Cut Your Losses! Learning to Prune Paths Early for Efficient Parallel Reasoning

Jiaxi Bi[1,3*†] **Tongxu Luo**[1,2*] **Wenyu Du**[4] **Zhengyang Tang**[1] **Benyou Wang**[1,2‡]
[1]The Chinese University of Hong Kong, Shenzhen
[2]Shenzhen Loop Area Institute
[3]USTB [4]DualityRL
jiaxibi@xs.ustb.edu.cn tongxuluo@cuhk.edu.cn wangbenyou@cuhk.edu.cn

Abstract

Parallel reasoning enhances Large Reasoning Models (LRMs) but incurs prohibitive costs due to futile paths caused by early errors. To mitigate this, path pruning at the prefix level is essential, yet existing research remains fragmented without a standardized framework. In this work, we propose the first systematic taxonomy of path pruning, categorizing methods by their signal *source* (internal vs. external) and *learnability* (learnable vs. non-learnable). This classification reveals the unexplored potential of *learnable internal methods*, motivating our proposal of **STOP** (**S**uper **TO**ken for **P**runing). Extensive evaluations across LRMs ranging from 1.5B to 20B parameters demonstrate that STOP achieves superior effectiveness and efficiency compared to existing baselines. Furthermore, we rigorously validate the scalability of STOP under varying compute budgets—for instance, boosting GPT-OSS-20B accuracy on AIME25 from 84% to nearly 90% under fixed compute budgets. Finally, we distill our findings into formalized empirical guidelines to facilitate optimal real-world deployment. Code, data and models are available at https://bijiaxihh.github.io/STOP.

1 Introduction

Parallel reasoning has established itself as a standard paradigm for solving complex problems (OpenAI, 2024; Wang et al., 2025b). The core principle is to sample multiple independent reasoning paths and subsequently aggregate them to derive a robust consensus. However, this accuracy gain comes at a prohibitive cost. Generating dozens or even hundreds of trajectories per query increases computational overhead by orders of magnitude (Jin et al., 2025) and escalates inference costs to nearly $6 per query (NVIDIA Corporation, 2025).

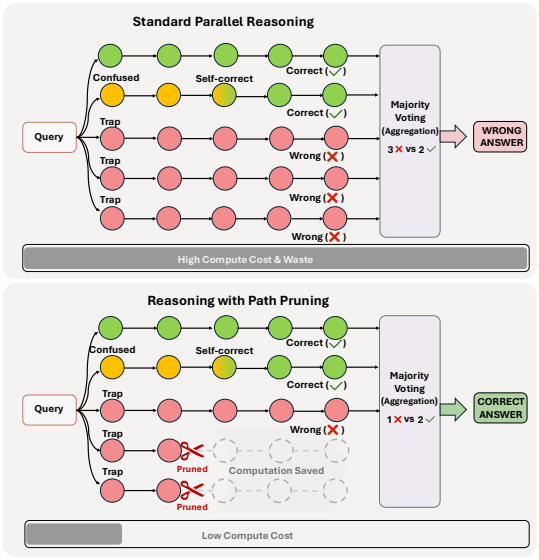


Figure 1: The necessity of pruning early. Early errors often lead to irreversible failure. Pruning these futile paths early not only saves computation but also purifies the candidate set for better consensus.

Why Prune Early in Parallel Reasoning? Crucially, recent studies (Luo et al., 2025; Hassid et al., 2025) reveal that this extensive computation is largely squandered: **not every path contributes to the solution**. Many trajectories are flawed from inception, yet they consume equal resources to generate and subsequently pollute the final answer aggregation. As illustrated in Figure 1, once a reasoning path begins with a flawed prefix, the LRM struggles to self-correct, inevitably spiraling into a futile trajectory (Luo et al., 2025). Consequently, identifying and terminating these unpromising paths at the *prefix level*—a technique known as **path pruning (or prefix rejection)**—is essential.

A Unified Taxonomy While existing methods attempt to filter paths using auxiliary reward models (Liao et al., 2025), internal confidence (Fu et al., 2025), or semantic redundancy (Hong et al., 2025), they lack a standardized evaluation protocol, leading to fragmented research. So first, we propose the

* Equal contribution; alphabetical by last name.
† Work done during interning at CUHK-Shenzhen.
‡ Corresponding author.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 19171–19189
July 2-7, 2026

first systematic taxonomy of path pruning, classifying methods based on the *source* (internal vs. external) and *learnability* (learnable vs. non-learnable) of their signals (see Figure 2). This taxonomy reveals a significant research gap: the unexplored potential of *learnable internal methods*. Conceptually, learnable internal methods offer unique advantages, as learning enables task-specific accuracy gains, while internal signals provide early, fine-grained indicators of reasoning failure without incurring extra computational overhead. To bridge this gap, we introduce **STOP** (**S**uper **TO**ken for **P**runing), the first efficient instantiation of this paradigm. Extensive evaluations demonstrate that STOP outperforms existing baselines in both effectiveness and efficiency.

Further Evaluation and Empirical Analysis Despite the promise of path pruning, its widespread adoption is currently hindered by unverified scalability across varying computational budgets and model sizes; and the absence of empirical guidelines for determining optimal pruning configurations in real-world scenarios. To overcome them, we rigorously validate the utility of path pruning in practical settings. We conduct extensive experiments across diverse model sizes (1.5B to 20B) and compute budgets, confirming that STOP exhibits robust scalability. Moreover, we distill our empirical analysis into actionable guidelines, providing a formalized method to determine the optimal retention ratio for varying resource constraints.

Contributions In summary, this work makes four primary contributions: (1) We present the first systematic investigation and taxonomy of path pruning. (2) We propose STOP, a novel pruning method based on learnable internal signals. (3) We provide a comprehensive evaluation demonstrating STOP's superior scalability and effectiveness. (4) We establish empirical guidelines to support the practical implementation of path pruning.

2 A Unified Taxonomy of Path Pruning

2.1 Problem Definition

Consider a LRM Θ and an input query x, parallel reasoning improves accuracy by generating N independent trajectories $T = \{\tau_i\}_{i=1}^{N}$, where $\tau_i \sim P_\Theta(x)$, and aggregating them through a consensus strategy, such as majority voting. The final prediction $\hat{y}$ is typically computed as:

$$\hat{y} = \text{vote}(\{\tau_i\}_{i=1}^{N}). \quad (1)$$

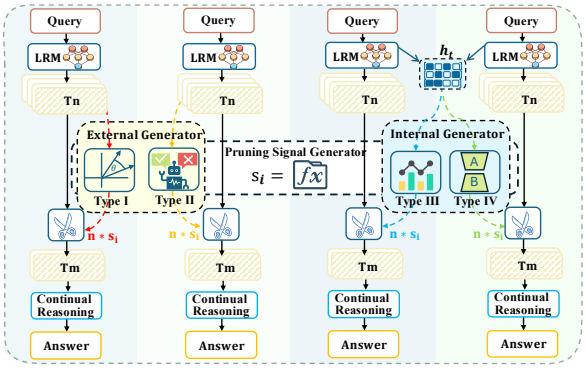


Figure 2: The proposed taxonomy of path pruning.

However, generating N complete trajectories incurs a linear computational cost ($C \propto N$). To mitigate this cost, path pruning aims to identify and discard unpromising trajectories early in the decoding process.

