2: PyTorch-style pseudocode for the Selective KL Anchoring loss.

```
procedure loss_KL(ratio, A, kl, logp, CT, T, L_limit,
                  beta_base, lambda_decay)
    # ratio is pi_theta / pi_old             [T]
    # A is token-level advantage             [T]
    # kl is KL(pi_theta || pi_ref)           [T]
    # logp is log pi_theta(a_t | s_t)        [T]
    # CT is core mask, 1 = core token        [T]
    # T is rollout length
    # L_limit is current length budget

    beta  = torch.full_like(ratio, beta_base)   # default KL weight
    gamma = torch.zeros_like(ratio)             # default sparsity weight

    # over-budget redundant tokens. Drop KL, add overflow-proportional penalty
    overflow_token = (CT == 0) & (T > L_limit)
    beta  = torch.where(overflow_token, 0.0, beta)
    gamma = torch.where(overflow_token,
                        lambda_decay * (T - L_limit) / T,
                        gamma)
    # core tokens & within-budget redundant tokens fall back to (beta_base, 0)

    return (ratio * A - beta * kl - gamma * logp).mean()
```

S3.4 Reward Shaping (RS)

RS leaves the regulariser of vanilla GRPO untouched and instead reshapes the token-level advantage $\hat{A}_t$ on *correct* rollouts (Eq. 6). Two effects are added on top of $\hat{A}_t$. (i) core tokens receive a positive bonus proportional to their NormLP score, so high-density reasoning steps on a correct trace are upweighted. (ii) once the rollout overruns its budget, redundant tokens are penalised in proportion to the overflow ratio. Wrong-answer rollouts keep the vanilla advantage so that we do not propagate misleading credit to tokens that should be re-examined.

Listing 3: PyTorch-style pseudocode for the Reward Shaping loss.

```
procedure loss_RS(ratio, A, kl, normlp, CT, r_seq, T, L_limit,
                  beta_base, alpha, lam, beta)
    # ratio is pi_theta / pi_old             [T]
    # A is token-level advantage             [T]
    # kl is KL(pi_theta || pi_ref)           [T]
    # normlp is per-token NormLP score       [T]
    # CT is core mask, 1 = core token        [T]
    # r_seq is sequence-level reward

    # bonus on core tokens. Penalty on redundant tokens once over budget
    core_bonus = alpha * normlp * (CT == 1)
    overflow   = max(0.0, T - L_limit) / T
    red_pen    = lam * overflow * (CT == 0)

    A_shaped = A + beta * (core_bonus - red_pen)

    # only correct rollouts get the reshaped advantage
    correct = (r_seq > 0)
    A_RS = torch.where(correct, A_shaped, A)

    return (ratio * A_RS - beta_base * kl).mean()
```

S4 Appendix of Experiments

S4.1 Training Settings

We train on the **DeepScaleR** [44] dataset, a curated collection of approximately 40,000 challenging mathematics problem to answer pairs compiled from four distinct sources. We retain only problems whose prompt fits within 1,024 tokens, yielding the final training corpus. During training, the maximum rollout length is **4,096 tokens**.

Dataset Composition.

- **AIME (1984 to 2023).** Problems from the American Invitational Mathematics Examination spanning four decades. These are among the hardest pre-olympiad competition problems, requiring deep algebraic, combinatorial, and number-theoretic reasoning. *Example*. "A cyclic pentagon $ABCDE$ has a right angle $\angle ABC = 90^\circ$ and side lengths $AB = 15$, $BC = 20$. Find the length CD." (Answer. 20.)
- **AMC (prior to 2023).** Problems from the American Mathematics Competitions (AMC 10 / AMC 12) covering high-school topics in a multiple-choice format. *Example*. "Let m/n be the reduced fraction equal to $3 + \frac{1}{3+\frac{1}{3+\frac{1}{3}}}$. What is $m + n$?" (Answer. 142.)
- **Omni-MATH** [18]. A large-scale olympiad problem bank drawn from international and national olympiads, spanning number theory, geometry, combinatorics, and algebra at olympiad difficulty. *Example*. "Petya and Vasya each construct ten fifth-degree polynomials. In how many ways can they choose the polynomials such that the sum of any one of Petya's and any one of Vasya's polynomials has no real roots?" (Answer. 50.)
- **STILL** [47]. A curated collection of synthesis problems designed to push model reasoning beyond standard competition formats, including counting, expected value, and mixed-domain problems. *Example*. "How many sequences of ten binary digits exist in which neither two consecutive zeros nor three consecutive ones appear?" (Answer. 28.)

Training Recipe. All runs use the veRL framework [65] with GRPO [64]. We adopt train / validation batch sizes of 64 / 512, a PPO mini-batch size of 64 with dynamic batching (up to 20,480 tokens per GPU), a maximum prompt length of 1,024 tokens, and a maximum rollout length of 4,096 tokens. The actor is optimised at learning rate $1\mathrm{e}{-6}$ with a low-variance KL loss of coefficient 10^{-3} and an equal KL-control coefficient of 10^{-3}. Rollouts are generated by vLLM with n=16 samples, temperature 0.6, in bfloat16 with FlashAttention-2. FSDP parameter and optimiser offload together with gradient checkpointing are enabled to fit training in GPU memory. Each run is one epoch over the DeepScaleR training split on a single node with 8 GPUs, with tensor-parallel size 1 for models up to 1.5B and 2 for 3B to 8B models. Checkpoints are saved every 50 steps and evaluation on the AIME validation split is run at the same cadence.

S4.2 Evaluation Settings

Same-Domain Benchmarks (Main Results). Following standard practice in efficient reasoning research, we evaluate on four mathematical reasoning benchmarks.

- **AIME 2025** (30 problems). The 2025 American Invitational Mathematics Examination. Problems require multi-step algebraic and number-theoretic reasoning without multiple choice. *Example*. "Find the sum of all integer bases $b > 9$ for which 17_b divides 97_b." (Answer. 70.)
- **AMC** (83 problems). Multiple-choice problems from recent American Mathematics Competitions, testing computational fluency and combinatorial thinking. *Example*. "Thirteen cards numbered 1 to 13 are arranged in a row and swept up in increasing order by repeated left-to-right passes. For how many initial arrangements does this take exactly 2 passes?" (Answer. 8,178.)
- **MATH-500** [27, 37] (500 problems). A 500-problem representative subset of the Hendrycks MATH benchmark, spanning seven difficulty levels across algebra, geometry, number theory, and probability. *Example*. "Find all integer values of x such that $x^2 - 5x + 6 = 0$." (Answers. $x = 2$ and $x = 3$.)
- **OlympiadBench** [25] (675 problems). Bilingual (English/Chinese) olympiad-level problems drawn from national and international competitions, spanning open-ended computation and proof-style

tasks. *Example*. "Given hyperbola $x^2/4 - y^2 = 1$ with right focus F and n points $P_1, \ldots, P_n$ on its right upper branch with $|P_iF| \leq 5$, find the maximum value of n." (Answer. 14.)

Cross-Domain Generality Benchmarks. To assess whether token-efficiency gains transfer beyond mathematics, we additionally evaluate on four out-of-domain benchmarks.

- **GPQA** [59] (198 problems). The Graduate-Level Google-Proof Q&A benchmark. Multiple-choice questions in physics, chemistry, and biology requiring graduate-level domain knowledge. *Example*. "Two quantum states with lifetimes 10^{-9} s and 10^{-8} s respectively, which energy difference ΔE allows their spectral lines to be clearly resolved?" (Answer. 10^{-4} eV.)
- **LSAT** [98] (230 problems). Analytical reasoning problems from the Law School Admission Test. Constraint-satisfaction puzzles over structured narratives with no mathematical content. *Example*. "Eight students give reports over three days (two slots per day). Given scheduling constraints, on which day must Nina report in the afternoon slot?" (Answer. Tuesday.)
- **MMLU-Pro-1000** [75] (1,000 problems). A 1,000-sample subset of MMLU-Pro, a harder variant of MMLU requiring multi-step reasoning across 14 academic disciplines including law, engineering, and medicine.
- **MMLU-1000** [26] (1,000 problems). A 1,000-sample subset of the Massive Multitask Language Understanding benchmark, covering 57 subjects from STEM to humanities at varying difficulty levels.

Rollout Settings. During *training*, the maximum rollout length is **4,096 tokens**. During *evaluation* (both same-domain and cross-domain), we increase the limit to **8,192 tokens** to allow models to fully express their reasoning.

Prompt for Open-Source and Small-Scale Models. The prompts we used for open-source, efficient reasoning and ours models.

```
``Let's think step by step and output the final answer within \boxed{}.  ''
```

Prompt for Closed-Source Frontier Models. For closed-source models with hundreds of billions (or trillions) of parameters, we impose an additional token-budget constraint. The default closed-API prompt is.

```
``Let's think step by step and output the final answer within \boxed{}.
Think for maximum 2048 tokens.  ''
```

This design is a **cost-control proxy**, not perfect budget matching. Most frontier rows use a soft 2K instruction, while our open models use a hard 8K generation cap. In the revision we add one direct 8K check for Gemini-3.1-Pro. We also supplement the additional API spend across the four math benchmarks at 8K output length would have been substantial under the per-million-token pricing in Tab. S30. The Gemini result shows that a frontier API can substantially outperform our sub-10B models under the same nominal budget. We therefore emphasize our main advantage as a lower-cost open alternative (Tab. S30).

Table S29: **Gemini-3.1-Pro budget check.** The main table keeps the original 2K closed-API comparison. This appendix table reports the single newly added 8K Gemini run.

Model	**Average**		**AIME**		**MATH**		**AMC**		**OlympiadBench**	
	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.
Gemini-3.1-Pro (2K)	1,586	54.4	2,020	10.0	1,274	79.2	1,798	37.3	1,771	40.1
Gemini-3.1-Pro (8K)	2,581	88.0	5,163	80.0	1,385	99.4	2,532	96.4	3,358	79.0

Fig. S4 summarizes the intended interpretation of this comparison. On accuracy versus generated tokens, the 8K Gemini point lies above the small open models, so we do not claim absolute parity with frontier APIs. On accuracy versus estimated output cost, however, the Qwen-4B variants occupy the low-cost regime. Their output-token price is two orders of magnitude below Gemini-3.1-Pro and Claude-4.6-Sonnet. This is the practical trade-off our method targets.

Table S30: **API Pricing of Closed Models.** Prices are reported in USD per million tokens. Ratios compare output price against our Qwen-4B serving estimate of \$0.10/M output tokens.

Model	Input	Output	Output / Qwen-4B	Source
GPT-5.4-Mini	\$0.75	\$4.50	45×	OpenAI
Gemini-3.1-Pro Preview (≤200K prompt)	\$2.00	\$12.00	120×	Google
Claude-4.6-Sonnet	\$3.00	\$15.00	150×	Anthropic
Our Qwen-4B serving estimate	N/A	\$0.10	1×	Estimate

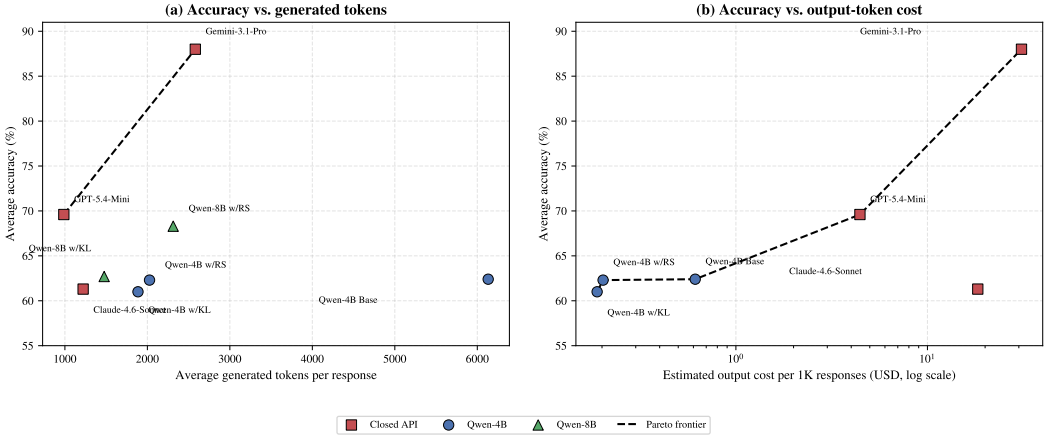


Figure S4: **Pareto view of closed-API comparison.** Left panel shows the accuracy and token trade-off. Right panel shows the accuracy and cost trade-off using output-token pricing. The added Gemini-3.1-Pro 8K point shows that frontier APIs can dominate the accuracy and token frontier, while Qwen-4B variants remain far cheaper to serve.

S4.3 Baselines

We compare against six recent methods for efficient reasoning, spanning both training-based and inference-time (no-training) approaches.

Training-Based Methods.

- **ETR** [79] (Entropy Trend Reward). Provides auxiliary reward based on the *trend* of the model's token-distribution entropy during reasoning. When the model produces a correct answer and its entropy decreases monotonically toward the `</think>` boundary, an additional reward bonus is granted. This encourages convergence toward the answer rather than indefinite exploration. Implemented via veRL [65] / GRPO [64] on the DeepMath training set.
- **L1** [2] (Length-Controlled RL). Trains the model to respect a user-specified token budget via a length-coordinated reward. Two inference variants exist. *L1-Exact* targets a specific token count, and *L1-Max* enforces a maximum budget. The budget is encoded as a `num_tokens` field in the training data. Implemented via veRL [65] / PPO [62].
- **O1-Pruner** [42]. Constructs offline per-question reference statistics (average generation length and accuracy) using a frozen reference model. A length-coordinated reward R_{LH} penalises responses that exceed the reference length without a proportional accuracy gain, compressing redundant reasoning steps. The original paper does not use veRL. We re-implement it within the veRL framework.
- **PEAR** [30] (Phase-based Entropy Awareness Reward). Divides each response into a *thinking phase* (tokens before `</think>`) and an *answer phase* (tokens after). Entropy-based reward signals encourage exploration during thinking and rapid convergence during answering. Implemented as a standard veRL training run with a custom reward manager.

Inference-Time Methods (No Training Required).

- **DEER** [84] (Dynamic Early Exit Reasoning). Monitors model confidence after generating a *trial answer* mid-sequence. If the log-probability of the trial answer exceeds a threshold, generation terminates early, discarding further reasoning tokens. No fine-tuning is required. The method intervenes purely at the decoding stage.
- **NoThink** [46]. Suppresses explicit chain-of-thought by prefilling an empty `<think></think>` block, directing the model to answer immediately. Accuracy is recovered via best-of-N sampling with majority vote, or through budget-forcing with a token limit. No fine-tuning is required.