The Path Pruning Formulation Formally, we define a checkpoint at length L_{prefix} where the generation is paused. At this stage, the model has produced a set of prefixes $\mathcal{P} = \{p_i\}_{i=1}^{N}$. The core of path pruning is a **pruning signal generator** S, which maps each prefix to a scalar score representing its potential correctness:

$$s_i = S(p_i \mid x, \Theta), \quad (2)$$

where $s_i \in [0, 1]$ denotes the pruning signal. Based on these signals, we retain only the top-k promising paths (where $k \ll N$) for full completion, discarding the rest. The final aggregated answer is then derived exclusively from this pruned subset:

$$\hat{y}_{\text{pruned}} = \text{vote}(\{\text{finish}(p_i) \mid s_i \in \{s_j\}_{j=1}^{k}\}). \quad (3)$$

So, the objective of path pruning is to design an S that maximizes $\hat{y}_{\text{pruned}}$'s accuracy while minimizing the computational cost (the number of generated tokens). Therefore, the design of S dictates the effectiveness of the entire framework.

2.2 A Unified Taxonomy of Pruning Signal Generators

As defined in Section 2.1, the efficacy of path pruning hinges entirely on the quality of the pruning signal generator S. While the function of S is consistent—scoring prefixes—existing methods differ fundamentally in *how* this signal is produced. To systematically evaluate these approaches, we categorize them based on two critical dimensions: the *source* of the signal (External vs. Internal) and the *learnability* of the generator (Learnable vs. Non-learnable), as summarized in Table 1.

Table 1: A Unified Taxonomy of Path Pruning Methods. We categorize methods based on the pruning signal source and learnability. **Type IV** satisfies both **Desideratum 1** (Internal) and **Desideratum 2** (Learnable).

	Non-Learnable	**Learnable** (Desideratum 2)
External Source	**Type I** **SlimSC** (Hong et al., 2025)	**Type II** **DeepPrune** (Tu et al., 2025), **LaBoR** (Liao et al., 2025) **ThinkPRM** (Khalifa et al., 2025), **MAV** (Lifshitz et al., 2025)
Internal Source (Desideratum 1)	**Type III** **DeepConf** (Fu et al., 2025), **AdaDec** (He et al., 2025) **Think Just Enough** (Sharma and Chopra, 2025)	**Type IV** **STOP (Ours)**

Two Desiderata for Signal Generators Before categorizing specific methods, we establish two desiderata for an ideal signal generator:

Desideratum 1. ***Internal Source*** *An ideal S should leverage the rich, high-dimensional internal states of the LRM.*

Internal signals contain fine-grained information about uncertainty and reasoning dynamics that are often lost in the final text output used by external methods.

Desideratum 2. ***Learnability*** *An ideal S should be trainable to adapt to specific data distributions.*

Learnable parameters allow the generator to capture complex, non-linear patterns of error that rigid, pre-defined heuristics cannot model.

Based on these axes, we classify existing works into four distinct types.

External Signal Source Methods in this category derive pruning signals from the generated textual output or by querying separate models. They fail to satisfy Desideratum 1.

Type I. ***Surface Heuristics*** *These methods rely on human-designed rules (e.g. similarity) applied to the surface form of the generated text.*

While computationally cheap, these heuristics are rigid and blind to the model's actual confidence. To overcome these, the next type introduces learnability into the external evaluation process.

Type II. ***External Judges*** *These approaches employ a separate, trained model to evaluate the reasoning path.*

Although they satisfy Desideratum 2, they incur significant computational overhead due to the need for additional model inference and fail to access the LRM's internal certainty. To overcome this rigidity, the next category introduces learnability into the external evaluation process.

Internal Signal Source Methods in this category extract signals directly from the LRM's internal states, accessing to richer information (satisfying Desideratum 1).

Type III. ***Raw Confidence*** *This paradigm utilizes intrinsic metrics directly derived from the decoding process, such as perplexity or token probability.*

However, these methods rely on fixed definitions of confidence, violating Desideratum 2; raw probability does not always correlate with reasoning correctness.

Type IV. ***Learned Intuition*** *The final category represents the intersection of both desiderata: a trainable module inserted into the LRM to process internal states.*

This approach can leverage rich hidden representations (Internal) while adapting to the specific error patterns of the task (Learnable).

3 Methodology: Super Token for Pruning

As established in our taxonomy, Type IV represents the ideal pruning paradigm but remains unexplored. In this section, we introduce **STOP** (**S**uper **TO**ken for **P**runing), the first efficient instantiation of this paradigm. We delineate the motivation in Section 3.1, followed by the architectural design and workflow in Section 3.2.

3.1 Motivation for Type IV Pruning

As illustrated in Figure 2, prior methods compromise on either information richness or adaptability. Type II suffers from high latency, while Type III lacks the capacity to model complex error patterns. Type IV represents an ideal optimum: it combines the *efficiency* of accessing internal states with the *adaptability* of learnable parameters. However, this type remains unexplored due to the challenge of designing a module that extracts these signals without disrupting the LRM's generative capabilities.

3.2 Instantiation of Type IV Pruning: STOP

To instantiate this type, we design **STOP** as a lightweight, non-invasive module that integrates seamlessly with the backbone LRM.

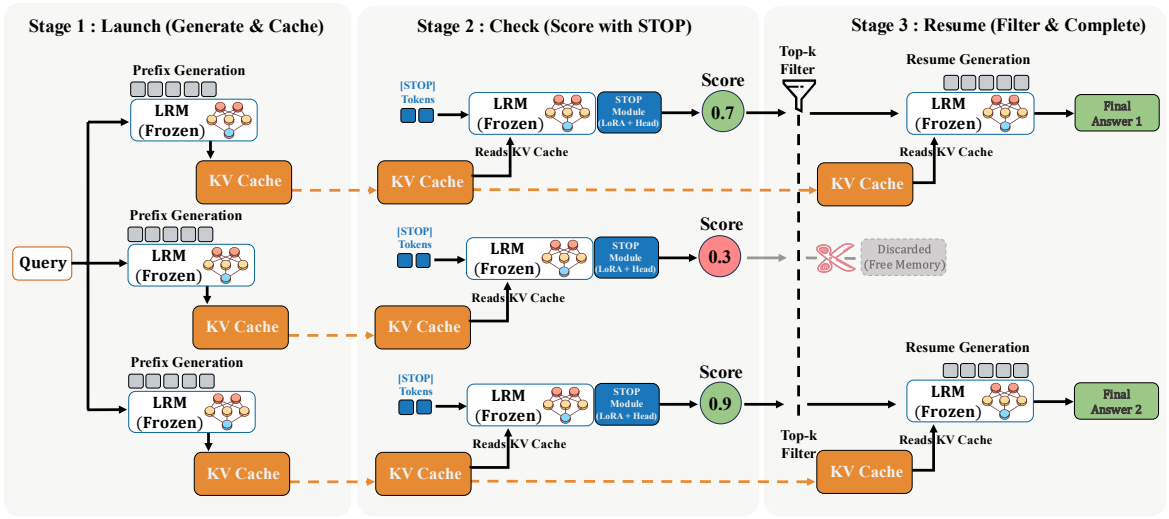


Figure 3: The inference process comprises three stages: caching initial prefixes (**Launch**), scoring them via the **STOP** module (**Check**), and completing only the top-ranked candidates (**Resume**).