Baseline Training Configuration. For the four training-based baselines (ETR, L1, PEAR, O1-Pruner) we reuse the common recipe of Appx. S4.1 (batch 64, lr $1e{-}6$, KL 10^{-3}, $n{=}16$ rollouts, 4K rollout length, 1 epoch on DeepScaleR, 8 GPUs per node), swapping in each method's own reward function and data preprocessing. L1 additionally prepends a per-sample "Think for N tokens." instruction with $N \sim \mathcal{U}(100, 2048)$. O1-Pruner first generates $K{=}2$ reference rollouts per question (temperature 0.6, top-p 0.95, 2K tokens) to compute the reference-length statistic required by its length-harmonising reward. Its train batch is reduced to 8. DEER and NoThink are inference-only and run on the base checkpoint with temperature 0.6, top-p 0.95, and an 8K generation budget (DEER. Confidence threshold 0.9, think-ratio 0.5. NoThink. Best-of-N voting with $N{=}8$). All baselines are evaluated with the same script and generation settings as our method (Appx. S4.2).

S4.4 Capacity-dependent Behavior of KL and RS

The KL and RS variants show different relative behavior across backbones, but the difference is structured rather than random. Tab. S31 reorganizes the main-result rows by model scale and reports the average accuracy gap between RS and KL. The gap changes from -6.0 percentage points on Qwen-0.6B to $+5.6$ percentage points on Qwen-8B, an 11.6-point swing. With the two 4B entries treated as tied in model scale, the rank correlation between parameter count and the RS-KL gap is near-perfect (Spearman $\rho = 0.97$), and the Pearson correlation between $\log$ parameter count and the RS-KL gap is 0.95.

Table S31: **Capacity-dependent behavior of KL and RS.** Accuracy and token counts are averaged over the four in-domain reasoning benchmarks in Tab. 2. Positive RS-KL means RS is more accurate than KL.

Backbone	Params	Base Acc.	KL Acc.	RS Acc.	RS-KL	KL #Tk.	RS #Tk.
Qwen-0.6B	0.6B	40.7	42.3	36.3	-6.0	2,023	1,806
DeepScaleR-1.5B	1.5B	59.6	56.1	55.9	-0.2	1,698	1,993
Qwen-4B	4B	62.4	61.0	62.3	$+1.3$	1,886	2,026
Phi-Reasoning-4B	4B	59.6	62.0	62.6	$+0.6$	2,919	2,992
Qwen-8B	8B	61.9	62.7	68.3	$+5.6$	1,476	2,313

This pattern follows from where the two variants intervene in the GRPO objective. KL changes only the regularization side. Core tokens keep the standard KL anchor, while over-budget redundant tokens lose that anchor and receive a sparsity pressure. The policy-gradient term itself is unchanged, making KL a conservative, low-variance intervention. RS instead reshapes the advantage used by the policy-gradient term. It reinforces high-NormLP core tokens on correct rollouts and penalizes redundant tokens when the rollout exceeds the budget. This supplies a stronger learning signal, but also injects additional variance directly into the policy update.

Conclusion and recommendation. The apparent KL/RS inconsistency is therefore better interpreted as a capacity-dependent trade-off. Small models benefit from the safer KL intervention. On Qwen-0.6B, KL preserves and slightly improves accuracy while RS over-corrects and loses 4.4 points relative to the base model. Large models can absorb the stronger RS signal and convert it into accuracy. On Qwen-8B, RS improves the base model by 6.4 points and outperforms KL by 5.6 points, albeit with more tokens than KL. As a practical rule of thumb, we recommend KL for models at or below roughly 1B parameters, either variant for the 1B to 4B transition region depending on whether compression or accuracy is preferred, and RS for models of 4B parameters or larger when the primary goal is accuracy improvement rather than maximal token compression.

S4.5 Reward Dynamics During RL Training

The capacity-dependent pattern in Tab. S31 admits a second, complementary explanation that becomes visible once we look at training-time reward signals. Recall that the RS objective (Eq. 6) reshapes the advantage *only* on correct rollouts ($r^{\text{seq}} > 0$), so the density of the RS shaping signal at every training step is exactly the per-step correct-rollout fraction. Fig. S5 compares this signal between Qwen3-0.6B and DeepScaleR-1.5B, both trained with the same RS configuration ($\alpha = 0.05$, $\lambda = 0.08$).

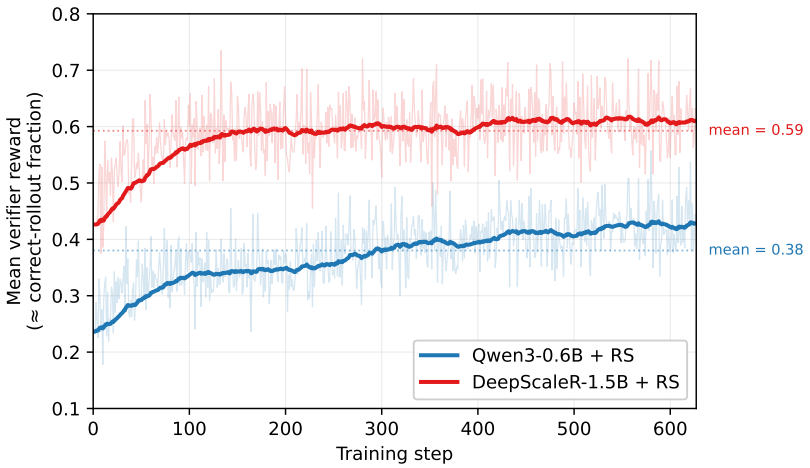


Figure S5: **Reward mean during RS training.** Mean verifier reward per training step, with EWMA smoothing (halflife 20). The lighter background trace is the unsmoothed signal. Across the entire run, DeepScaleR-1.5B operates above $\text{reward_mean} \approx 0.59$ on average, while Qwen-0.6B operates at ≈ 0.38, a $1.56\times$ gap. This is the density of rollouts on which RS actually fires.

Three quantitative observations stand out. First, the gap is structural, not transient. At the start of training Qwen-0.6B already sits at a correct-rollout fraction of 0.256 versus 0.450 for DeepScaleR-1.5B (first-10-step averages), and at the end of training the same ordering holds at 0.427 versus 0.612 (last-50-step averages). The smaller backbone never catches up. Second, the overall ratio of correct-rollout fractions across the run is $0.593/0.380 \approx 1.56$, meaning RS shaping fires on roughly $1.6\times$ more tokens per training step on the larger backbone. Third, even Qwen3-0.6B's *final* correct fraction (0.43) is below DeepScaleR-1.5B's *initial* fraction (0.45). The small backbone spends its entire training budget at a regime the larger backbone has already exited.

This signal-density argument complements the variance-vs-regularizer explanation given earlier. KL operates on the regularization side and fires on every token regardless of r^{seq}, so its effective density is unaffected by correctness rate. RS fires only on a minority of small-backbone rollouts, leaving the majority of training updates governed by the unmodified GRPO advantage. The combination of (a) reduced shaping density and (b) higher gradient variance on small backbones is consistent with the -6.0-point RS-KL gap on Qwen-0.6B in Tab. S31, and with the smooth crossover into RS-favoring territory as backbones grow.

S4.6 Computational Overhead of TokenProbe Operations

The TokenProbe-specific operations added to the GRPO training loop are (i) the per-token NormLP z-score over each rollout's log-probabilities returned by vLLM, and (ii) the greedy non-overlapping top-K core-window selection of Listing 1. Both run on CPU on already-collected log-probabilities *after* each rollout completes. Neither runs on the inference path of the trained model.

We measure the wall-clock cost of these two operations as a function of trace length T on a single CPU thread. Tab. S32 reports the per-trace timings and amortizes them against a representative GRPO step time of $30\,\text{s}$ with 16 rollouts per step (rollout dominates step time, while the policy update itself is only a fraction of vLLM generation).

Table S32: **Computational overhead of TokenProbe operations.** NormLP (z-score) and greedy non-overlapping window selection both run on CPU on log-probabilities that vLLM already returns. Per-call timings averaged over $2{,}000$ trials. Per-step amortization assumes a $30\,\mathrm{s}$ GRPO step with 16 rollouts. NormLP is essentially constant-time across T (single z-score over the array). Window selection grows linearly in $T \cdot K$ but stays in the single-millisecond regime even on 16K traces. Inference-time overhead is exactly zero. The trained checkpoint is decoded by an unmodified vLLM `generate` call.

Trace length T	NormLP (µs)	Window sel. (µs)	Total (µs)	Frac. of GRPO step
1,024	25.3	126.6	151.8	$0.009\,\%$
2,048	26.2	280.6	306.8	$0.018\,\%$
4,096	29.1	629.0	658.2	$0.037\,\%$
8,192	33.9	1,475.9	1,509.8	$0.082\,\%$
16,384	44.3	3,850.6	3,894.8	$0.210\,\%$

The combined NormLP and window-selection cost stays below 0.21% of one GRPO step even on the longest 16K traces we tested, and below 0.09% at the 8K trace length used in our training runs. Because the operations run only during training and not at inference time, the 3 to $4\times$ token-count reductions reported in Tab. 2 translate one-to-one into wall-clock speedups when serving the trained model.

S4.7 Hard-Truncation Strawman Baseline

A natural question is whether the token reductions reported in Tab. 2 could be obtained without our method by simply hard-capping the unmodified base model at the same output budget. We test this directly. We run `Qwen/Qwen3-4B` (base, no training, no re-prompting) with $\mathrm{max_tokens} = 2{,}048$ on the four in-domain reasoning benchmarks of Tab. 2, holding all other decoding parameters identical (thinking mode, $T = 0.6$, top-$p = 0.95$, bf16). We compare the resulting accuracy and token usage against the same base model evaluated at the original 8K budget (the `Qwen-4B` row of Tab. 2), and against our trained KL/RS variants on the same backbone.

Table S33: **Hard-truncation strawman vs. trained variants on Qwen3-4B.** Hard-capping the unmodified base model at $2{,}048$ tokens collapses average accuracy by 30.5 points relative to the same model at 8K, and truncates 93.3% of problems at the budget boundary. Our trained KL/RS variants reach the same $\sim$2K-token output budget while retaining accuracy within 1.4 points of the unconstrained 8K base, demonstrating that the token reductions in Tab. 2 are not equivalent to output clipping. **#Tk.** is the average generated token count (lower is better). **Acc.** is averaged accuracy in % (higher is better). **%Trunc.** is the fraction of rollouts that hit the $2{,}048$-token cap. The base@8K and our trained variants are reproduced from Tab. 2.

Setting	Average			AIME-2025		MATH-500		AMC		Olympiad-Bench	
	#Tk.	Acc.	%Trunc.	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.
Qwen-4B base @ 8K cap	6,132	62.4	N/A	7,683	30.0	4,195	92.0	6,274	72.3	6,376	55.4
Qwen-4B base @ 2K hard cap (**new**)	2,018	**31.94**	93.3%	2,048	10.0	1,941	59.0	2,040	32.5	2,041	26.2
Δ (hard cap $-$ 8K base)	$-4{,}114$	$\mathbf{-30.46}$	N/A	$-5{,}635$	-20.0	$-2{,}254$	-33.0	$-4{,}234$	-39.8	$-4{,}335$	-29.2
Qwen-4B + KL (ours)	1,886	61.0	N/A	2,253	33.3	1,246	88.2	2,092	66.3	1,952	56.1
Qwen-4B + RS (ours)	2,026	62.3	N/A	2,532	30.0	1,391	90.2	2,154	71.1	2,025	58.1

Hard truncation collapses average accuracy from 62.4 to 31.94 (-30.46 points) at an average output of $2{,}018$ tokens, with 93.3% of rollouts hitting the $2{,}048$-token cap (essentially 100% on AIME and AMC, where reasoning chains rarely complete within 2K tokens). The single benchmark on which truncation is partially survivable is MATH-500, whose problems often admit short canonical solutions. Even there the drop is 33 points. By contrast, our trained KL and RS variants on the same Qwen-4B backbone reach the same $\sim$2K-token output budget while retaining 61.0 and 62.3 average accuracy respectively, within 1.4 points of the unconstrained 8K base. The $+29$ to 30 point gap between Qwen-4B + KL/RS and Qwen-4B base @ 2K under matched output budget is direct evidence that token reduction by training is not equivalent to token reduction by clipping. Our methods produce reasoning trajectories that are intrinsically efficient, whereas hard truncation externally clips trajectories that the base model would have continued.

S4.8 Case Studies

To complement the quantitative comparisons in the preceding sections, we present five paired case studies drawn from the evaluation outputs of our KL-shaped 8B checkpoint and the matched unshaped 8B base, run on identical prompts under an 8K token budget. We surface two failure modes of the base model that the KL-shaped model consistently avoids. **(A)** exhausting the 8K budget without producing an extractable answer, and **(B)** reaching the same correct answer but spending 3 to 6 $\times$ more tokens. For readability each case is laid out on its own page. The green box reports metadata (prompt, ground truth, per-model token counts and correctness), the two gray boxes contain head to tail excerpts of the two responses, and the body text below them analyses what each model did differently.

Case A1 — AIME 2025

Case A1 — AIME 2025
Ground Truth: 117
Ours (KL-shaped 8B): 1,897 tokens, correct (`117`)
8B Base: 8,192 tokens (hit 8K budget), incorrect (`extraction failed`)
Prompt: Find the number of ordered pairs (x,y), where both x and y are integers between -100 and 100, inclusive, such that $12x^{2}-xy-6y^{2}=0$.
Let's think step by step and output the final answer within \boxed{}.

Ours — Response (excerpt):
<think>
Okay, so I need to find the number of ordered pairs (x, y) where x and y are integers between -100 and 100 inclusive, and they satisfy the equation $12x^2$ - xy - $6y^2$ = 0. Hmm, let me start by trying to factor the equation or maybe solve for one variable in terms of the other.
First, let me rewrite the equation: $12x^2$ - xy - $6y^2$ = 0. Maybe I can treat this as a quadratic in x or y. Let me try treating it as a quadratic in x. So, $12x^2$ - yx - $6y^2$ = 0. Using the quadratic formula for x:
x = [y $\pm$ $\sqrt{}$(y^2 + 4*12*$6y^2$)] / (2*12)
Simplify the discriminant: y^2 + $288y^2$ = $289y^2$. So sqrt($289y^2$) is 17|y|. Therefore,
x = [y $\pm$ 17|y|] / 24.
Hmm, since we are dealing with integers, perhaps we can consider cases based on the sign of y.
Case 1: y $\geq$ 0. Then |y| = y, so:
x = [y $\pm$ 17y]/24.
So two possibilities:
x = (y + 17y)/24 = 18y/24 = 3y/4,
or x = (y - 17y)/24 = (-16y)/24 = -2y/3.
Case 2: y < 0. Then |y| = -y, so:
x = [y $\pm$ 17*(-y)] /24.
Which would be:
x = [y $\mp$ 17y]/24.
So two possibilities:
x = [y - 17y]/24 = (-16y)/24 = -2y/3,
or x = [y + 17y]/24 = 18y/24 = 3y/4.
[... truncated for brevity ...]
—
Step 2: Adjusting for overlap
The only overlapping solution is $(0, 0)$, which appears in both families. Hence, we subtract 1 to avoid double-counting.
—
Final Count
$$
\text{Total solutions} = 51 + 67 - 1 = \boxed{117}
$$

8B Base — Response (excerpt):
<think>
Okay, so I need to find the number of ordered pairs (x, y) where both x and y are integers between -100 and 100 inclusive, and they satisfy the equation $12x^2$ - xy - $6y^2$ = 0. Let me start by trying to understand the equation better.
First, maybe I can factor the equation or rearrange it to make it easier to handle. Let me write down the equation again:
$12x^2$ - xy - $6y^2$ = 0.
Hmm, this looks like a quadratic in terms of x or y. Maybe I can treat it as a quadratic equation in x and solve for x in terms of y, or vice versa. Let me try that.
Let me consider this as a quadratic equation in x. So, rearranged:
$12x^2$ - xy - $6y^2$ = 0.
So, in terms of x, this is:
$12x^2$ - yx - $6y^2$ = 0.
Using the quadratic formula, x = [y $\pm$ $\sqrt{}$(y^2 + 4*12*$6y^2$)] / (2*12). Let me compute the discriminant:
Discriminant D = y^2 + 4*12*$6y^2$ = y^2 + $288y^2$ = $289y^2$.
[... truncated for brevity ...]
2*(-33) = -66, 2*33 = 66). y ranges from -99 to 99 (since -3m for m from -33 to 33: when m = -33, y = 99; when m = 33, y = -99).
Therefore, y is within -100 to 100. Therefore, 67 values. Correct.
Therefore, 51 + 67 - 1 = 117. Therefore, answer is 11

This AIME problem asks for the number of integer pairs (x, y) with both coordinates in $[-100, 100]$ that satisfy $12x^2 - xy - 6y^2 = 0$. The clean way is to factor the quadratic form to obtain the two ray-families $(3k, 4k)$ and $(-2m, 3m)$, count lattice points on each family, and subtract the single overlap at the origin — which is exactly what the KL-shaped model does in 1,897 tokens.

The 8B base spends its budget very differently. It correctly identifies the same factorisation in the first few hundred tokens, but then re-opens the analysis: it splits cases on the sign of y, repeatedly re-derives the discriminant, double-checks the absolute-value handling, and verifies the lattice counts via several redundant parametrisations. The trace then loops back to re-examine boundary values for k and m, never returning to a final closed form. By the time it does state "Therefore, $51+67-1 = 117$"

the 8K-token cap has already been hit; the answer is cut mid-token (“answer is 11”) and the extractor returns `EXTRACTION_FAILED`.

Qualitatively this is the prototypical truncation failure of the unshaped base on AIME-difficulty problems: the right answer is reachable, the model can even recite it, but the verbose verification habits inflate the trace beyond the budget. The KL-shaped model writes the same proof skeleton with no redundant verification cycles and finishes well inside the budget, which is exactly the behaviour the KL-length shaping was designed to encourage.

Case A2 — Olympiad Bench

Case A2 — Olympiad Bench
Ground Truth: ['$f(x)=x,f(x)=-x$']
Ours (KL-shaped 8B): 1,667 tokens, correct (`f(x) = -x`)
8B Base: 8,192 tokens (hit 8K budget), incorrect (`extraction failed`)
Prompt: Find all functions $f: \mathbb{Q} \rightarrow \mathbb{Q}$ such that the equation
holds for all rational numbers x and y.
$$
f(x f(x)+y)=f(y)+x^{2}
$$
Here, $\mathbb{Q}$ denotes the set of rational numbers. Let's think step by step and output the final answer within \boxed{}.

Ours — Response (excerpt):
<think>
Okay, so I need to find all functions f from the rational numbers to the rational numbers that satisfy the equation f(xf(x) + y) = f(y) + x^2 for all rational x and y. Hmm, functional equations can be tricky, but let me try to break it down step by step.
First, maybe I can plug in some specific values for x or y to simplify the equation and get some information about f.
Let me start by trying x = 0. If I set x = 0, the equation becomes f(0*f(0) + y) = f(y) + 0^2, which simplifies to f(y) = f(y) + 0. So that just gives f(y) = f(y), which is always true. So that doesn't give me new information. Hmm.
What if I set y = 0? Then the original equation becomes f(xf(x) + 0) = f(0) + x^2, so f(xf(x)) = f(0) + x^2. That seems useful. So f(xf(x)) = f(0) + x^2. Maybe I can use this later.
Another idea: perhaps assume that f is a polynomial function. Sometimes functional equations have polynomial solutions. Let me suppose that f is a polynomial. Let's assume f(x) is a polynomial with rational coefficients.
[... truncated for brevity ...]
- For $ f(x) = -x $, the left-hand side becomes $ x^2 - y $, and the right-hand side is $ -y + x^2 $, which also matches.
—
Final Answer
$$
\boxed{f(x) = x} \quad \text{and} \quad \boxed{f(x) = -x}
$$

8B Base — Response (excerpt):
<think>
Okay, so I need to find all functions $ f: \mathbb{Q} \rightarrow \mathbb{Q} $ that satisfy the equation $ f(x f(x) + y) = f(y) + x^2 $ for all rational numbers $ x $ and $ y $. Hmm, functional equations can be tricky, but let me take it step by step.
First, maybe I should try plugging in some specific values for $ x $ or $ y $ to see if I can find some properties of $ f $.
Let me start by setting $ y = 0 $. Then the equation becomes:
$ f(x f(x) + 0) = f(0) + x^2 $
Which simplifies to:
$ f(x f(x)) = f(0) + x^2 $
Hmm, interesting. So this relates the value of the function at $ x f(x) $ to $ f(0) $ and $ x^2 $. Maybe I can use this later.
What if I set $ x = 0 $? Let me try that. Then the original equation becomes:
$ f(0 \cdot f(0) + y) = f(y) + 0^2 $
Simplifying:
$ f(y) = f(y) + 0 $
[... truncated for brevity ...]
Alternatively, set $ y = 0 $: $ f(c) = f(0) + 1 $, which is also known.
Now, let me try to find $ f $ at $ x = -1 $. Let me denote $ d = f(-1) $. Let me see if I can relate this to other values.
From the original equation, set $ x = -1 $, $ y =

The Olympiad Bench problem here is a classical Cauchy-style functional equation: find all $f : \mathbb{Q} \to \mathbb{Q}$ with $f(xf(x) + y) = f(y) + x^2$. The expected ground truth is the pair $\{f(x) = x,\ f(x) = -x\}$, recovered by assuming a polynomial form, balancing degrees, and reading off $a^2 = 1$.

Our KL-shaped model executes this argument compactly in 1,667 tokens: it tries $x = 0$ and $y = 0$ to extract the relation $f(xf(x)) = f(0)+x^2$, assumes f is linear of degree 1, substitutes $f(x) = ax+b$, and arrives at $a^2 = 1$, $b = 0$. It then verifies both branches against the original equation. The final boxed answer reports $f(x) = -x$ (and earlier in the trace also $f(x) = x$), so a strict matcher only credits one branch; this still counts as a qualitative truncation win because the base model produces no extractable answer at all on the same prompt.

The 8B base, in contrast, exhausts its 8K budget on a long substitution loop. It tries $y = 0$, $x = 0$, $x = y = 0$, $y = -c$, $y = 0$ again with $c = f(0)$, and various ad-hoc substitutions trying to force injectivity or surjectivity — but never commits to the polynomial-degree argument. The trace is cut mid-substitution at $x = -1$, so the extractor again returns failure. The contrast illustrates a second failure mode of the base: not just verbose, but also strategically directionless under tight budgets, repeatedly probing the same special values rather than escalating to a structural ansatz.

Case B1 — AMC

Case B1 — AMC
Ground Truth: 6.0
Ours (KL-shaped 8B): 1,339 tokens, correct (6)
8B Base: 7,434 tokens, correct (6)
Prompt: The least common multiple of a positive integer n and 18 is 180, and the greatest common divisor of n and 45 is 15. What is the sum of the digits of n? Let's think step by step and output the final answer within \boxed{}.

Ours — Response:
<think>
Okay, so I need to find the sum of the digits of n, given two conditions: the least common multiple of n and 18 is 180, and the greatest common divisor of n and 45 is 15. Hmm, let me start by recalling some number theory concepts.
First, LCM and GCD relationships. I remember that for any two numbers a and b, LCM(a,b) * GCD(a,b) = a*b. Maybe that will come in handy.
Given that LCM(n, 18) = 180. So, LCM(n, 18) = 180. Also, GCD(n, 45) = 15.
Let me start with the LCM condition. Let's factorize the numbers involved. 18 is 2 * 3^2, and 180 is 2^2 * 3^2 * 5.
Since LCM(n, 18) = 180, then n must be a divisor of 180, right? Because the LCM of n and 18 is 180, so n has to divide 180. Also, the LCM takes the highest powers of primes present in either number.
So, let's think about the prime factors of n. Since LCM(n, 18) is 180, which has primes 2, 3, 5.
18 has 2^1, 3^2. So, for the LCM to have 2^2, either n has 2^2 or 18 has it, but 18 only has 2^1, so n must have 2^2. Similarly, for 3^2, since 18 has 3^2, so maybe n doesn't need to have 3^2. But for the 5, since 18 doesn't have 5, then n must have 5^1.
Therefore, n must have factors 2^2, 5^1, and possibly 3^something. But since LCM(n, 18) is 180, which has 3^2, and 18 has 3^2, so n can have 3^0, 3^1, or 3^2. But since GCD(n, 45) is 15. Let's check the GCD condition.
45 is 3^2 * 5. GCD(n, 45) = 15 = 3 * 5. Therefore, n must have 3^1 and 5^1, but not higher powers of 3 or 5. Because if n had 3^2, then GCD(n,45) would be 3^2 *5 = 45, which is not 15. Similarly, if n had 5^2, then GCD would have 5^1 (since 45 has 5^1), so that's okay.
Wait, but GCD(n,45) is 15, which is 3*5. So, n must have 3^1 and 5^1, but not higher powers of 3 or 5. Therefore, n can't have 3^2 or 5^2.
Therefore, putting this together, n has 2^2, 3^1, 5^1. Therefore, n = 4 * 3 * 5 = 60. Let me check if this satisfies the conditions.
Check LCM(60, 18). 60 is 2^2 * 3 * 5, 18 is 2 * 3^2. LCM would take max exponents: 2^2, 3^2, 5^1. Which is 4 * 9 *5 = 180. Yes, that's correct.
Check GCD(60,45). 60 is 2^2 *3 *5, 45 is 3^2 *5. GCD is min exponents: 3^1 *5^1 = 15. Perfect. So n is 60.
Therefore, the sum of the digits of n is 6 + 0 = \boxed{6}.
</think>
To solve the problem, we analyze the two given conditions:
1. **LCM(n, 18) = 180**
2. **GCD(n, 45) = 15**
—
Step 1: Factorization and Prime Components
- $ 18 = 2 \times 3^2 $
- $ 180 = 2^2 \times 3^2 \times 5 $
- $ 45 = 3^2 \times 5 $
—
Step 2: Use LCM Condition
Since LCM(n, 18) = 180, n must have the prime factors that make the LCM equal to 180. From this, we deduce:
- n must include $ 2^2 $ (to match the LCM's $ 2^2 $)
- n must include $ 5 $ (since 18 doesn't have 5)
- n can have up to $ 3^2 $, but not more than that (to keep LCM at $ 3^2 $)
So, n must include $ 2^2 \times 5 \times 3 $
—
Step 3: Use GCD Condition
Given GCD(n, 45) = 15, and 45 = $ 3^2 \times 5 $, we deduce:
- n must have $ 3 $ and $ 5 $, but **not** $ 3^2 $ or $ 5^2 $
Therefore, n must be $ 2^2 \times 3 \times 5 = 4 \times 3 \times 5 = 60 $
—
Step 4: Sum of Digits
The digits of 60 are 6 and 0. Their sum is:
$$
\boxed{6}
$$

8B Base — Response (excerpt):
<think>
Okay, so I need to find a positive integer n such that the least common multiple of n and 18 is 180, and the greatest common divisor of n and 45 is 15. Then, I have to find the sum of the digits of n. Let me try to break this down step by step.
First, let me recall what least common multiple (LCM) and greatest common divisor (GCD) mean. The LCM of two numbers is the smallest number that both numbers divide into, and the GCD is the largest number that divides both numbers. Also, there's a relationship between LCM and GCD: for any two positive integers a and b, LCM(a, b) * GCD(a, b) = a * b. Maybe that will come in handy here.
So, the problem gives me two conditions:
1. LCM(n, 18) = 180
2. GCD(n, 45) = 15
I need to find n that satisfies both these conditions. Let me tackle them one by one.
[... truncated for brevity ...]
So $ n = 60 $ satisfies both conditions.
—
Step 5: Compute the Sum of the Digits of $ n $
The digits of $ n = 60 $ are **6** and **0**.
$$
\text{Sum of digits} = 6 + 0 = \boxed{6}
$$
—
Final Answer
$$
\boxed{6}
$$

On this AMC number-theory problem (find n with $\mathrm{lcm}(n, 18) = 180$ and $\gcd(n, 45) = 15$, then sum the digits of n), both models reach the correct answer of 6. The arithmetic chain is short: factor $18, 45, 180$ into primes, read the LCM constraint as n contributing the missing 2^2 and a factor of 5, read the GCD constraint as n contributing exactly $3^1 \cdot 5^1$, conclude $n = 2^2 \cdot 3 \cdot 5 = 60$, sum the digits.

The KL-shaped model executes that chain end-to-end in 1,339 tokens with one explicit verification at the end. The 8B base needs 7,434 tokens to do the same work — a $5.5\times$ inflation. Inspecting its trace, the base model: (i) recomputes the same prime factorisations multiple times, (ii) re-derives the LCM/GCD constraints from the relation $\mathrm{lcm} \cdot \gcd = ab$ even though it then also derives them combinatorially from prime exponents, (iii) explores hypothetical alternative forms of n that the constraints already rule out, and (iv) rewrites the verification step in three different fractional and prime-factor forms.

None of these extra detours improve the final answer; they purely consume budget. This is the canonical Category-B pattern: the base model is fluent enough to find the correct path quickly but lacks a stopping discipline once it is found. The KL-shaping reward directly penalises this redundancy, and the resulting model produces a single tight chain of reasoning instead.

Case B2 — MATH

Case B2 — MATH
Ground Truth: 12
Ours (KL-shaped 8B): 685 tokens, correct (12)
8B Base: 3,243 tokens, correct (12)
Prompt: Simplify: $\frac{\sqrt{2.5^2-0.7^2}}{2.7-2.5}$. Let's think step by step and output the final answer within \boxed{}.