Components We augment the fixed LRM Θ with three learnable components: (1) **A Super Token (`[STOP]`)** added to the vocabulary, acting as a specialized query vector to aggregate information; (2) **A Critique Adapter** LoRA (θ_{LoRA}), activated only when processing the `[STOP]` token to extract error-specific features without altering the LRM's general reasoning capabilities; (3) **A Classification Head** (W_{cls}), which projects the hidden state of the `[STOP]` token to a scalar probability.

This design ensures **modularity**: the original parameters Θ remain frozen, preserving the LRM's generative capability while enabling efficient parameter-efficient fine-tuning (PEFT).

Training: Learn to Use Internal Information The goal of training is simple: teach the model to distinguish promising prefixes from futile ones. Formally, for a prefix p_i, we derive a soft label $s_i^{mc} \in [0, 1]$ via Monte Carlo estimation (details in Appendix B). The training process involves two steps: First, we compute the KV cache of the prefix using the frozen LRM: $\mathcal{C}_{p_i} = \text{LRM}(p_i; \Theta)$. Second, we append a sequence of learnable `[STOP]` tokens, denoted as T_s, and process them using the LoRA-augmented model. The final hidden state h_i is fed into the classifier to minimize the soft binary cross-entropy loss:

$$\begin{aligned}\mathcal{L} = \quad & -[s_i^{mc} \log \sigma(W_{cls} h_i) \\ & +(1 - s_i^{mc}) \log(1 - \sigma(W_{cls} h_i))],\end{aligned} \tag{4}$$

where $h_i = \text{LRM}(T_s \mid \mathcal{C}_{p_i}; \Theta, \theta_{\text{LoRA}})_{-1}$.

Training Cost Constructing the MC supervision requires sampling multiple continuations per prefix to estimate s_i^{mc} (e.g., $K = 32$), which introduces an upfront computational cost during data construction. However, this cost is incurred only once, and the resulting STOP module is lightweight and reusable across tasks. To facilitate transparency and reproducibility, we provide detailed cost statistics in Appendix B.3 and will release the constructed dataset and trained checkpoints, allowing practitioners to bypass this step entirely. Importantly, this one-time cost is amortized during deployment, where STOP improves efficiency by pruning unpromising paths early.

Inference: "Launch-Check-Resume" To efficiently prune paths without slowing down generation, we design a three-stage pipeline (Figure 3):

Stage 1: Launch Instead of generating the full trajectories immediately, we first generate N short prefixes (e.g., first 1024 tokens) for the query. Crucially, we cache the internal states (KV Cache) of these prefixes.

Stage 2: Check We append the `[STOP]` tokens to the cached prefixes. The trained module reads the KV cache and outputs a quality score for each prefix. *Note:* This step is extremely fast because it processes only a few tokens (the `[STOP]` sequence) and reuses the heavy computation already done in Stage 1.

Stage 3: Resume We rank the prefixes by their scores and apply a **Top-k Filter**. Futile paths are discarded immediately to free up memory. Only the top-k most promising prefixes are resumed and generated to completion to obtain the final answers.

4 A Close Look at Path Pruning through the Lens of Signal Generators

4.1 On the Effectiveness of Pruning

To systematically evaluate the effectiveness of four types of pruning signal generators in our taxonomy,

Table 2: Results of avg@k (avg@m|k) across various models and benchmarks. The best result in each row is **bolded** and the second best is underlined.

Model	Dataset	No pruning (Baseline)		Type I		Type II		Type III		Type IV	
		avg@64 (↑)	Tokens (↓)	avg@8|64 (↑)	Tokens (% ↓)	avg@8|64 (↑)	Tokens (% ↓)	avg@8|64 (↑)	Tokens (% ↓)	avg@8|64 (↑)	Tokens (% ↓)
DS-Qwen-2.5-1.5B	AIME24	30.10	782.3k	26.25	218.3k (-72.09%)	32.50	325.9k (-58.34%)	32.92	210.6k (-73.08%)	**37.92**	**204.3k** (-73.88%)
	AIME25	22.76	784.8k	24.17	214.7k (-72.64%)	24.17	325.0k (-58.59%)	23.75	208.7k (-73.40%)	**26.67**	**206.6k** (-73.68%)
	BRUMO25	30.99	774.6k	29.58	212.8k (-72.53%)	31.67	325.6k (-57.96%)	31.67	209.7k (-72.93%)	**33.75**	**204.4k** (-73.61%)
	HMMT25	15.05	856.4k	15.83	224.2k (-73.82%)	15.00	337.2k (-60.63%)	17.08	220.9k (-74.21%)	**17.92**	**215.5k** (-74.84%)
	GPQA-D	33.08	550.9k	32.32	187.1k (-66.03%)	-	-	34.98	180.4k (-67.25%)	**48.42**	**179.4k** (-67.43%)
DS-Qwen-2.5-7B	AIME24	54.69	666.2k	53.75	202.7k (-69.58%)	54.58	312.5k (-53.09%)	55.00	197.4k (-70.38%)	**61.67**	**189.0k** (-71.63%)
	AIME25	39.67	703.0k	35.42	207.4k (-70.50%)	39.17	317.6k (-54.82%)	41.67	202.6k (-71.18%)	**42.50**	**197.5k** (-71.91%)
	BRUMO25	50.99	656.6k	50.00	199.3k (-69.64%)	51.25	312.1k (-52.46%)	52.92	194.5k (-70.38%)	**56.67**	**190.2k** (-71.03%)
	HMMT25	23.91	808.9k	25.00	219.5k (-72.87%)	23.33	330.8k (-59.11%)	24.58	216.1k (-73.28%)	**27.08**	**211.6k** (-73.84%)
	GPQA-D	45.95	443.8k	47.09	173.6k (-60.89%)	-	-	46.02	166.7k (-62.43%)	**55.75**	**165.9k** (-62.61%)
DS-Qwen-3-8B	AIME24	76.93	1361k	77.50	290.4k (-78.67%)	78.75	398.4k (-70.73%)	78.33	284.3k (-79.11%)	**79.17**	**279.0k** (-79.51%)
	AIME25	70.68	1427k	69.17	297.2k (-79.18%)	72.50	408.4k (-71.39%)	70.42	291.2k (-79.60%)	**72.92**	**290.9k** (-79.62%)
	BRUMO25	75.00	1320k	76.25	284.8k (-78.43%)	75.83	394.9k (-70.10%)	74.58	280.2k (-78.78%)	**78.75**	**277.5k** (-78.98%)
	HMMT25	51.04	1601k	50.00	322.1k (-79.88%)	50.83	427.8k (-73.28%)	51.25	314.0k (-80.38%)	**54.58**	**311.7k** (-80.53%)
	GPQA-D	56.87	652.6k	57.07	201.0k (-69.20%)	-	-	58.78	196.6k (-69.86%)	**63.32**	**193.5k** (-70.35%)
GPT-OSS-20B	AIME24	75.26	594.2k	73.33	196.6k (-66.91%)	76.25	299.8k (-49.55%)	75.00	187.0k (-68.54%)	**77.50**	**184.4k** (-68.98%)
	AIME25	70.99	673.4k	69.17	205.1k (-69.54%)	69.17	311.7k (-53.71%)	70.83	197.7k (-70.64%)	**75.42**	**191.1k** (-71.62%)
	BRUMO25	68.02	575.6k	69.17	187.6k (-67.41%)	66.25	298.8k (-48.09%)	67.92	**179.0k** (-68.90%)	**70.00**	183.6k (-68.11%)
	HMMT25	48.13	910.8k	45.83	236.3k (-74.06%)	45.42	336.9k (-63.01%)	46.25	228.0k (-74.97%)	**52.92**	**216.1k** (-76.27%)
	GPQA-D	65.55	277.2k	66.41	151.8k (-45.24%)	-	-	68.43	145.9k (-47.36%)	**77.46**	143.4k (-48.26%)

we conduct extensive experiments on five reasoning benchmarks. We employ a diverse suite of LRMs ranging from 1.5B to 20B parameters, specifically the DeepSeek-R1-Distill-Qwen series (Guo et al., 2025) and gpt-oss-20b (OpenAI, 2025).