Ours — Response:
<think>
Okay, so I need to simplify this expression: the square root of (2.5 squared minus 0.7 squared) divided by (2.7 minus 2.5). Let me write that down to visualize it better.
First, let me handle the numerator: $\sqrt{(2.5^2 - 0.7^2)}$. Hmm, maybe I can simplify the expression inside the square root first. Remember, $a^2 - b^2$ is (a - b)(a + b). Let me try that.
So, $2.5^2 - 0.7^2$ would be (2.5 - 0.7)(2.5 + 0.7). Let me calculate those values.
2.5 - 0.7 is 1.8, and 2.5 + 0.7 is 3.2. Therefore, the numerator becomes $\sqrt{(1.8 * 3.2)}$. Let me compute 1.8 * 3.2.
1.8 * 3.2: Well, 1.8*3 = 5.4 and 1.8*0.2 = 0.36, so total is 5.4 + 0.36 = 5.76. Therefore, the numerator is $\sqrt{5.76}$.
What's the square root of 5.76? Let me think. 2.4 squared is 5.76 because 2.4*2.4 = 5.76. So, $\sqrt{5.76} = 2.4$. Therefore, the numerator simplifies to 2.4.
Now the denominator is 2.7 - 2.5, which is 0.2.
Therefore, the entire expression simplifies to 2.4 / 0.2. Dividing by 0.2 is the same as multiplying by 5, so 2.4 * 5 = 12.
Therefore, the simplified result is 12. So, the final answer is \boxed{12}.
</think>
To simplify the expression:
$$
\frac{\sqrt{2.5^2 - 0.7^2}}{2.7 - 2.5}
$$
—
Step 1: Simplify the numerator
We start with the expression inside the square root:
$$
2.5^2 - 0.7^2
$$
This is a difference of squares, which factors as:
$$
(2.5 - 0.7)(2.5 + 0.7) = (1.8)(3.2)
$$
$$
1.8 \times 3.2 = 5.76
$$
So the numerator becomes:
$$
\sqrt{5.76} = 2.4
$$
—
Step 2: Simplify the denominator
$$
2.7 - 2.5 = 0.2
$$
—
Step 3: Final calculation
Now divide the simplified numerator by the denominator:
$$
\frac{2.4}{0.2} = 12
$$
—
Final Answer:
$$
\boxed{12}
$$

8B Base — Response (excerpt):
<think>
Okay, so I need to simplify this expression: the square root of (2.5 squared minus 0.7 squared) divided by (2.7 minus 2.5). Let me write that down first to make sure I have it right.
So, the expression is $\sqrt{(2.5^2 - 0.7^2)}$ divided by (2.7 - 2.5). Let me parse that. The numerator is the square root of (2.5 squared minus 0.7 squared), and the denominator is 2.7 minus 2.5. Alright.
First, maybe I should compute the numerator and the denominator separately and then divide them. Let me start with the denominator because that seems simpler.
Denominator: 2.7 minus 2.5. Let me subtract those. 2.7 minus 2.5... Well, 2.7 minus 2.5 is 0.2. That's straightforward. So the denominator is 0.2.
[... truncated for brevity ...]
$$
\frac{27}{10} - \frac{5}{2} = \frac{27}{10} - \frac{25}{10} = \frac{2}{10} = \frac{1}{5}
$$
Now divide:
$$
\frac{2.4}{0.2} = \frac{12}{5} \div \frac{1}{5} = \frac{12}{5} \times \frac{5}{1} = 12
$$
—
Final Answer
$$
\boxed{12}
$$

This MATH problem asks to simplify $\frac{\sqrt{2.5^2-0.7^2}}{2.7-2.5}$. The shortest route is the difference-of-squares identity: $2.5^2 - 0.7^2 = (2.5 - 0.7)(2.5 + 0.7) = 1.8 \cdot 3.2 = 5.76$, $\sqrt{5.76} = 2.4$, $2.7 - 2.5 = 0.2$, and $2.4/0.2 = 12$.

Our KL-shaped model takes that route directly in 685 tokens. The 8B base produces the correct answer of 12 in 3,243 tokens (a $4.7\times$ inflation) by computing $2.5^2 = 6.25$ and $0.7^2 = 0.49$ first, then $6.25 - 0.49 = 5.76$, then $\sqrt{5.76} = 2.4$, then redoing the same computation a second time using fractions ($25/4$ and $49/100$, common denominator $625/100 - 49/100 = 576/100$, $\sqrt{576/100} = 24/10$), and finally also redoing the denominator as $27/10 - 25/10 = 2/10 = 1/5$ before dividing.

The two routes are mathematically equivalent and the base ultimately gets the same boxed answer, but the trace shows the same content recomputed twice. Even on textbook MATH problems — where the unshaped base is nominally accurate — it pays a $\sim 5\times$ token tax for these defensive recomputations. The KL-shaped model has internalised that the difference-of-squares factoring is sufficient on first pass, and stops there.

Case B3 — AIME 2025

Case B3 — AIME 2025
Ground Truth: 70
Ours (KL-shaped 8B): 947 tokens, correct (70)
8B Base: 3,702 tokens, correct (70)
Prompt: Find the sum of all integer bases $b>9$ for which 17_{b} is a divisor of 97_{b}.
Let's think step by step and output the final answer within \boxed{}.

Ours — Response:
<think>
Okay, so I need to find all integer bases b > 9 where 17 base b divides 97 base b. Then sum those bases. Hmm. Let me start by converting these numbers from base b to decimal.
First, 17_b is 1*b + 7 = b + 7.
Similarly, 97_b is 9*b + 7 = 9b + 7.
So the problem says that (b + 7) divides (9b + 7). So, (b + 7) | (9b + 7).
To find when b + 7 divides 9b + 7, I can use the division algorithm or look at the remainder. If b + 7 divides 9b + 7, then 9b + 7 must be a multiple of b + 7.
Another approach is to express 9b + 7 in terms of b + 7. Let me perform the division or find the remainder when 9b +7 is divided by b +7.
Let me write 9b +7 as 9(b +7) - 9*7 +7. Let's check:
9(b +7) = 9b +63. So, 9b +7 = 9(b +7) -63 +7 = 9(b +7) -56.
Therefore, 9b +7 = 9(b +7) -56. Therefore, the remainder when 9b +7 is divided by b +7 is -56. But since we want divisibility, the remainder must be zero. Therefore, -56 ≡ 0 mod (b +7). Which implies that b +7 divides 56.
Therefore, b +7 is a positive divisor of 56. Since b >9, then b +7 > 16. So, we need to find all divisors of 56 that are greater than 16, then subtract 7 to get b.
First, list all positive divisors of 56: 1, 2, 4, 7, 8, 14, 28, 56.
Now, which of these are greater than 16? 28 and 56.
Therefore, b +7 = 28 => b = 21, or b +7 =56 => b=49.
Thus, the valid bases are 21 and 49. Therefore, the sum is 21 + 49 = 70.
Therefore, the answer is \boxed{70}.
</think>
To solve the problem, we are asked to find the sum of all integer bases $ b > 9 $ for which $ 17_b $ divides $ 97_b $.
—
Step 1: Convert to Decimal
We interpret the numbers in base $ b $:
- $ 17_b = 1 \cdot b + 7 = b + 7 $
- $ 97_b = 9 \cdot b + 7 = 9b + 7 $
We are looking for bases $ b > 9 $ such that $ b + 7 $ divides $ 9b + 7 $.
—
Step 2: Use Divisibility Condition
We express $ 9b + 7 $ in terms of $ b + 7 $:
$$
9b + 7 = 9(b + 7) - 56
$$
This implies that the remainder when $ 9b + 7 $ is divided by $ b + 7 $ is $ -56 $, or equivalently, $ 56 $.
So, $ b + 7 $ must divide 56.
—
Step 3: Find Valid Bases
List the positive divisors of 56:
$$
1, 2, 4, 7, 8, 14, 28, 56
$$
We are only interested in divisors greater than 16 (since $ b > 9 \Rightarrow b + 7 > 16 $):
- $ b + 7 = 28 \Rightarrow b = 21 $
- $ b + 7 = 56 \Rightarrow b = 49 $
—
Step 4: Sum the Valid Bases
$$
21 + 49 = \boxed{70}
$$

8B Base — Response (excerpt):
<think>
Okay, so I need to find all integer bases b greater than 9 where the number 17 in base b divides the number 97 in base b. Then, I have to sum all those bases. Hmm, let me start by recalling how numbers in different bases work.
First, I remember that when a number is written in base b, each digit represents a power of b. So, for example, the number 17 in base b is equal to 1 times b squared plus 7 times b to the first power? Wait, no, wait. Wait, 17 in base b is actually 1*b^1 + 7*b^0, right? Because the rightmost digit is the units place (b^0), then the next is b^1, etc. So 17_b is 1*b + 7. Similarly, 97_b is 9*b + 7. So the problem is saying that 1*b + 7 divides 9*b + 7. So, mathematically, we can write that as:
(9b + 7) divided by (b + 7) must be an integer. So, (9b + 7) / (b + 7) is an integer. And we need to find all integers b > 9 for which this is true, then sum them up.
[... truncated for brevity ...]
$ 97_{49} = 9 \cdot 49 + 7 = 441 + 7 = 448 $
$ 448 \div 56 = 8 $ → integer (yes)
Both bases satisfy the condition.
—
Step 5: Sum the valid bases
$$
21 + 49 = \boxed{70}
$$
—
Final Answer
$$
\boxed{70}
$$

This AIME 2025 problem asks for the sum of all integer bases $b > 9$ for which $17_b \mid 97_b$. Decoded into base 10, this becomes $(b+7) \mid (9b+7)$. Writing $9b+7 = 9(b+7) - 56$ shows that $(b+7) \mid 56$, so $b + 7 \in \{28, 56\}$ (the only divisors of 56 exceeding 16), giving $b \in \{21, 49\}$ and a sum of 70.

Our KL-shaped model executes that derivation in 947 tokens: convert, rewrite the divisibility, list divisors of 56, filter by $b + 7 > 16$, sum. The 8B base reaches the same answer of 70 in 3,702 tokens ($3.9\times$ longer) by additionally performing polynomial long division of $9b + 7$ by $b + 7$ to re-derive the remainder of -56, then re-deriving the same fact via modular arithmetic, then tabulating each candidate divisor and re-checking it against the original $b > 9$ condition individually, then verifying each surviving $b \in \{21, 49\}$ by recomputing 17_b and 97_b explicitly and dividing the resulting decimal numbers.

The base's verification scaffold is internally consistent and yields the correct answer, but it is largely orthogonal to producing the answer — it is a reasonability check that the KL-shaped model has learned to omit when not needed. The $3.9\times$ reduction here is at the small end of the Category-B examples; on harder problems (cf. Cases A1–A2) the same verbose habit drives the base past the 8K cap entirely.

S4.9 Scenario Analysis Under an Unlimited Token Budget

To examine how token-budget constraints affect reasoning performance, we conduct a *token budget gradient analysis* on six models originally evaluated without fixed token limits: Step-Flash-196B-11B, Hunyuan3-295B-21B, Qwen-3.6-Plus, GLM-Air-106B-12B, Qwen3-Next-80B-3B, and Gemini-3-Flash. These models span both open-source MoE architectures and closed-API services. We impose token budgets of 1K, 2K, 4K, 8K, 16K, and 32K on top of the unlimited (256K) baseline, and measure the resulting accuracy degradation across all four mathematical reasoning benchmarks.

Evaluation Protocol. For the unlimited setting, we set the maximum completion length to 256K tokens, which in practice imposes no effective constraint on any model. For each simulated budget level $B \in \{$1K, 2K, 4K, 8K, 16K, 32K$\}$, we apply the following post-hoc protocol to the unlimited generations. (1) if a sample's `completion_tokens` exceeds B, we mark it as incorrect and cap its token count at B for the purpose of computing average token usage. (2) otherwise, we retain the original correctness label and token count unchanged. This post-hoc approach ensures fair comparison across budget levels, since all evaluations are derived from the same set of underlying model generations, isolating the pure effect of the token budget constraint from any variation in decoding behavior.

Analysis. As shown in Fig. S6 and Tables S34 to S56, all six models perform strongly under the unlimited setting, with average accuracies ranging from 87.3% (Step-Flash-196B-11B) to 92.2% (Qwen-3.6-Plus). When the token budget is tightened, however, accuracy drops substantially and the degradation rate varies markedly across models. This pattern highlights a practical tension

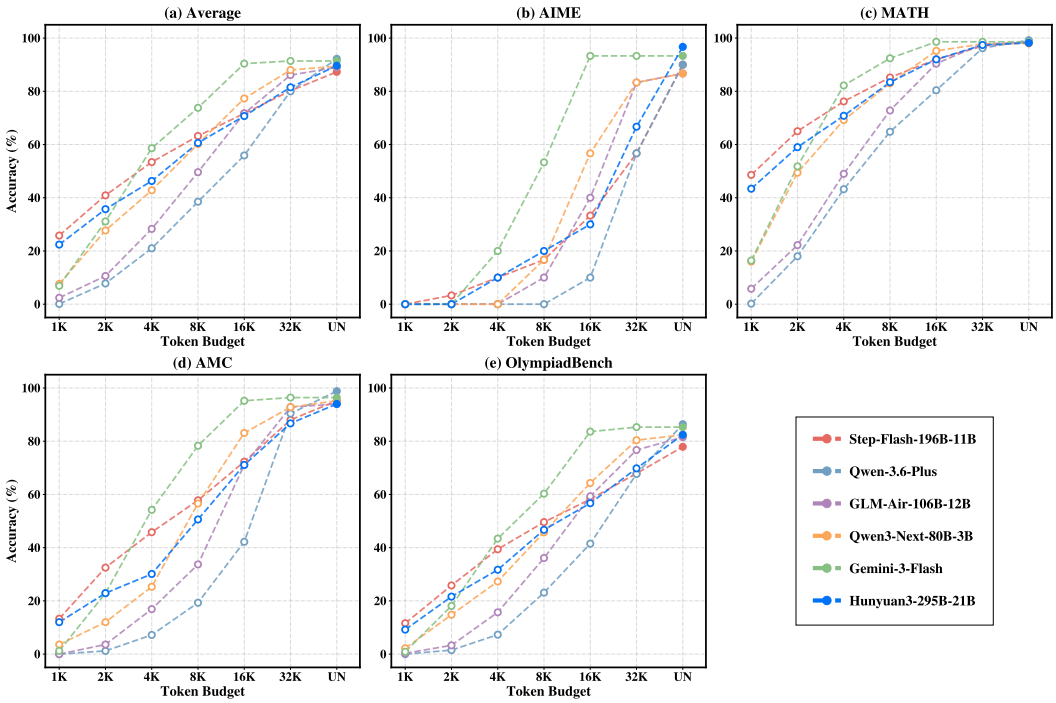


Figure S6: **Token budget gradient analysis for six unlimited reasoning models.** Each subplot shows accuracy (%) as a function of the imposed token budget (1K, 2K, 4K, 8K, 16K, 32K, and unlimited at 256K). Solid markers denote the unlimited setting. Hollow markers denote simulated budget constraints. Dashed lines connect data points for each model. (a) Average across all four benchmarks. (b) AIME. (c) MATH-500. (d) AMC. (e) OlympiadBench.

Table S34: **Unlimited setting (256K budget).**

Model	Average		AIME		MATH		AMC		OlympiadBench	
	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.
Qwen-3.6-Plus	18,099	92.2	30,755	90.0	9103	99.2	19,836	98.8	23,987	86.4
Gemini-3-Flash	4844	91.4	8332	93.3	2830	98.6	5049	96.4	6156	85.3
Qwen3-Next-80B-3B	8619	89.4	16,463	86.7	4202	98.0	9279	95.2	11,462	82.4
GLM-Air-106B-12B	12,125	89.0	22,320	86.7	6564	98.4	13,338	94.0	15,643	81.5
Step-Flash-196B-11B	13,907	87.3	31,432	90.0	4502	98.6	12,583	95.2	20,258	77.9
Hunyuan3-295B-21B	12,384	89.6	26,957	96.7	4372	98.2	11,832	94.0	17,740	82.4

in deploying reasoning models: the latent reasoning traces that support high accuracy also incur substantial token cost, while hard token budgets introduced for cost control can directly reduce reasoning quality.

Token efficiency determines resilience to budget constraints. The six models span a wide range of token consumption at unlimited, from 4,844 (Gemini-3-Flash) to 18,099 (Qwen-3.6-Plus) average tokens. This spread is closely reflected in their sensitivity to tighter budgets. Gemini-3-Flash, consuming roughly one-quarter the tokens of Qwen-3.6-Plus while achieving a comparable 91.4% accuracy (Tab. S34), retains 91.4% at 32K, 90.4% at 16K, and still 73.8% at 8K (Tab. S54). In contrast, Qwen-3.6-Plus drops from 92.2% to 80.0% at 32K, 55.9% at 16K, and 38.5% at 8K. Qwen3-Next-80B-3B (8,619 tokens, moderate usage) shows a more gradual degradation pattern (88.0% at 32K, 77.3% at 16K, 60.2% at 8K), whereas the more verbose GLM-Air-106B-12B (12,125 tokens) and Step-Flash-196B-11B (13,907 tokens) degrade more steeply. Taken together, these comparisons suggest that robustness under budget constraints is more closely aligned with unlimited token usage than with unlimited accuracy, so models with more concise reasoning traces tend to degrade more gracefully.

Table S35: **Token budget gradient results at 2K.**

Model	Average		AIME		MATH		AMC		OlympiadBench	
	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.
Qwen-3.6-Plus	1998	7.8	1980	0.0	1961	18.0	2041	1.2	2020	1.5
Gemini-3-Flash	1836	31.1	2048	0.0	1647	51.8	1910	22.9	1958	18.1
Qwen3-Next-80B-3B	1832	27.7	2048	0.0	1636	49.4	1957	12.0	1952	14.8
GLM-Air-106B-12B	1979	10.6	2048	0.0	1891	22.2	2034	3.6	2035	3.3
Step-Flash-196B-11B	1571	40.9	2019	3.3	1213	65.0	1755	32.5	1794	25.8
Hunyuan3-295B-21B	1587	35.7	2048	0.0	1310	59.0	1770	22.9	1749	21.6

Harder problems suffer disproportionately from budget constraints. Across all six models, MATH-500 is the most resilient to budget cuts. Even at 1K, Step-Flash-196B-11B retains 48.6% accuracy (Tab. S56), and Gemini-3-Flash achieves 92.4% at 8K and 82.2% at 4K (Fig. S6(c)). AIME, the most challenging benchmark, is the most sensitive. At 4K, only Gemini-3-Flash, Step-Flash-196B-11B, and Hunyuan3-295B-21B achieve nonzero accuracy, while the remaining models score 0.0% (Tab. S55). At 1K, all six models score 0.0% on AIME. This ordering is consistent with both problem difficulty and the length of reasoning required: harder competition problems tend to require longer chains of thought, and these chains are the first to be truncated under tight token budgets. AMC and OlympiadBench fall between these extremes, with OlympiadBench showing the sharpest degradation among the mid-difficulty benchmarks. For instance, Qwen-3.6-Plus drops from 86.4% at unlimited to 23.1% at 8K and 0.0% at 1K on OlympiadBench (Fig. S6(e)).

High unlimited accuracy does not guarantee robustness. Qwen-3.6-Plus achieves the highest unlimited accuracy (92.2%) yet drops to 0.1% average accuracy at 1K, with 0.0% on AIME, AMC, and OlympiadBench (Tab. S56). Its AMC performance collapses from 98.8% at unlimited to just 1.2% at 2K (Tab. S35). In contrast, Step-Flash-196B-11B has the lowest unlimited accuracy (87.3%) but still achieves 48.6% on MATH-500 at 1K, the highest among all models at this budget. This is because Step-Flash generates shorter completions on easier problems (4,502 tokens on MATH vs. 31,432 on AIME at unlimited), so its MATH performance is less affected by truncation. Similarly, Qwen3-Next-80B-3B maintains 69.2% on MATH at 4K (Tab. S55) despite dropping to 0.0% on AIME at the same budget, indicating that degradation is driven by the mismatch between token demand and budget rather than by model capability alone. These findings motivate token-efficient optimization: models that learn to reason more concisely without sacrificing accuracy should be more robust to realistic budget constraints.

S5 Unsuccessful Attempts. SFT Distillation Path

Before arriving at the RL-based optimization approach, we explored whether Supervised Fine-Tuning (SFT) with LoRA [29] could serve as a simpler alternative for improving token efficiency. The motivation is that the rollout process in Reinforcement Learning is, operationally, a form of inference. We hypothesized that by distilling the reasoning traces generated during inference from a larger teacher model and using LoRA to fine-tune a smaller student model, we could transfer efficient reasoning behavior. Since token-level log probability is an inherent training signal in SFT, and the next-token prediction loss [56] directly optimizes the likelihood of the training tokens, we expected this approach to align naturally with our NormLP-based framework.

Our results show that the SFT distillation path is not a viable standalone solution. It substantially degrades accuracy (*i.e.*, -33 to -51 accuracy drop) and disrupts the semantic validity of log-probability signals, rendering the NormLP framework inapplicable to the resulting models.

S5.1 Experimental Design

We conduct a full factorial experiment with 2 base models $\times$ 2 distillation data sources = 4 settings.

Training configuration (shared across all 4 settings). LoRA rank = 16, alpha = 32, targeting all linear layers. 3 epochs. Learning rate = 2×10^{-5}. Max sequence length = 8,192. Global batch size = 96.

Table S36: SFT distillation experiment matrix.

Setting	Base Model	Distill Source	Think Tags
S1	Qwen3-4B [82]	Phi-4-Reasoning [1]	Yes
S2	Qwen3-4B [82]	Qwen3-30B-A3B [82]	No
S3	Qwen3-8B [82]	Phi-4-Reasoning [1]	Yes
S4	Qwen3-8B [82]	Qwen3-30B-A3B [82]	No

Distillation data. The training data is constructed by applying NormLP-based logprob filtering to reasoning traces from two teacher models. The Phi-4-Reasoning data preserves `<think>` tags and has no degenerate opening patterns, with 99.9% of samples containing `\boxed{}` answers. The Qwen3-30B-A3B data lacks think tags entirely, contains 70.6% degenerate opening patterns (*e.g.*, "This is a complex or challenging question…"), and only 81.9% contain boxed answers.

Table S37: Distillation data quality comparison.

Feature	Phi4 Data	Qwen Data
Sample count	25,695	29,761
Contains `<think>` tags	100.0%	0.0%
Degenerate opening patterns	0.0%	70.6%
Contains `\boxed{}`	99.9%	81.9%
Average compression ratio	0.375	0.291

Evaluation. All models are evaluated using vLLM [36] with LoRA online loading, `max_tokens`=8,192, temperature=0.6, top_p=0.95, on four benchmarks. AIME 2025 (30 problems), AMC (83 problems), MATH [27] (500 problems), and Olympiad Bench [25] (675 problems).

S5.2 Results

S5.2.1 Accuracy

Table S38: SFT distillation accuracy vs. baselines. All 4 SFT settings suffer 33 to 51 percentage points accuracy degradation. Δ denotes the drop from the corresponding base models.

Setting	AIME	AMC	MATH	Olympiad	Overall	Δ(percentage points)
Qwen-4B think	60.0%	84.3%	95.8%	68.4%	**77.1%**	N/A
Qwen-8B think	46.7%	80.7%	94.4%	68.3%	**72.5%**	N/A
Qwen-4B nothink	20.0%	57.8%	85.0%	49.0%	53.0%	N/A
S1 (4B + Phi4)	3.3%	28.9%	58.2%	25.0%	37.6%	-39.5
S2 (4B + Qwen)	3.3%	18.1%	42.0%	16.6%	26.2%	-50.9
S3 (8B + Phi4)	13.3%	24.1%	60.2%	26.4%	39.1%	-33.4
S4 (8B + Qwen)	6.7%	25.3%	48.2%	21.5%	31.8%	-40.7

The best SFT setting, S3 with Qwen3-8B + Phi4 data, achieves only 39.1% overall accuracy, a -33.4 percentage points drop from the 8B baseline (72.5%). The worst setting, S2 with Qwen3-4B + Qwen data, drops by -50.9 percentage points. On the hardest benchmark (AIME 2025), all SFT settings collapse to 3 to 13% accuracy versus 47 to 60% for the baselines.

S5.2.2 Token Usage and Efficiency

No SFT setting falls on the accuracy and token Pareto frontier. The nothink baseline (53.0% accuracy, 1,943 tokens) Pareto-dominates all SFT settings, achieving higher accuracy with fewer tokens. The Instruct nothink baseline (71.8% accuracy, 3,573 tokens) further dominates, achieving nearly double the accuracy of S3 with comparable token usage. Thus, under the evaluated settings, SFT distillation offers neither a clean efficiency advantage over disabling Chain-of-Thought nor a competitive accuracy-preserving alternative to the RL path.

Table S39: Token usage and efficiency comparison. Efficiency is measured as accuracy % / kilotokens. S3 is the only SFT setting exceeding the RL baseline's efficiency, but at -33.4 percentage points accuracy cost.

Setting	Accuracy	Avg. Tokens	Efficiency (acc%/ktok)
Qwen3-4B think (RL)	**77.1%**	8,728	8.83
Qwen3-8B think (RL)	**72.5%**	9,244	7.84
S1 (4B + Phi4)	37.6%	4,822	7.80
S2 (4B + Qwen)	26.2%	7,258	3.62
S3 (8B + Phi4)	39.1%	4,161	9.39
S4 (8B + Qwen)	31.8%	6,127	5.18
Qwen3-4B nothink	53.0%	1,943	**27.28**
Qwen3-4B-Instruct nothink	71.8%	3,573	**20.10**

S5.3 Conclusion and Analysis

S5.3.1 Pathological TokenProbe Reversal

In RL-trained baseline models, higher TokenProbe (the proportion of tokens with above-average log probability) correlates positively with correctness ($r \approx +0.25$ to $+0.31$). SFT distillation reverses this relationship in all four settings.

Table S40: TokenProbe reversal in SFT models. The sign of both correlations flips, indicating that the logprob signal no longer carries the same semantic meaning after SFT distillation.

Model	Correct TokenProbe	Incorrect TokenProbe	$r(\text{TokenProbe}, \text{correct})$	$r(\text{tokens}, \text{TokenProbe})$
4B think (RL baseline)	0.784	0.762	$+0.307$	-0.415
8B think (RL baseline)	0.790	0.774	$+0.254$	-0.367
S1 (4B + Phi4)	0.810	0.891	$\mathbf{-0.404}$	$\mathbf{+0.858}$
S2 (4B + Qwen)	0.800	0.878	$\mathbf{-0.464}$	$\mathbf{+0.599}$
S3 (8B + Phi4)	0.789	0.883	$\mathbf{-0.525}$	$\mathbf{+0.866}$
S4 (8B + Qwen)	0.782	0.852	$\mathbf{-0.422}$	$\mathbf{+0.492}$

In RL baselines, correct samples have higher TokenProbe than incorrect ones (the model is more confident when right). In all four SFT settings, incorrect samples have *substantially higher* TokenProbe than correct ones, meaning the model is most confident when wrong. The correlation $r(\text{TokenProbe}, \text{correctness})$ shifts from $+0.3$ (RL) to -0.5 (SFT), a total swing of 0.8. Similarly, $r(\text{tokens}, \text{TokenProbe})$ reverses from -0.4 to $+0.9$. In RL models, longer responses tend to carry lower confidence, consistent with genuine exploration; in SFT models, longer responses instead have *higher* confidence, consistent with degeneration loops that fill the token budget with repetitive high-confidence patterns.

S5.3.2 Degeneration Loop Microstructure

We inspect the logprob microstructure to understand the mechanism behind TokenProbe reversal.

Table S41: Degeneration loop statistics. Incorrect samples exhibit flattened logprob distributions, rare zero-crossings, and extremely long positive runs, all of which are hallmarks of repetitive degeneration.

Setting	Category	lp_std	ZC/100tok	Max pos. run	2nd-half TokenProbe
S1	Correct (n=10)	0.398	22.8	124	0.854
S1	Incorrect (n=30)	0.240	12.0	7,522	0.920
S3	Correct (n=11)	0.440	24.0	74	0.824
S3	Incorrect (n=29)	0.276	12.6	7,811	0.915
S4	Correct (n=7)	0.454	24.6	545	0.823
S4	Incorrect (n=33)	0.353	15.0	7,096	0.911

The statistics in Tab. S41 reveal three recurring signatures.

1. **Logprob flattening**. Incorrect samples' logprob standard deviation is only 60 to 78% of correct samples', indicating near-uniform confidence across all tokens, which is a signature of repetitive pattern generation.
2. **Monotonic positive runs**. The longest contiguous segment of above-average logprob in incorrect samples reaches 7,000 to 7,800 tokens. The model sustains "full confidence" for thousands of consecutive tokens while producing entirely wrong output.
3. **Second-half TokenProbe escalation**. Incorrect samples' TokenProbe in the second half reaches 0.91 to 0.92, compared to 0.82 to 0.85 for correct samples, indicating that degeneration becomes more pronounced as generation proceeds.

S5.3.3 Detailed Case Studies

We present three case studies that illustrate distinct failure modes. All examples use the same prompts across settings.

Case Study 1. MATH #12, Four-Way Comparison. *Problem*. "The proper divisors of 12 are 1, 2, 3, 4 and 6. A proper divisor of an integer N is a positive divisor of N that is less than N. What is the sum of the proper divisors of the sum of the proper divisors of 284?" *Ground truth*. 284.

Table S42: MATH #12 across all 4 SFT settings. The same problem elicits four completely different failure modes.

Setting	Tokens	TokenProbe	Behavior
S1 (4B+Phi4)	451	0.885	Compact think section. Factorization → sum → correct answer $\boxed{284}$
S2 (4B+Qwen)	8,192	0.894	Empty think → correct reasoning initially, then enters infinite loop. "284's sum of proper divisors is 220. 220's sum of proper divisors is 284." repeated until truncation
S3 (8B+Phi4)	484	0.872	Think section present, correct approach, but arithmetic error. Sums 220's divisors to 234 instead of 284 → $\boxed{234}$
S4 (8B+Qwen)	8,192	0.948	Empty think → "This is a complex or challenging question…" → brute-force enumeration checking every integer from 4 to 365 → truncated

Together, the four outcomes illustrate the range of failure modes induced by SFT distillation.

S1 (the only correct setting) produces a compact 451-token solution using the think section for quick factorization.

S2 demonstrates *degeneration loops*. The model correctly computes that 284 and 220 are amicable numbers, but then enters an infinite loop alternating between the two facts. Its second-half TokenProbe is 1.000 (every token in the second half is above the mean logprob), showing that the model remains highly confident while producing an endless repetition that consumes all 8,192 tokens without ever outputting a boxed answer.