Standardized protocol. To ensure a fair comparison, we establish a standardized evaluation protocol: for each query, we generate 64 initial reasoning paths. We prune these to the top 8 candidates. For each S, we apply pruning at **2,048 tokens** to rigorously evaluate their ability to identify futile paths with limited context.

Evaluation metrics. We report two metrics: **(1) avg@k**, defined as the average accuracy over the k paths. In the context of pruning, we denote this metric as **avg@m|k** (selecting m from k). Since random pruning theoretically yields an average accuracy equivalent to the no-pruning baseline, **a pruning method is considered effective only if its avg@m|k surpasses the baseline avg@k**, thereby indicating a higher density of correct answers in the selected subset. **(2) total tokens**, which is used to quantify computational cost. We calculate the relative token reduction Δ as:

$$\Delta = \frac{\text{Tokens}_{\text{original}} - \text{Tokens}_{\text{pruned}}}{\text{Tokens}_{\text{original}}} \times 100\%. \quad (5)$$

We list the detailed experimental settings, including infrastructure and hyperparameters in Appendix C.

Performance Hierarchy across Four Types Pruning As presented in Table 2, while most pruning signals demonstrate effectiveness, we observe distinct performance hierarchies. First, internal-based generators (Type III and Type IV) consistently outperform external-based ones (Type I and Type II). This advantage stems from their access to internal LRM states—such as hidden states and KV caches—which encode significantly richer representations than the constrained natural language outputs used by external methods. Second, learnable generators (Type IV and Type II) surpass non-learnable baselines, as both leverage training data to detect reasoning errors at early stages; we further validate this by explicitly training Type II on our data (see Appendix D). Most remarkably, **Type IV (STOP) dominates all other paradigms** in both effectiveness and efficiency. For instance, on the AIME 24 benchmark (1.5B), STOP increases average accuracy from 30.10% to **37.92%**—significantly exceeding Type II (32.50%) and Type III (32.92%)—while simultaneously reducing total token consumption by over **73%**.

Findings 1. *Type IV pruning offers better efficiency-accuracy trade-off.*

4.2 On the Scalability of Pruning

After validating the effectiveness, we now put these S into practical parallel inference settings to assess their scalability. We show the cons@N vs. total compute (tokens) in Figure 4. We fix the retention ratio at $\gamma = M/N = 1/2$ for all methods and vary the initial sample size N to cover different compute budgets. All other configurations remain consistent with Section 4.1.

Robustness across Tasks and Model Scales We observe a key phenomenon: across all tasks and model scales, some pruning signals achieve better performance than the no-pruning baseline. However, most existing methods do not exhibit consistent improvements across different tasks and models. For example, Type III outperforms the baseline on AIME 2024 with the 1.5B model but falls below it on AIME 2025. In contrast, our proposed Type IV demonstrates stable and consistently superior scalability across nearly all tasks. We attribute this robustness to the fact that Type IV captures the

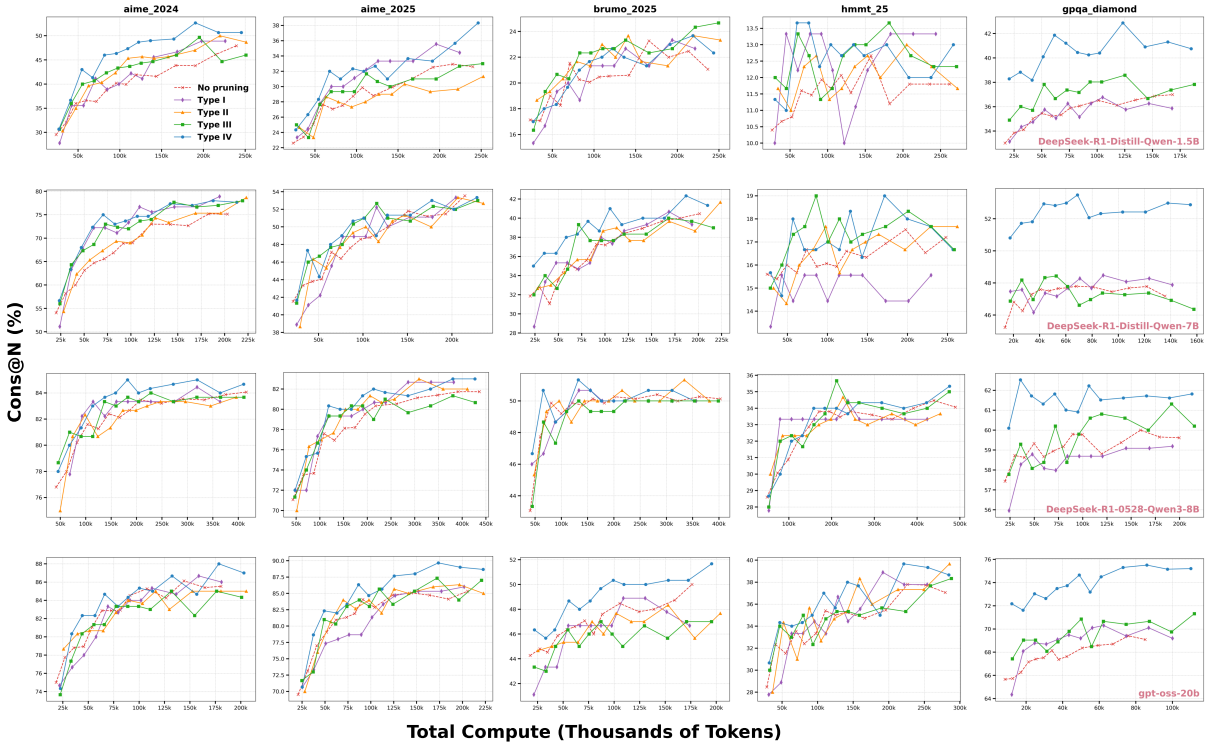


Figure 4: Performance vs. compute for four types of S on math and stem benchmarks.

intrinsic logical consistency of reasoning paths, which we further analyze in Section 5.3.

Findings 2. *Type IV pruning scales robustly across varying compute budgets.*

5 A Closer Look at STOP

5.1 Determining the Optimal remaining ratios

While the effectiveness of Type IV is established, optimal deployment requires precise tuning of two critical hyperparameters: the prefix length (L_{prefix}) and the retention ratio (γ). Since increasing L_{prefix} generally enhances error detection at the cost of higher latency, users typically fix this parameter according to their specific latency budget. However, determining the optimal retention ratio γ remains non-trivial. To provide a practical guideline, we formalize the objective as finding a function $\gamma = f(C, L_{\text{prefix}}, L_{\text{task}})$ that maximizes accuracy given a compute budget C (in tokens) and a reference task length L_{task}:

$$\arg\max_{f} \text{Accuracy}(C, L_{\text{prefix}}, L_{\text{task}}, \gamma), \tag{6}$$

where γ determines the proportion of paths retained. Identifying this function f enables the prediction of the optimal γ for any given configuration.

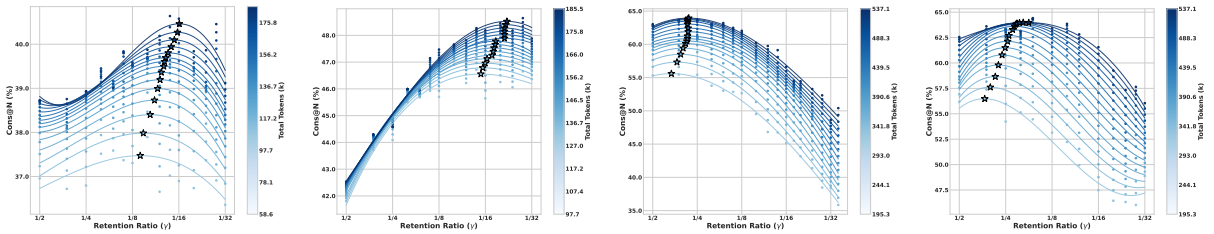
Consistent Empirical Trends across Various Settings To derive f, we conduct experiments using DS-Qwen-2.5-1.5B on AIME 2024 and GPQA Diamond, sweeping γ from $1/32$ to $1/2$ across four distinct L_{prefix} settings. The results, plotted in Figure 5, exhibit consistent trends: the optimal γ decreases as either the compute budget C or the prefix length L_{prefix} increases. These observations indicate that with sufficient compute or richer context, the model identifies futile paths more reliably, thereby allowing for more aggressive pruning (lower γ) without compromising accuracy.

Formalizing Empirical Findings Building on these insights, we model the relationship using a power-law formulation:

$$\gamma^{-1} = f(C, L_{\text{prefix}}, L_{\text{task}}) = aC^b \frac{L_{\text{prefix}}^c}{L_{\text{task}}^d}. \tag{7}$$

In this formulation, all input variables are normalized to units of 1,024 tokens. Fitting this model to our empirical data yields empirical coefficients $a \approx 1.17 \times 10^4$, $b \approx 0.46$, $c \approx 0.40$, and $d \approx 4.55$. As illustrated in Figure 6, the predicted curve aligns closely with the empirical optimal points, offering a robust guideline for parameter selection in practical deployments.

Applying the Empirical Guideline To facilitate practical deployment, we apply the derived guideline to predict the optimal retention ratio γ for specific configurations without exhaustive search. Specifically, for a task with a shorter response horizon ($L_{\text{task}} \approx 8{,}650$), a prefix length of $L_{\text{prefix}} = 2{,}048$, and a total compute budget of $C = 158$k tokens, the scaling law predicts an optimal inverse retention ratio of $\gamma^{-1} \approx 9.63$, corresponding to $\gamma \approx 10\%$. Conversely, for a task with a



(a) GPQA ($L_{\text{prefix}} = 512$) (b) GPQA ($L_{\text{prefix}} = 1024$) (c) AIME ($L_{\text{prefix}} = 2048$) (d) AIME ($L_{\text{prefix}} = 4096$)

Figure 5: Performance comparison under different retention ratios (γ) and prefix lengths (L_{prefix}).

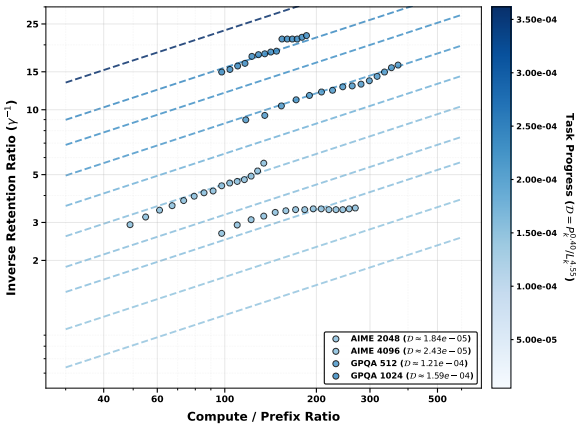


Figure 6: Inverse retention ratio γ^{-1} vs. compute-to-prefix ratio. The theoretical curves (Eq. 7) closely align with empirical observations across varying reasoning progress levels.

longer reasoning chain ($L_{\text{task}} \approx 12\text{k}$, $L_{\text{prefix}} = 3\text{k}$, and $C = 275\text{k}$), it yields a more conservative estimate of $\gamma^{-1} \approx 3.36$.

These predictions are consistent with our empirical observations, indicating that the scaling law naturally adapts to variations in task complexity. For detailed lookup guidelines across a broader range of configurations, we refer readers to **Appendix E.2**.

5.2 Ablations and Analysis

To validate the core design choices of **STOP**, we examine two critical dimensions: the quality of the supervision signal and the computational overhead during inference.

Ablation: Quality of the Supervision Signal **STOP** uses Monte Carlo (MC) estimation with $K = 32$ samples to generate probabilistic soft labels (s^{mc}), and we compare this setting with binary hard-label supervision, which corresponds to a single-sample estimate ($K = 1$). While hard labels are computationally cheap, they introduce high variance because prefix quality depends on a single stochastic continuation. As shown in Table 3, increasing the sampling budget from $K = 1$ to $K = 32$ consistently improves performance. On AIME 2024, soft supervision improves Cons@N from 46.67% to 53.33%. These results indicate that MC-based soft labels provide a low-variance signal that enables the lightweight **STOP** module to learn stable pruning boundaries.

Table 3: Performance comparison between hard labels ($K = 1$) and MC-estimated soft labels ($K = 32$).

Dataset	Supervision Type	avg@8|64 (%)	Cons@N (%)
AIME 24	Hard Labels ($K = 1$)	35.42	46.67
	Soft Labels ($K = 32$)	**36.67**	**53.33**
GPQA	Hard Labels ($K = 1$)	40.78	47.98
	Soft Labels ($K = 32$)	**41.73**	**48.48**

Findings 3. *When training pruning method, soft labels (0.0 to 1.0) have lower variance than hard labels (0 or 1).*

Ablation: Necessity of Critique Adapter Given that the LRM's internal states already encode rich reasoning history, a natural question arises: Is a simple linear classifier sufficient to decode the pruning signal? As shown in Table 4, the answer is negative. Removing the LoRA adapter leads to a significant performance drop (e.g., from **36.67%** to **31.67%** on AIME 2024). This phenomenon highlights a fundamental misalignment: the LRM's native representations are optimized for predicting next token, not value discrimination. A linear head alone struggles to extract quality assessments from this generation-centric feature space.

Table 4: Comparing the STOP module with a simple linear classifier confirms that raw internal states require adaptation to perform effective self-evaluation.