S3 illustrates a subtler failure: *loss of self-correction capability*. The model's approach is correct (find divisors of 220 and sum them), but it makes an arithmetic error ($1+2+4+5+10+11+20+22+44+55+110=284$, not 234). An RL-trained model would typically catch such errors through "Wait" self-check loops. The SFT model lacks this self-correction behavior because distillation removed the "Wait" tokens from the training data (they fall in low-NormLP regions and are filtered out).

S4 shows *complete strategic collapse*. Unable to use prime factorization (which requires the structured reasoning destroyed by distillation), the model resorts to brute-force trial division, checking *every integer* from 4 to 365 as a potential divisor of 284. After 8,192 tokens, it has only reached divisor candidate 365 and is truncated without producing an answer. Its TokenProbe of 0.948 (highest among

all four settings) reflects the mechanically repetitive nature of brute-force enumeration, where every "N. $N \times 2 = M$, too big, nope" follows the identical pattern with maximal confidence.

Case Study 2. MATH #377, Extreme Degeneration. *Problem*. "The superfactorial $n\$$ is defined as $n\$ = \underbrace{n!^{n!^{\cdot^{\cdot^{\cdot^{n!}}}}}}_{n!}$. What is the units digit of $4\$$?" *Ground truth*. 6. *S3 prediction*. EXTRACTION_FAILED.

The model correctly identifies that $4! = 24$ and begins constructing the tower $24^{(24^{(24^{\cdots})}}$, but then falls into an infinite loop, generating "24^(24^(24^(..." hundreds of times. The resulting statistics are as follows.

- **Token length**. 8,192 (truncated at limit)
- **TokenProbe**. 0.969 (among the highest in the entire dataset)
- **Max positive run**. 7,010 tokens, meaning that for 7,010 consecutive tokens, every single token is above the mean logprob
- **Zero-crossings**. 3.0 per 100 tokens (vs. 24.0 for correct S3 samples)
- **Second-half TokenProbe**. 1.000, indicating that in the second half, the logprob of *every* token exceeds the mean

This case illustrates the core pathology. The model has learned to generate a syntactically plausible mathematical expression (exponent towers) with high confidence, but no longer reasons about the mathematical content needed to determine units digits via modular arithmetic. The degeneration is therefore largely invisible to logprob-based quality metrics: by such measures, the response appears high quality.

Case Study 3. MATH #52, Correct vs. Infinite Loop. *Problem*. "Simplify. $\frac{\sqrt{2.5^2-0.7^2}}{2.7-2.5}$." *Ground truth*. 12.

- **S3** (8B + Phi4). Correct in 568 tokens. Uses difference-of-squares factorization in the think section. $2.5^2 - 0.7^2 = (2.5 - 0.7)(2.5 + 0.7) = 1.8 \times 3.2 = 5.76$, $\sqrt{5.76} = 2.4$, answer $= 2.4/0.2 = 12$. TokenProbe = 0.803, zero-crossings = 24.3/100tok, reflecting a healthy and diverse reasoning trace.
- **S2** (4B + Qwen). Incorrect, 8,192 tokens. Empty think block, then enters a loop where "$6.25 - 0.25 = 6.25 - 0.25 = \ldots$" repeats 286 times. The model correctly identifies $2.5^2 = 6.25$ and $0.7^2 = 0.49$ but then *miscomputes* $6.25 - 0.49$ as "$6.25 - 0.25$" and loops endlessly on this error because it has no self-correction mechanism.

This comparison further suggests that think tags and structured reasoning preserve a compact self-correction pathway, whereas the setting without think tags makes an arithmetic error and then loops because it has lost the ability to self-correct.

S5.3.4 Why SFT Fails While RL Succeeds. Structural Analysis

The failures above are not implementation deficiencies fixable by hyperparameter tuning. They stem from six structural limitations of the SFT distillation paradigm.

1. **Self-correction marker destruction.** The logprob-based distillation filter removes over 90% of self-correction markers ("Wait", "Actually", "Let me re-check", etc.) from the training data. In the Phi4 dataset, the average number of self-correction markers per sample drops from 8.53 to 0.58 (6.8% retention). In the Qwen dataset, from 33.27 to 3.19 (9.6% retention). This occurs because self-correction tokens, which represent moments of genuine model uncertainty, tend to have negative NormLP, placing them in the same region as wasteful redundancy. The filter cannot distinguish between a valuable "Wait, let me reconsider the geometry..." (course correction after discovering a misconception) and a wasteful "Wait, let me recalculate... wait, let me recalculate again..." (repeatedly dwelling on a confirmed conclusion).
2. **Logprob filter ambiguity.** Both valuable exploration and wasteful redundancy present as low-NormLP tokens. Valuable corrections tend to occur at moments of model "uncertainty" (negative NormLP), and valueless repetitions also occur at moments of model "uncertainty." The logprob filter is blind along this critical dimension and can only apply indiscriminate

deletion. This reflects a structural limitation: no static threshold on NormLP can separate the two categories because they overlap in logprob space.

3. **Causal context destruction.** When the filter removes tokens from the middle of a reasoning chain, the remaining tokens are concatenated into a sequence that was *never generated by any model*, as illustrated below.

$$\underbrace{A \to B \to C \to [\text{Wait, C is wrong}] \to D \to E}_{\text{Original}} \quad \Longrightarrow \quad \underbrace{A \to B \to D \to E}_{\text{Distilled}}$$

   In the original chain, D was generated conditioned on the correction of C. In the distilled chain, D follows B directly, a transition that never occurred in any natural reasoning process. The model is forced to learn $P(D \mid A, B)$ when the correct generation actually depends on $P(D \mid A, B, C, [\text{correction}])$, a much richer context.

4. **NTP loss lacks credit assignment.** The next-token prediction loss [56] $\mathcal{L}_{\text{NTP}} = -\frac{1}{T}\sum_{t=1}^{T} \log P_\theta(x_t \mid x_{<t})$ applies equal weight to every token. There is no mechanism to distinguish core reasoning steps from redundant filler. In contrast, RL's GRPO [64] loss weights tokens by advantage $\hat{A}_t$, enabling statistical credit assignment across multiple rollouts. Patterns consistently appearing in successful trajectories are reinforced. Patterns consistently appearing in failed trajectories are suppressed.

5. **No feedback loop.** SFT training operates on a fixed, offline dataset. Once the distillation filter makes its decisions, the model has no mechanism to "contest" incorrect deletions during training. RL operates online. The model generates rollouts, receives rewards, and updates. If skipping self-correction leads to wrong answers (negative reward) while performing self-correction leads to correct answers (positive reward), the model learns when correction is valuable through thousands of such trial-and-error cycles.

6. **Compounded exposure bias.** Standard SFT already suffers from exposure bias [58, 6]. Training uses teacher forcing (ground-truth prefix), while inference uses autoregressive generation (model's own prefix). Distillation worsens this. The training data contains artificially compressed token transitions (from the filter), which diverge even further from the model's natural generation patterns at test time. The result is that small deviations at inference time cascade into large errors, with no recovery mechanism.

Why RL avoids these pitfalls. RL preserves complete causal context (soft penalty via reward signals, never hard deletion), enables statistical credit assignment (GRPO [64] advantage estimation across rollouts), trains on the model's own distribution (no exposure bias), and allows new behaviors to emerge. DeepSeek-R1 [21] demonstrated that self-correction behaviors can emerge from scratch through pure RL training, without SFT data containing such behaviors. The distinction is therefore structural: **SFT deletes information and then imitates the compressed trace, whereas RL preserves the full trajectory and reinforces it differentially.**

S5.4 Outlook. SFT as RL Warm-Start

Although pure SFT distillation is unviable as a standalone training method, these experiments offer two valuable insights for the RL path.

SFT as cold-start initialization. The SFT distillation data retains value not as the final training objective, but as an initialization for RL training. SFT can provide an initial bias toward compact reasoning style, nudging the model to prefer shorter, more structured traces, while RL's online exploration subsequently recovers reasoning quality through reward-signal correction of the errors and degeneration introduced by SFT. This follows the established pipeline of DeepSeek-R1 [21], which uses SFT warm-start before GRPO [64] training. The key caveat is that one must verify after SFT warm-start that $r(\text{TokenProbe}, \text{correctness})$ remains positive, because otherwise the NormLP-based reward shaping would operate on an inverted signal.

New perspectives on RL efficiency training. The SFT experiments clarify why token efficiency is difficult to obtain through post-hoc data compression alone. The value of Chain-of-Thought reasoning often lies in the apparently "redundant" exploration process that enables self-correction. This insight motivates the RL-based approach: rather than compressing existing reasoning traces

offline, the model should learn *naturally efficient* reasoning strategies through online exploration with efficiency-aware rewards. Our data shows that models answering correctly use fewer tokens (Pearson $r(\text{tokens}, \text{correct}) \approx -0.59$ to -0.74), suggesting that the objective is not to remove self-correction, but to make it more precise: correcting decisively when needed and advancing decisively otherwise.

Alignment when transferring teacher distributions. If future warm-start methods transfer token distributions between models with different vocabularies, token alignment becomes an additional design consideration. Probabilistic Token Alignment for Large Language Model Fusion [93] formulates this alignment through optimal transport and provides a soft mapping for model fusion. Such distribution-aware alignment could inform future teacher–student transfer, although its ability to preserve self-correction in compressed reasoning traces would require separate evaluation.

S6 Exploring Token Compression Limits

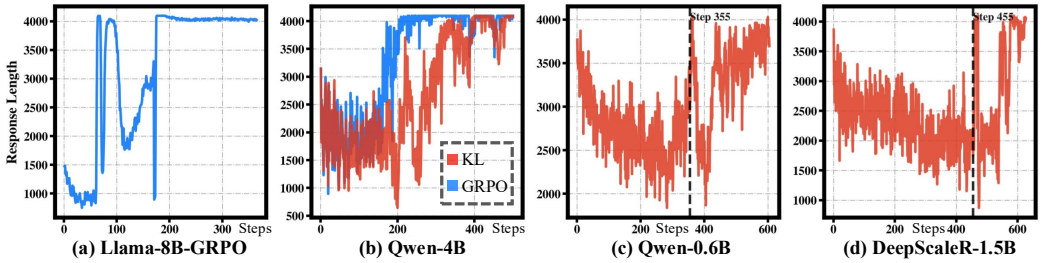


Figure S7: **Failure modes of length-controlled RL training. (A)** Llama-3.2-8B-Instruct under vanilla GRPO. Response length oscillates wildly and locks at the 4096-token cap after ∼200 steps. **(B)** Qwen3-4B-Instruct under both GRPO (blue) and our KL variant (red). Both runs saturate at the cap with degenerate, repetitive outputs. **(C, D)** Qwen3-0.6B-Base and DeepScaleR-1.5B trained with our method. Training is stable until the running compression scale crosses $\approx 1.6\times$ (dashed line, at step 355 for 0.6B and step 455 for 1.5B), after which length explodes, accuracy collapses to zero, and outputs degenerate into pure repetition.

Failure 1. Instruction-tuned models collapse under vanilla GRPO. Figure S7(A) shows Llama-3.2-8B-Instruct trained with plain GRPO, without any compression-related modification. After an initial dip the response length oscillates violently and then saturates at the 4096-token cap, never recovering. We attribute this to two compounding effects of instruction tuning. First, SFT on instruction data sharply narrows the output entropy. The policy has already committed to a small set of high-probability templates, leaving little exploration headroom for the on-policy GRPO updates. Once a few reward gradients push the policy off these templates, the loss landscape becomes brittle and the model transitions abruptly between disparate output styles rather than improving smoothly. Second, instruction-tuned checkpoints carry strong stylistic priors, including preambles, headings, hedging phrases, and answer formatting, that act as implicit length regulators. Once the verifiable reward starts breaking those priors, the latent format constraint disappears and length grows without bound. Stable RLHF/RLVR pipelines on instruction models typically depend on heavy KL regularization to a frozen SFT reference, conservative learning-rate schedules, and reward models that explicitly penalize verbosity, none of which are present in vanilla GRPO. This is consistent with the common practice in the recent reasoning-RL literature of starting from base rather than instruction-tuned checkpoints.

Failure 2. Qwen3-4B-Instruct exhibits the same pathology with both GRPO and our KL method. Figure S7(B) confirms that this failure is not Llama-specific. On Qwen3-4B-Instruct, both vanilla GRPO and our KL variant converge to the same terminal state. Length pinned at 4096 and outputs collapsed onto a narrow set of repeating n-grams. Manual inspection of late-training rollouts reveals that the model produces the same phrase or short paragraph dozens of times until the budget is exhausted. Once this regime is entered the verifier reward signal carries essentially no useful gradient information. The mechanism is a positive feedback loop characteristic of neural-text degeneration. The instruction-tuned policy starts in a low-entropy basin. The compression objective rewards short *correct* answers but provides no penalty for repetitive padding when the answer is wrong. As accuracy degrades the model pads more, lowering reward variance and further attenuating

gradients. Crucially, in this setting our KL anchor is the instruction-tuned policy itself, so anchoring offers no protection. It pulls the policy back toward a reference that is already prone to degeneration.

Failure 3. A hard compression limit at $\approx 1.6\times$. Figures S7(C) and S7(D) reveal a more fundamental boundary that appears even on base models trained with our full method. Qwen3-0.6B-Base (C) and DeepScaleR-1.5B (D) train stably for several hundred steps, with average response length decreasing as expected. However, the running *compression scale*, defined as the ratio of the initial average CoT length to the current average length, keeps climbing, and at the moments marked by the dashed lines (step 355 for 0.6B and step 455 for 1.5B) it crosses approximately $1.6\times$. At that crossing both runs undergo the same transition. A sharp burst of variance, a brief partial recovery, and then a runaway increase in length until the cap is hit. Post-collapse rollouts are uniformly degenerate (zero accuracy, length saturated at the maximum, outputs purely repetitive), mirroring the instruction-model failures of panels A and B. We interpret $1.6\times$ as an empirical lower bound on the information density of the reasoning trace. Below the corresponding length, the chain-of-thought no longer carries enough intermediate computation to support the final answer, the verifier reward becomes effectively random, and the policy loses any signal that would keep it in the compressed regime. In our experiments, this boundary appears similar across the 0.6B and 1.5B checkpoints despite their different absolute reasoning budgets. This finding aligns with a growing body of work arguing that chain-of-thought length is not cosmetic but corresponds to actual sequential computation, and that for problems of a given difficulty there exists a minimum number of intermediate tokens below which an autoregressive solver cannot succeed. Once that floor is crossed, the model abandons reasoning altogether and emits the maximum-length repetitive output that minimizes the immediate negative reward.