Dataset	Configuration	avg@8|64 (%)	Cons@N (%)
AIME 24	STOP w/o Adapter	31.67	46.67
	STOP	**36.67**	**53.33**
GPQA	STOP w/o Adapter	33.96	35.35
	STOP	**41.73**	**48.48**

Findings 4. *High-quality self-correction cannot be achieved by merely probing the states in LRMs; it requires a specialized transformation to bridge the gap between thinking forward (generation) and looking back (reflection).*

Ablation: Sensitivity to Design Choices We further examine the sensitivity of **STOP** to key design choices, namely the number of `[STOP]` tokens

and the LoRA rank. As shown in Table 5, performance improves with more tokens, peaks at 4–6, and then degrades with further increases, indicating a trade-off between expressive capacity and overfitting. Similarly, Table 6 shows that moderate ranks (e.g., $r = 128$) achieve the best performance, while larger ranks lead to slight degradation, suggesting that excessive capacity is unnecessary.

Findings 5. *STOP is robust to reasonable hyperparameter choices and does not require large adapters to perform effectively.*

Table 5: Effect of the number of `[STOP]` tokens (DS-Qwen-2.5-1.5B, AIME 2024, $L_{\text{prefix}} = 2048$).

# Tokens	avg@32|256	# Tokens	avg@32|256
1	30.10	6	**37.71**
2	33.54	7	36.15
3	35.94	8	35.00
4	36.86	9	33.65
5	36.77	-	-

Table 6: Effect of LoRA rank (DS-Qwen-2.5-1.5B, AIME 2024).

Rank	Params (M)	avg@8|64
32	36.9	32.50
64	73.9	36.25
128	**147.7**	**36.67**
256	295.4	35.83

Analysis: Computational Overhead We quantify the inference latency on a single NVIDIA H100 GPU using DS-Qwen-2.5-7B with a fixed prefix length of $2,048$. As detailed in Table 7, existing paradigms incur notable costs: Type II requires full sequence re-encoding, resulting in the highest latency (**1.13 s**, 3.37% overhead), while Type I suffers from the computational bottleneck of pairwise similarity calculations (**0.38 s**). In stark contrast, **STOP** (Type IV) minimizes overhead to a negligible **0.20 s** (**0.59%**). This efficiency stems directly from our architectural design: by **reusing the precomputed KV cache** and restricting verification to a single forward pass of special tokens, STOP eliminates redundant computation, ensuring high-throughput deployment.

Table 7: Inference overhead analysis. STOP achieves near-zero cost by avoiding re-encoding.

Pruning Paradigm	Latency / Check	Relative Overhead
Type II	1.13 s	3.37%
Type I	0.38 s	0.93%
Type IV (STOP)	**0.20 s**	**0.59%**

Analysis: Generalization to Non-Math/STEM Tasks To assess whether **STOP** captures universal reasoning patterns beyond mathematics and science, we extend our evaluation to **ZebraLogic**, a benchmark designed to evaluate combinatorial reasoning and constraint satisfaction capabilities through logic grid puzzles. Specifically, we conduct experiments on the multiple-choice mode (`mc_mode`) to test reasoning under constraints. Using the **DS-Qwen-2.5-7B** model, we evaluate 500 randomly sampled instances of moderate difficulty (Rows, Cols ≤ 4). As shown in Table 8, **STOP** improves accuracy from 73.73% to **77.23%**. This consistent gain confirms that the pruning signals learned by the module are not strictly domain-dependent, but rather transferable to general logical inference tasks.

Table 8: **Generalization on ZebraLogic. STOP robustly generalizes** beyond math and science tasks.

Model	No pruning (Baseline)	STOP	Gain
	avg@64 (%)	avg@8|64 (%)	
DS-Qwen-2.5-7B	73.73	**77.23**	**+3.50%**

Analysis: Generalization to Tool Use We further evaluate whether **STOP** generalizes to realistic tool-use scenarios by submitting our system to the **AIMO3** competition, where models solve mathematical problems with access to external tools under a fixed evaluation protocol. Built on a **GPT-OSS-120B + tool** framework, we compare against a baseline that directly performs parallel reasoning without pruning under the same resource constraints; due to the competition setting (single H100 GPU and a 5-hour limit for 50 problems), the baseline cannot scale to larger sampling budgets. As shown in Table 9, both STOP configurations consistently outperform the baseline, improving the score from **39** to **42** (24→8) and **43** (16→8), with the best configuration reaching **silver-level performance** on the public leaderboard, demonstrating that **STOP** remains effective in tool-augmented reasoning and translates into tangible gains in real-world competitive settings.

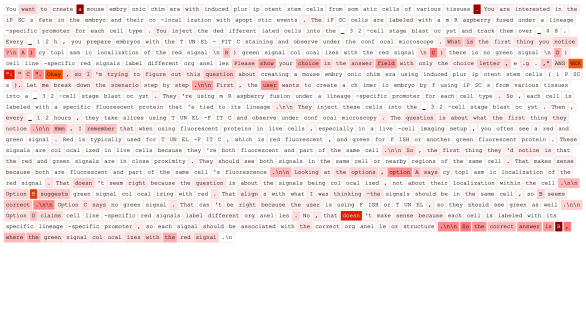
Table 9: Results on the AIMO3 competition setting with tool use (GPT-OSS-120B).

Method	Score
Baseline + Tool	39
STOP (24→8)	42
STOP (16→8)	**43**

5.3 How STOP Attends

To understand how **STOP** distinguishes valid reasoning trajectories, we visualize the attention distri-

(a) **High-scoring Path**



(b) **Low-scoring Path**

Figure 7: **Attention Analysis of [STOP] Decision-Making.** High-scoring paths prioritize logical pivots (e.g., self-correction markers), whereas low-scoring paths fixate on terminal answer tokens. This contrast confirms that **STOP** functions as a process-oriented evaluator, rewarding reasoning integrity over premature closure.

bution of the [STOP] token (Figure 7). Overall, the module exhibits a broad attention pattern. It consistently attends to **multiple-choice options** (A, B, C, D) as well as discourse markers (e.g., "Hmm", "Wait"), which enables it to track the structural progression of the reasoning process.

Process-oriented Evaluation Importantly, high-scoring and low-scoring trajectories present clearly distinct attention signatures. In the **high-score case** (Figure 7a), attention prioritizes the reasoning process rather than the final outcome. Specifically, the [STOP] token focuses on **cognitive pivots** (e.g., the negation "don't"), indicating an emphasis on logical operations that trigger self-correction. In contrast, the **low-score case** (Figure 7b) demonstrates a pattern of **premature closure**: attention shifts early to the terminal token (e.g., "B") while critical logical markers receive little attention. Consequently, **STOP** penalizes such trajectories and interprets the lack of attention to logical pivots as evidence of reasoning failure. See Appendix G for more cases.

6 Conclusion

In this work, we address the critical efficiency bottleneck of parallel reasoning by establishing the first unified taxonomy of path pruning. This framework not only resolves the fragmentation in existing research but also reveals the unexplored potential of *learnable internal methods* (Type IV). To bridge this gap, we introduce **STOP**, a lightweight method that leverages internal representations to identify and terminate futile prefixes effectively. Extensive evaluations demonstrate that STOP consistently dominates existing paradigms, significantly enhancing reasoning accuracy while reducing token consumption by over 70%. Moreover, we resolve scalability and deployment uncertainties by deriving a robust interaction formulation. This provides practitioners with a precise empirical guideline for optimizing the trade-off between exploration and exploitation under varying computational constraints. Finally, our in-depth analysis of the mechanism and architectural choices offers valuable insights to guide future research.

Acknowledgment

This work was supported by Major Frontier Exploration Program (Grant No. C10120250085) from the Shenzhen Medical Academy of Research and Translation (SMART), Shenzhen Medical Research Fund (B2503005), NSFC grant 72495131, the 1+1+1 CUHK-CUHK(SZ)-GDSTC Joint Collaboration Fund, Guangdong Provincial Key Laboratory of Mathematical Foundations for Artificial Intelligence (2023B1212010001), and the International Science and Technology Cooperation Center, Ministry of Science and Technology of China (under grant 2024YFE0203000).

Limitations

As the pioneering instantiation of the internal learnable paradigm (Type IV), **STOP** validates the potential of intrinsic representations for trajectory pruning. However, we acknowledge specific limitations in our current scope and highlight promising directions for future research.

Limitations.

- **Verification at Extreme Scales** Our current evaluation spans models up to 20B parameters and standard compute budgets (e.g., $N = 64$). The behavior of STOP on substantially larger models (e.g., 70B+) and under massive sampling regimes (e.g., $N \geq 1000$) remains to be empirically verified.

- **Structural Flexibility** This work focuses on single-stage pruning at fixed positions (e.g.,

$L_{\text{prefix}} = 2048$). We have not yet explored more complex settings, such as multi-stage sequential pruning or unstructured pruning where checkpoints are determined dynamically rather than at fixed token indices.

Future Directions.

- **Progressive Multi-Stage Pruning** A natural extension is to apply STOP in a cascading manner (e.g., funneling candidates from $64 \rightarrow 32 \rightarrow 16$ at successive checkpoints). This "progressive filtering" strategy could further optimize the compute allocation by dynamically narrowing the search space as reasoning deepens.

- **Accelerating RL Training** Beyond inference, STOP holds significant potential for training efficiency. In Reinforcement Learning (e.g., PPO or GRPO), STOP can serve as an online rejection mechanism during the rollout phase, terminating low-value trajectories early to increase the density of high-quality training signals per unit of compute.

References

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, et al. 2024. Large language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Han Cai, Jing Li, Wei Liu, and Tianqi Chen. 2024. Medusa: Simple framework for accelerating llm generation with multiple decoding heads. *arXiv preprint arXiv:2401.10774*.

Brendan Chan, Chen Liang, Yiming Yang, and Tian Wang. 2023. Chameleon: Plug-and-play compositional reasoning with large language models. *arXiv preprint arXiv:2304.09842*.

Yichao Fu, Xuewei Wang, Yuandong Tian, and Jiawei Zhao. 2025. Deep think with confidence. *arXiv preprint arXiv:2508.15260*.

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

Michael Hassid, Gabriel Synnaeve, Yossi Adi, and Roy Schwartz. 2025. Don't overthink it. preferring shorter thinking chains for improved llm reasoning. *arXiv preprint arXiv:2505.17813*.

Kaifeng He, Mingwei Liu, Chong Wang, Zike Li, Yanlin Wang, Xin Peng, and Zibin Zheng. 2025. Adadec: Uncertainty-guided adaptive decoding for llm-based code generation. *arXiv preprint arXiv:2506.08980*.

Colin Hong, Xu Guo, Anand Chaanan Singh, Esha Choukse, and Dmitrii Ustiugov. 2025. Slim-sc: Thought pruning for efficient scaling with self-consistency. *arXiv preprint arXiv:2509.13990*.

Yunho Jin, Gu-Yeon Wei, and David Brooks. 2025. The energy cost of reasoning: Analyzing energy usage in llms with test-time compute. *arXiv preprint arXiv:2505.14733*.

Muhammad Khalifa, Rishabh Agarwal, Lajanugen Logeswaran, Jaekyeom Kim, Hao Peng, Moontae Lee, Honglak Lee, and Lu Wang. 2025. Process reward models that think. *arXiv preprint arXiv:2504.16828*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th ACM Symposium on Operating Systems Principles*.

Baohao Liao, Xinyi Chen, Sara Rajaee, Yuhui Xu, Christian Herold, Anders Søgaard, Maarten de Rijke, and Christof Monz. 2025. Lost at the beginning of reasoning. *arXiv preprint arXiv:2506.22058*.

Shalev Lifshitz, Sheila A. McIlraith, and Yilun Du. 2025. Multi-agent verification: Scaling test-time compute with multiple verifiers. *arXiv preprint arXiv:2502.20379*.

Tongxu Luo, Wenyu Du, Jiaxi Bi, Stephen Chung, Zhengyang Tang, Hao Yang, Min Zhang, and Benyou Wang. 2025. Learning from peers in reasoning models. *arXiv preprint arXiv:2505.07787*.

Mathematical Association of America. 2024. American invitational mathematics examination (aime) 2024. https://maa.org/math-competitions/american-invitational-mathematics-examination-aime. Accessed: February 2024.

Mathematical Association of America. 2025. American invitational mathematics examination (aime) 2025. https://maa.org/math-competitions/american-invitational-mathematics-examination-aime. Accessed: February 2025.

NVIDIA Corporation. 2025. Llm inference benchmarking: How much does your llm inference cost? https://developer.nvidia.com/blog/llm-inference-benchmarking-how-much-does-your-llm-infer Accessed: 2025-11-05.

OpenAI. 2024. Learning to reason with LLMs. Accessed: 2025-11-01.

OpenAI. 2025. gpt-oss model card (gpt-oss-120b & gpt-oss-20b). Accessed: 2025-11-01.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2024. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling (COLM)*.

Aman Sharma and Paras Chopra. 2025. Think just enough: Sequence-level entropy as a confidence signal for llm reasoning. *arXiv preprint arXiv:2510.08146*.

Shangqing Tu, Yaxuan Li, Yushi Bai, Lei Hou, and Juanzi Li. 2025. Deepprune: Parallel scaling without inter-trace redundancy. *arXiv preprint arXiv:2510.08483*.

Peiyi Wang, Lifan Li, Zhenyu Shao, Ruixuan Xu, Dong Dai, Yanzhe Li, Yuzhuo Yao, and Zhifang Sui. 2024. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9426–9439, Bangkok, Thailand. Association for Computational Linguistics.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, and Sharan Narang. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Yifan Wang, Yichi Zhang, Xinyi Li, and Jie Zhou. 2025a. A survey on parallel reasoning. *arXiv preprint arXiv:2510.12164*.

Ziqi Wang, Boye Niu, Zipeng Gao, Zhi Zheng, Tong Xu, Linghui Meng, Zhongli Li, Jing Liu, Yilong Chen, Chen Zhu, Hua Wu, Haifeng Wang, and Enhong Chen. 2025b. A survey on parallel reasoning.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pages 11809–11822.

Jian Zhao, Rui Liu, Kai Zhang, Zihan Zhou, Jun Gao, Dong Li, and Bowen Zhou. 2025. Genprm: Scaling test-time compute of process reward models via generative reasoning. *arXiv preprint arXiv:2504.00891*.

A Related Work

A.1 Parallel Reasoning

Parallel reasoning, which generates multiple trajectories to verify or aggregate answers, has become a standard paradigm for enhancing LRM performance. A recent survey by (Wang et al., 2025a) systematically categorizes these approaches into three dimensions: **(1) Non-interactive Reasoning**, which generates independent paths without communication, including majority voting in *Self-Consistency* (Wang et al., 2022), ranking in *Best-of-N* (Brown et al., 2024), and structured exploration in *Tree-of-Thoughts* (Yao et al., 2023). **(2) Interactive Reasoning**, which enables active information exchange among paths, for example, internal state sharing in *Leap* (Luo et al., 2025) or multi-agent collaboration (Chan et al., 2023). **(3) Efficiency Optimization**, which focuses on accelerating decoding mechanics, such as speculative decoding in *Medusa* (Cai et al., 2024). Although these methods enhance reasoning performance, they still suffer from substantial inference costs, which remain a major limitation.