Is the $\sim 1.6\times$ floor task-dependent? The $1.6\times$ value is observed on Qwen3-0.6B-Base and DeepScaleR-1.5B trained on DeepScaleR (math-only) and evaluated on the four math benchmarks of Tab. 2. We describe it as *backbone-agnostic* across these two checkpoints, not *task-agnostic*. Mechanistically the floor reflects the minimum number of intermediate tokens an autoregressive solver requires for problems drawn from a particular difficulty distribution, which is necessarily task-dependent. The supplementary budget-gradient analysis in §S11.2 is consistent with this view. MATH-500, the easiest of the four math benchmarks for our checkpoints, remains partially solvable down to 1K tokens (Step-Flash-196B-11B retains 48.6% on MATH-500 at the 1K cap. Tab. S56), while AIME-2025 collapses to 0% on every one of the five large models once the cap drops below 4K (Tab. S55). The *compression floor* is therefore lower on easier distributions and higher on harder ones. Reporting one number tied to a single distribution, DeepScaleR-style competition mathematics, is what we mean by "backbone-agnostic." We did not retrain at the frontier on non-math tasks because the $1.6\times$ collapse is destructive (Fig. S7(C, D)) and replicating these failure runs per task would be prohibitive. Characterising the floor as a function of task difficulty distribution is left to future work.

S7 Limitations and Future Work

S7.1 Theoretical Certification

Our analysis identifies a strong and consistent empirical association between TokenProbe, reasoning accuracy, and token usage across model-level, quartile-level, and response-level measurements. The present work focuses on validating and leveraging this association as a practical inference-native signal for efficient reasoning. A natural direction for future work is to develop a deeper mechanistic account of why normalized token-level confidence aligns with reasoning quality and token efficiency. This question is also open for related log-probability-, entropy-, and confidence-based reasoning diagnostics, which are often motivated by empirical correlations with downstream quality indicators. We view our findings as an empirical basis for such future theoretical certification, while the proposed framework remains directly usable as a diagnostic and optimization tool.

S7.2 General Reasoning Efficiency

We also conducted a lightweight transfer diagnostic to examine whether the TokenProbe trend is tied only to mathematical reasoning. The diagnostic uses two domains. Math, consisting of AIME2025, AMC, MATH, and OlympiadBench, and General, consisting of MMLU, MMLU-Pro, LSAT, and GPQA. For each domain, we use 10 fixed samples per dataset, giving 40 samples per domain. We

compare three Qwen3-4B variants. The base model, the KL-trained variant, and the RS-trained variant.

Table S43: **TokenProbe on fixed 40-sample transfer diagnostics.** Each domain contains four datasets with 10 fixed samples per dataset. We report only TokenProbe.

Domain	Base	w/ KL	w/ RS
Math	0.782	0.887	0.846
General	0.736	0.846	0.802

The same directional pattern appears in both domains. TokenProbe increases after applying either KL or RS. This provides trend-level evidence that the efficiency behavior is relatively task-agnostic rather than tailored only to math reasoning. We view this as an initial diagnostic rather than a complete generality claim, and future work will extend the study of general efficiency to broader settings such as code generation and agentic reasoning.

Efficient adaptation across tasks and modalities. Parameter-efficient adaptation [22] offers further settings in which to examine whether TokenProbe remains informative after a change in training regime. MEPT [94] uses multiple prompt experts to adapt to diverse data distributions, while Visual Fourier Prompt Tuning [92] incorporates frequency-domain information into visual prompts for adapting vision Transformers. These approaches motivate future studies of combining efficient adaptation with efficient reasoning. For visual or multimodal settings, applying TokenProbe would require a generated reasoning trace with accessible token log-probabilities; the present experiments do not establish its validity for visual prompt tokens.

S8 Reproducibility, Compute, and Responsible Use Details

S8.1 Open Access to Data and Code

The training data used for our efficient-GRPO runs is DeepScaleR-Preview-Dataset, which is publicly available on Hugging Face at `https://huggingface.co/datasets/agentica-org/DeepScaleR-Preview-Dataset`. The dataset card lists the license as MIT. The appendix provides pseudocode for the core implementation components: core-window selection in 1, selective KL anchoring in 2, and reward shaping in 3.

S8.2 Statistical Significance

To estimate run-to-run variability for the main efficient-GRPO conclusions, we rerun two representative configurations three times and report the sample standard deviation across runs, using denominator $n - 1$. The reported values measure variability across repeated training/evaluation runs under the same benchmark protocol. S44 shows that the average-accuracy variance is small for both a low-capacity KL setting and a mid-scale RS setting, while per-benchmark variance is larger on AIME because it contains only 30 problems.

Table S44: **Run-to-run standard deviation.** Each entry is the sample standard deviation across three repeated runs. We report standard deviation of accuracy in percentage points.

Model	Average	AIME	MATH	AMC	Olympiad
Qwen-0.6B w/ KL	0.90	1.96	1.70	1.83	0.95
Qwen-4B w/ RS	0.20	5.10	1.01	3.17	1.85

S8.3 Compute Resources

All main training runs use one node with $8\times$ NVIDIA H100 80GB GPUs. The training recipe is otherwise the same as in S4.1. S45 reports the average wall-clock time for each backbone family. These times cover the main efficient-GRPO training runs and do not include exploratory failed runs, closed-API evaluation time, or offline analysis scripts.

Table S45: **Compute resources for main training runs.** All runs use a single node with $8\times$ NVIDIA H100 80GB GPUs.

Backbone	Hardware	Average training time
Qwen-0.6B	$8\times$ H100 80GB	12h 14m
DeepScaleR-1.5B	$8\times$ H100 80GB	9h 10m
Qwen-4B	$8\times$ H100 80GB	21h 56m
Phi-Reasoning-4B	$8\times$ H100 80GB	25h 13m
Qwen-8B	$8\times$ H100 80GB	38h 20m

The TokenProbe-specific operations add negligible training overhead. As reported in S32, the extra CPU-side computation accounts for 0.009%–0.210% of a representative GRPO step across trace lengths from 1K to 16K tokens. At inference time, the overhead is zero because the trained checkpoint is decoded with an unmodified generator.

S8.4 Broader Impacts

The primary positive impact of TokenProbe-guided efficient reasoning is reduced inference cost. By lowering generated-token usage while preserving reasoning accuracy, the method can reduce serving cost and energy consumption and can make reasoning-capable models more accessible in resource-constrained settings. This is especially relevant for open sub-10B models, where lower inference cost can make deployment feasible without relying on large closed APIs.

The same efficiency gains may also lower the cost of misuse. Cheaper reasoning can make automated generation, persuasion, or decision-support pipelines easier to scale, including undesirable uses. In addition, the method does not eliminate standard risks of language models: a compressed reasoning trace may still be incorrect, incomplete, or overconfident. We therefore view TokenProbe as an efficiency and diagnostic method, not as a safety or factuality guarantee. This work does not release a new high-risk foundation model; it studies a training objective and diagnostic signal built on existing models and public benchmarks.

S8.5 Licenses for Existing Assets

We use existing models, datasets, and software frameworks, and credit the corresponding creators through citations and model/dataset identifiers in Tables S1, S2, and S3. S46 summarizes the licenses or terms listed by the corresponding public model cards or provider documentation.

S8.6 Declaration of LLM Usage

LLMs were used to assist with editing, including grammar, spelling, and word choice, with data processing and filtering, and with understanding technical concepts during the research and writing process. LLM-generated text was not used as an unverified source of experimental results. All quantitative claims reported in the paper are based on the experiments and analyses described in the main text and appendix.

Table S46: **Licenses and terms for existing assets.** License entries follow the corresponding public model or dataset cards when available. Closed-API models are governed by provider API terms rather than open model licenses.

Asset	Identifier / source	License / terms	Role
DeepScaleR-Preview-Dataset	`agentica-org/DeepScaleR-Preview-Dataset`	MIT	Training data
Qwen family	`Qwen/Qwen3-*`, `Qwen/Qwen3.5-*`, and `Qwen/Qwen3-Next-*`	Apache-2.0	Training and open baselines
DeepScaleR-1.5B	`agentica-org/DeepScaleR-1.5B-Preview`	MIT	Training backbone
Phi-4-mini-reasoning	`microsoft/Phi-4-mini-reasoning`	MIT	Training backbone
DeepSeek-R1-Distill-Qwen	`deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B`	MIT	Open baseline
GPT-OSS	`openai/gpt-oss-120b`	Apache-2.0	Open baseline
GLM family	`zai-org/GLM-4.5-Air`, `zai-org/GLM-4.7`, and `zai-org/GLM-5`	MIT	Open baselines
Step-3.5-Flash	`stepfun-ai/Step-3.5-Flash`	Apache-2.0	Open baseline
DeepSeek V3/V4 family	`deepseek-ai/DeepSeek-V3.2`, `deepseek-ai/DeepSeek-V4-Pro`, and `deepseek-ai/DeepSeek-V4-Flash`	MIT	Open baselines
Kimi-K2-Thinking	`moonshotai/Kimi-K2-Thinking`	Custom / other license	Open baseline
Hunyuan3	`tencent/Hy3-preview`	Custom / other license	Open baseline
Closed-API models	GPT, Claude, Gemini, Grok, and Qwen API models	Provider API terms	Closed baselines
veRL and vLLM	`verl` and `vllm` software frameworks	Apache-2.0	Training and inference infrastructure

S9 Supplementary Validation Experiments

S9.1 Within-problem analysis controlling for response-level factors

We analyzed responses from the original Qwen3-4B thinking model on MATH and OlympiadBench. Within each dataset, we standardized TokenProbe, log response length, mean token log probability, and 4-gram repetition coverage. We then demeaned correctness and these predictors within each problem and fit a no-intercept regression. This compares responses to the same problem while controlling for length, generation confidence, and repetition. TokenProbe retained a positive coefficient on both datasets ($\beta_{\mathrm{TP}} = 0.277$ on MATH and 0.148 on OlympiadBench), indicating an incremental association with correctness after these controls.

S9.2 Causal interventions on high-NormLP regions

We used Qwen3-4B on MATH and OlympiadBench and compared high-NormLP, low-NormLP, and random windows under the same token budget. For token deletion, we removed the selected tokens from the reasoning trace, retained the original prompt and the `</think>` boundary, removed the original final answer, and regenerated an answer with deterministic decoding. For attention-key masking, we preserved the tokens but prevented the selected positions from serving as attention keys for subsequent reasoning and answer tokens during prefill.

At a 20% intervention budget, the reported correctness damage was:

Table S47: **Correctness damage under token deletion and attention-key masking.** Entries in each method column are ordered High-NormLP / Random / Low-NormLP.

Dataset	Token deletion	Attention-key masking
MATH	2.6% / 1.1% / 0.8%	2.4% / 0.8% / 0.3%
OlympiadBench	1.8% / 0.7% / 0.5%	1.5% / 0.6% / 0.4%

High-NormLP deletion caused 1.5% and 1.1% more damage than random deletion on MATH and OlympiadBench, respectively. For attention-key masking, the corresponding differences from the tabulated results were 1.6% and 0.9%. Both interventions showed the ordering High-NormLP $>$ Random $>$ Low-NormLP, providing causal evidence that high-NormLP regions are more consequential to downstream answers in these evaluations.

S10 Supplementary Closed-API Reasoning Results

S10.1 Closed-source models and API prices

Table S48: **Closed-source baseline models used in our experiments.** Version denotes the model version number. Links point to official announcements or model pages. Output prices are standard public list prices in USD per million tokens, as of September 24, 2026.

Model Name	API Identifier	Version	Mode	Output price	Official link
OpenAI					
GPT-4o	`gpt-4o`	4o	Default	$10.00	Announcement
OpenAI o1	`o1`	o1	Reasoning	$60.00	Announcement
OpenAI o3	`o3`	o3	Reasoning	$8.00	Announcement
GPT-5.2 Pro	`gpt-5.2-pro`	5.2	Pro	$168.00	Announcement
GPT-5.4 Mini	`gpt-5.4-mini`	5.4	Mini	$4.50	Announcement
GPT-5.4 Pro	`gpt-5.4-pro`	5.4	Pro	$180.00	Announcement
GPT-5.5	`gpt-5.5`	5.5	Standard	$30.00	Announcement
GPT-5.5 Pro	`gpt-5.5-pro`	5.5	Pro	$180.00	Announcement
GPT-5.6 Sol	`gpt-5.6-sol`	5.6	Sol	$20.00	Announcement
GPT-5.6 Terra	`gpt-5.6-terra`	5.6	Terra	$12.00	Announcement
GPT-5.6 Luna	`gpt-5.6-luna`	5.6	Luna	$1.20	Announcement
xAI					
Grok-4.1-Fast	`grok-4-1-fast-reasoning`	4.1	Fast	$0.50	Announcement
Grok-4.3	`grok-4.3`	4.3	Default	$2.50	Model page
Grok-4.20	`grok-4.20-reasoning`	4.20	Reasoning	$2.50	Model page
Grok-4.20	`grok-4.20-non-reasoning`	4.20	Non-reasoning	$2.50	Model page
Grok-4.20-0309	`grok-4.20-0309-reasoning`	4.20-0309	Reasoning	$2.50	Model card
Grok-4.20-0309	`grok-4.20-0309-non-reasoning`	4.20-0309	Non-reasoning	$2.50	Model card
Google					
Gemini-3-Flash	`gemini-3-flash-preview`	3	Flash	$3.00	Model guide
Gemini-3.1-Pro	`gemini-3.1-pro-preview`	3.1	Pro	$12.00	Model guide
Gemini-3.1-Flash-Lite	`gemini-3.1-flash-lite`	3.1	Flash-Lite	$1.50	Model page
Anthropic					
Claude-4.5-Haiku	`claude-haiku-4-5`	4.5	Haiku	$5.00	Announcement
Claude-4.6-Sonnet	`claude-sonnet-4-6`	4.6	Sonnet	$15.00	Announcement
Claude-4.6-Opus	`claude-opus-4-6`	4.6	Opus	$25.00	Announcement
Alibaba Cloud					
Qwen-3.6-Plus	`qwen3.6-plus`	3.6	Plus	$1.651	Model page

Output-token prices use standard public API rates and exclude input tokens, caching discounts, batch discounts, and tool fees. GPT-5.6 reasoning-effort variants share the same model price. Where prices vary with context length, the listed rates are for prompts below each provider's threshold.

S10.2 Per-task results

Table S49 reports the four in-domain benchmarks. Table S50 reports the four out-of-domain benchmarks.

Table S49: **Per-task results on in-domain reasoning benchmarks. #Tk.** is the average number of generated completion tokens per response, including billable reasoning tokens where present. **Acc.** is accuracy in percent.

Model	**Average**		**AIME-2025**		**MATH-500**		**AMC**		**OlympiadBench**	
	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.
Previously reported closed-API baselines										
GPT-5.4-Mini	987	69.6	1,714	36.7	589	88.6	1,137	73.5	1,232	56.6
Claude-4.6-Sonnet	1,221	61.3	1,907	20.0	801	86.0	1,413	51.8	1,478	46.1
Grok-4.1-Fast	4,945	52.3	9,361	10.0	1,866	74.8	4,017	43.4	7,144	38.5
Gemini-3.1-Pro	1,586	54.4	2,020	10.0	1,274	79.2	1,798	37.3	1,771	40.1
Gemini-3-Flash	1,828	35.1	2,039	3.3	1,625	56.0	1,948	24.1	1,954	22.4
Claude-4.6-Opus	1,130	70.0	1,852	30.0	689	93.2	1,375	65.1	1,395	55.1
Claude-4.5-Haiku	1,339	72.0	1,803	23.3	1,061	89.8	1,533	66.3	1,500	61.6
Gemini-3.1-Flash-Lite	778	80.4	1,036	36.7	614	94.6	893	81.9	875	71.6
OpenAI evaluations										
GPT-4o	619	49.9	741	0.0	507	69.8	732	42.2	682	38.4
GPT-5.2 Pro	978	64.4	1,679	43.3	601	83.8	1,116	85.5	1,209	48.4
GPT-5.4 Pro	991	63.8	1,707	43.3	624	83.4	1,100	83.1	1,222	47.6
GPT-5.5	801	67.9	1,419	70.0	427	85.8	796	90.4	1,051	51.7
GPT-5.5 Pro	981	62.0	1,723	36.7	585	82.0	1,052	79.5	1,233	46.2
GPT-5.6 Luna	761	59.9	1,359	66.7	402	77.4	735	88.0	1,004	43.1
GPT-5.6 Sol	652	63.2	1,211	76.7	302	79.2	602	84.3	893	48.1
GPT-5.6 Sol (high)	695	63.0	1,365	56.7	332	80.4	647	85.5	940	47.6
GPT-5.6 Sol (low)	587	65.0	1,074	80.0	272	80.0	526	85.5	806	50.7
GPT-5.6 Sol (max)	837	56.8	1,556	43.3	444	75.8	870	77.1	1,092	40.9
GPT-5.6 Sol (xhigh)	747	60.0	1,445	56.7	365	78.2	710	84.3	1,004	43.7
GPT-5.6 Terra	628	65.4	1,176	83.3	292	81.2	551	92.8	862	49.5
OpenAI o1	1,390	40.5	1,976	0.0	994	62.4	1,651	28.9	1,625	27.4
OpenAI o3	1,282	44.4	1,947	13.3	886	63.8	1,497	50.6	1,518	30.7
xAI evaluations										
Grok-4.20 (non-reasoning)	1,480	38.9	2,008	10.0	1,075	58.6	1,746	37.3	1,723	25.8
Grok-4.20 (reasoning)	4,244	72.9	8,450	96.4	2,208	76.6	4,967	98.8	5,615	65.3
Grok-4.20-0309 (non-reasoning)	1,477	39.4	2,006	10.0	1,081	60.0	1,763	32.5	1,712	26.2
Grok-4.3	1,703	64.8	3,036	60.0	1,065	73.0	1,844	94.0	2,099	55.4

Table S50: **Per-task results on out-of-domain generalization.** The four tasks are MMLU, MMLU-Pro, LSAT, and GPQA. **#Tk.** is the average number of generated completion tokens per response, including billable reasoning tokens where present. **Acc.** is accuracy in percent.

Model	**Average**		**MMLU**		**MMLU-Pro**		**LSAT**		**GPQA**	
	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.
Previously reported closed-API baselines										
Grok-4.1-Fast	2,892	88.2	1,026	90.9	2,288	84.1	2,797	96.1	5,459	81.8
Claude-4.5-Haiku	1,090	72.0	706	88.6	883	79.8	1,521	54.8	1,250	64.7
Gemini-3.1-Pro	1,294	68.6	681	92.8	1,055	80.3	1,727	54.4	1,714	47.0
Claude-4.6-Sonnet	1,290	57.1	645	88.8	965	73.0	1,797	35.7	1,754	30.8
Gemini-3-Flash	1,651	29.1	1,123	64.9	1,593	33.9	1,915	12.6	1,973	5.1
OpenAI evaluations										
GPT-4o	443	71.9	345	86.8	474	70.2	563	38.3	646	43.9
GPT-5.2 Pro	495	84.7	215	92.8	437	82.0	1,393	82.2	1,162	60.6
GPT-5.4 Pro	667	85.7	378	93.1	670	83.0	1,371	86.5	1,307	61.1
GPT-5.5	373	88.7	208	94.4	363	85.1	709	96.1	865	69.2
GPT-5.5 Pro	675	83.9	429	92.4	685	80.7	1,115	87.0	1,347	53.0
GPT-5.6 Luna	—	—	169	24.3	341	13.6	726	70.9	—	—
GPT-5.6 Sol	—	—	138	22.1	255	25.6	474	84.8	—	—
GPT-5.6 Sol (high)	—	—	162	17.7	317	24.2	526	86.1	—	—
GPT-5.6 Sol (low)	—	—	103	28.3	201	31.0	418	88.3	—	—
GPT-5.6 Sol (max)	—	—	284	16.0	495	22.0	654	82.2	—	—
GPT-5.6 Sol (xhigh)	—	—	188	16.0	371	22.2	556	82.6	—	—
GPT-5.6 Terra	—	—	94	26.1	202	26.7	377	51.3	—	—
OpenAI o1	989	61.0	553	84.2	1,056	60.2	1,942	0.0	1,748	18.7
OpenAI o3	774	68.5	425	88.9	757	64.4	1,722	31.7	1,518	28.8
xAI evaluations										
Grok-4.20 (non-reasoning)	1,229	67.3	929	84.3	1,252	67.4	1,901	23.9	1,850	30.8
Grok-4.20 (reasoning)	2,526	90.5	1,393	92.8	2,707	86.8	3,794	96.1	5,951	91.2
Grok-4.20-0309 (non-reasoning)	1,235	66.4	935	83.9	1,258	65.9	1,900	23.5	1,862	30.3
Grok-4.20-0309 (reasoning)	2,501	91.1	1,384	93.3	2,705	87.3	3,767	96.1	5,744	93.2
Grok-4.3	898	89.7	629	92.9	850	86.1	1,810	95.2	1,443	84.8

S11 Appendix of Tables

S11.1 More Results on Reasoning Benchmarks

Table S51: **Main results on four in-domain reasoning benchmarks. #Tk.** is the average number of generated tokens per response (lower is better). **Acc.** is averaged accuracy in % (higher is better).

Model	Average		AIME		MATH		AMC		Olympid	
	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.
Closed-API Reasoning Models										
Claude-4.6-Opus	1,130	70.0	1,852	30.0	689	93.2	1,375	65.1	1,395	55.1
Claude-4.5-Haiku	1,339	72.0	1,803	23.3	1,061	89.8	1,533	66.3	1,500	61.6
Gemini-3.1-Flash-Lite	778	80.4	1,036	36.7	614	94.6	893	81.9	875	71.6
Open Source Reasoning Models										
GLM-744B-40B	1,742	31.1	2,048	0.0	1,487	52.0	1,900	16.9	1,898	18.8
DeepSeek-1.6T-49B	1,305	58.8	1,884	23.3	920	82.0	1,482	49.4	1,543	44.3
DeepSeek-284B-13B	1,308	57.3	1,892	16.7	906	79.2	1,510	50.6	1,554	43.7

S11.2 Per-task Results on Unlimited Token Budget

Table S52: **Token budget gradient results at 32K.**

Model	Average		AIME		MATH		AMC		OlympiadBench	
	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.
Qwen-3.6-Plus	15,207	80.0	25,812	56.7	8812	96.2	18,520	90.4	19,065	67.7
Gemini-3-Flash	4844	91.4	8332	93.3	2830	98.6	5049	96.4	6156	85.3
Qwen3-Next-80B-3B	8472	88.0	16,183	83.3	4185	97.8	9193	92.8	11,216	80.4
GLM-Air-106B-12B	11,278	86.1	20,859	83.3	6517	97.8	13,257	92.8	14,136	76.7
Step-Flash-196B-11B	10,147	80.2	23,018	56.7	4124	97.0	10,868	88.0	13,949	67.9
Hunyuan3-295B-21B	9552	81.5	22,038	66.7	4238	97.4	10,685	86.7	12,794	69.8

Table S53: **Token budget gradient results at 16K.**

Model	Average		AIME		MATH		AMC		OlympiadBench	
	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.
Qwen-3.6-Plus	10,678	55.9	15,472	10.0	7432	80.4	13,440	42.2	12,530	41.5
Gemini-3-Flash	4802	90.4	8332	93.3	2830	98.6	4989	95.2	6083	83.6
Qwen3-Next-80B-3B	7045	77.3	12,568	56.7	3988	95.2	8334	83.1	8905	64.3
GLM-Air-106B-12B	8831	71.7	14,191	40.0	5970	90.4	10,837	71.1	10,465	59.3
Step-Flash-196B-11B	6692	71.7	13,740	33.3	3338	92.0	7752	72.3	8734	58.2
Hunyuan3-295B-21B	6771	70.7	13,509	30.0	3728	92.0	8343	71.1	8532	56.7

S11.3 Per-task Results on Out-of-domain Generalization

Table S54: **Token budget gradient results at 8K.**

Model	Average		AIME		MATH		AMC		OlympiadBench	
	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.
Qwen-3.6-Plus	6492	38.5	7919	0.0	5196	64.8	7500	19.3	7265	23.1
Gemini-3-Flash	4106	73.8	6515	53.3	2673	92.4	4286	78.3	5039	60.3
Qwen3-Next-80B-3B	4991	60.2	7722	16.7	3326	83.0	6111	56.6	5965	45.8
GLM-Air-106B-12B	6033	49.6	7988	10.0	4711	72.8	7038	33.7	6801	36.1
Step-Flash-196B-11B	4245	63.2	7484	16.7	2527	85.2	5022	57.8	5279	49.6
Hunyuan3-295B-21B	4406	60.6	7465	20.0	2877	83.4	5494	50.6	5269	46.7

Table S55: **Token budget gradient results at 4K.**

Model	Average		AIME		MATH		AMC		OlympiadBench	
	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.
Qwen-3.6-Plus	3727	21.0	3959	0.0	3361	43.2	3986	7.2	3957	7.3
Gemini-3-Flash	2860	58.6	3885	20.0	2217	82.2	3038	54.2	3269	43.4
Qwen3-Next-80B-3B	3101	42.8	4096	0.0	2391	69.2	3590	25.3	3522	27.3
GLM-Air-106B-12B	3606	28.3	4096	0.0	3179	49.0	3875	16.9	3867	15.7
Step-Flash-196B-11B	2619	53.4	3962	10.0	1784	76.2	3004	45.8	3131	39.4
Hunyuan3-295B-21B	2698	46.3	3988	10.0	1994	70.8	3219	30.1	3098	31.7

Table S56: **Token budget gradient results at 1K.**

Model	Average		AIME		MATH		AMC		OlympiadBench	
	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.
Qwen-3.6-Plus	1016	0.1	990	0.0	1021	0.2	1024	0.0	1012	0.0
Gemini-3-Flash	1008	6.9	1024	0.0	986	16.4	1022	1.2	1023	0.9
Qwen3-Next-80B-3B	1007	7.6	1024	0.0	987	16.0	1021	3.6	1020	2.2
GLM-Air-106B-12B	1020	2.4	1024	0.0	1015	5.8	1024	0.0	1024	0.3
Step-Flash-196B-11B	912	25.8	1024	0.0	794	48.6	982	13.3	985	11.6
Hunyuan3-295B-21B	905	22.4	1024	0.0	832	43.4	962	12.0	946	9.2

Table S57: **Per-task results on out-of-domain generalization.** Models are evaluated on four general-domain tasks (MMLU, MMLU-Pro, LSAT, GPQA) under a 2,048-token rollout cap. **#Tk.** is the average number of generated tokens per response (lower is better). **Acc.** is accuracy in % (higher is better).

Model	Average		MMLU		MMLU-Pro		LSAT		GPQA	
	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.	#Tk.	Acc.
Closed-API Reasoning Models										
Grok-4.1-Fast	2892	88.2	1026	90.9	2288	84.1	2797	96.1	5459	81.8
Claude-4.5-Haiku	1090	72.0	706	88.6	883	79.8	1521	54.8	1250	64.7
Gemini-3.1-Pro	1294	68.6	681	92.8	1055	80.3	1727	54.4	1714	47.0
Claude-4.6-Sonnet	1290	57.1	645	88.8	965	73.0	1797	35.7	1754	30.8
Gemini-3-Flash	1651	29.1	1123	64.9	1593	33.9	1915	12.6	1973	5.1
Open-Source Reasoning Models										
DeepSeek-671B-37B	1556	43.6	918	82.1	1442	54.5	1938	19.6	1926	18.2
GLM-358B-32B	1575	39.8	962	75.6	1460	50.3	1936	16.1	1942	17.2
Qwen3-Next-80B-3B	1843	24.5	1509	62.3	1802	32.8	2031	0.9	2032	2.0
GPT-117B-5B	980	15.3	289	21.6	597	17.7	1671	8.6	1363	13.2
Ours										
Qwen-4B [82]	4401	59.6	1775	80.3	3466	63.6	5907	61.7	6457	32.8
- w/ KL	935	60.6	446	77.9	721	64.6	1574	50.4	1000	49.5
- w/ RS	1179	57.7	664	78.1	933	63.5	1796	43.9	1323	45.5

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: The claims are stated in Sections 1, 2, and 3, and the main experimental support is reported in Section 4.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: Limitations and future-work directions are discussed in Section S7, with additional compression-limit failure modes in Section S6.

   Guidelines:

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

3. **Theory assumptions and proofs**

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

Answer: [N/A]

Justification: The paper is empirical and method-oriented: it defines the diagnostic and objectives in Sections 2 and 3, but does not introduce formal theoretical theorems requiring proofs.

Guidelines:

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

4. **Experimental result reproducibility**

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

Answer: [Yes]

Justification: The method, pseudocode, data, training recipe, and evaluation protocol are described in Sections 3, S3, S4.1, S4.2, and S8.1.

Guidelines:

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

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

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

   Answer: [No]

   Justification: The training data and pseudocode are documented in Section S8.1.

   Guidelines:

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

6. **Experimental setting/details**

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

   Answer: [Yes]

   Justification: Training, evaluation, prompts, benchmark composition, and baseline settings are specified in Sections S4.1, S4.2, and S4.3.

   Guidelines:

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

7. **Experiment statistical significance**

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

   Answer: [Yes]

   Justification: Run-to-run variability is reported in Section S8.2 and Table S44.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

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

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: Hardware and training-time requirements are reported in Section S8.3 and Table S45; TokenProbe overhead is reported in Table S32.

   Guidelines:

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

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: The work uses public datasets, public model cards, and provider APIs, with responsible-use considerations discussed in Section S8.4.

   Guidelines:

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

10. **Broader impacts**

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

    Answer: [Yes]

    Justification: Positive and negative impacts are discussed in Section S8.4.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.

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

11. **Safeguards**

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

    Answer: [N/A]

    Justification: The paper does not release a new high-risk foundation model or scraped dataset; responsible-use context is discussed in Section S8.4.

    Guidelines:

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

12. **Licenses for existing assets**

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

    Answer: [Yes]

    Justification: Existing model, dataset, and software licenses are summarized in Section S8.5 and Table S46.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

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

    Answer: [N/A]

    Justification: No new dataset, model, or code asset is released with the submission; planned post-acceptance code release is described in Section S8.1.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: The experiments use existing benchmarks and model/API evaluations described in Section S4.2; no crowdsourcing or human-subject study is involved.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: The work does not involve crowdsourcing or human-subject research; the evaluation data and benchmarks are described in Section S4.2.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

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

    Answer: [N/A]

    Justification: LLM usage is declared in Section S8.6.

    Guidelines:

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