A.2 Path Pruning (Prefix Rejection)

To mitigate the high inference cost of parallel reasoning, path pruning strategies aim to terminate unpromising trajectories early. Consistent with the taxonomy in Section 2.2, we categorize existing works based on signal source and learnability.

Regarding **external** signals, **non-learnable** methods (Type I) like SlimSC (Hong et al., 2025) prune paths utilizing heuristic metrics such as semantic similarity to minimize redundancy. In contrast, **learnable** approaches (Type II) rely on trained verifiers. This category encompasses discriminative classifiers used in DeepPrune (Tu et al., 2025) and LaBoR (Liao et al., 2025), as well as generative verifiers in ThinkPRM (Khalifa et al., 2025) and multi-agent frameworks like MAV (Lifshitz et al., 2025). Shifting to **internal** sources, **non-learnable** methods (Type III) derive signals directly from intrinsic statistics. Representative works include confidence-based estimation in DeepConf (Fu et al., 2025) and AdaDec (He et al., 2025), or entropy-based metrics in Think Just Enough (Sharma and Chopra, 2025).

Notably, **prior works** leave the quadrant of **internal learnable** modules (Type IV) unexplored. **STOP** is designed to bridge this gap, utilizing a trainable adapter to extract rich internal semantics,

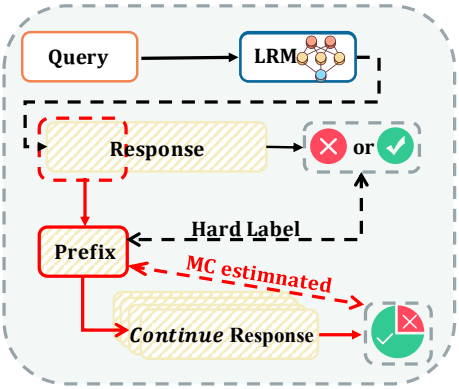


Figure 8: MC-based construction of prefix–potential supervision.

thus offering a solution that is both structurally efficient and data-driven.

B Data Construction Details

To train our **STOP** module, we require a dataset that directly maps prefixes of reasoning paths to the probability that the final answer succeeds. A single binary label on a complete path provides an insufficient and noisy signal, because a promising prefix may still end in an accidental failure, while a flawed prefix may occasionally be recovered by chance. Therefore, we construct a dataset of `(prefix, success probability)` pairs using **Monte Carlo (MC) estimation** (Wang et al., 2024; Zhao et al., 2025).

B.1 Source Benchmarks and Decontamination

We constructed a supervised fine-tuning dataset derived from high-quality mathematical and scientific benchmarks. Specifically, we aggregated approximately 1,000 problems from the **AIME** competition (spanning years 1984 to 2023) (Mathematical Association of America, 2024, 2025), augmented with the non-Diamond portion of the **GPQA** dataset (Rein et al., 2024). *Crucially, to ensure zero data leakage, we strictly excluded the evaluation sets from this training corpus: specifically, **AIME 2024**, **AIME 2025**, and the **GPQA Diamond** subset were entirely removed.*

B.2 Model-Specific Construction Pipeline

Since reasoning capabilities vary across model scales, we adopted a **model-specific pipeline** where each LRM (e.g., 1.5B) generates its own training data. The procedure proceeds as follows:

Table 10: **Statistics of model-specific training data.** Prefixes are extracted from Math (AIME) and Science (GPQA). Data volume decreases for larger models due to filtering of trivial samples.

Model	Math	Science	Total
DS-Qwen-2.5-1.5B	14,816	8,448	**23,264**
DS-Qwen-2.5-7B	12,092	5,666	**17,758**
DS-Qwen-3-8B	10,848	4,456	**15,304**
GPT-OSS-20B	7,872	2,378	**10,250**

Difficulty Stratification (Filtering). Before generating prefixes, we first filter source problems to focus on the model's *learnable boundary*. For each problem, we generate $N = 32$ reasoning paths and calculate the pass rate. We explicitly exclude **trivial samples** (> 28 correct answers) that the model has already mastered, as well as **intractable samples** (< 4 correct answers) likely beyond its current capacity. This ensures that the training data consists of problems where the pruning signal is most valuable.

Prefix Generation. From the retained problems, we use the LRM to generate a prefix p that forms part of a complete reasoning trajectory. To simulate a realistic mid-generation checkpoint, we truncate these paths at a fixed length of $L_{\textbf{prefix}} = 2{,}048$ tokens.

Potential Estimation via MC Rollouts. To estimate the potential of p, we fix the prefix and generate $K = 32$ continuations under a temperature of 0.6. This procedure produces a set of full-length responses $\{\tau'_1, \tau'_2, \ldots, \tau'_K\}$.

MC Score Calculation. We evaluate each response for correctness (1 if correct and 0 otherwise). The MC-estimated success probability s^{mc} is defined as the empirical accuracy:

$$s^{mc} = \frac{1}{K} \sum_{j=1}^{K} \text{is\_correct}(\tau'_j). \tag{8}$$

The resulting label $s^{mc} \in [0.0, 1.0]$ provides a fine-grained probabilistic target used to train the STOP module.

Data Statistics and Insights. Table 10 summarizes the composition of the constructed datasets. We observe a distinct **inverse scaling trend**: as the model size increases, the number of valid training samples decreases (e.g., from 23,264 for 1.5B to 10,250 for 20B). This confirms the efficacy of our difficulty stratification strategy: larger models (e.g., GPT-OSS-20B) achieve high pass rates

Table 11: **Training Cost for MC Supervision Construction.** We report the number of training pairs and the estimated wall-clock cost (in 8×H100 GPU hours) required to construct the dataset with $K = 32$ Monte Carlo samples per prefix.

Model	Math	Science	Total Training Pairs	8×H100 Hours
DS-Qwen-2.5-1.5B	14,816	8,448	**23,264**	43.08
DS-Qwen-2.5-7B	12,092	5,666	**17,758**	39.46
DS-Qwen-3-8B	10,844	4,456	**15,304**	37.79
GPT-OSS-20B	7,872	2,378	**10,250**	75.93

Table 12: **Training hyperparameters across model scales.**

Hyperparameter	1.5B	7B	8B	20B
Per-Device Batch Size	16	8	8	2
Gradient Accumulation	1	2	2	8
Learning Rate	2×10^{-5}	2×10^{-5}	2×10^{-5}	2×10^{-5}
LoRA Rank (r)	128	256	256	2048
LoRA Alpha (α)	256	512	512	4096
Target Modules	All Linear	All Linear	All Linear	All Linear
Optimizer	AdamW	AdamW	AdamW	AdamW
Max Prefix Length	2048	2048	2048	2048
Training Epochs	15	15	15	15
Precision	`bf16`	`bf16`	`bf16`	`bf16`

($> 28/32$) on a larger portion of the source benchmarks, causing these "trivial" instances to be filtered out. Consequently, the training data naturally adapts to focus on the *learnable boundary* specific to each model's capability.

B.3 Training Cost Details

Constructing the MC supervision dataset requires sampling multiple continuations per prefix (e.g., $K = 32$) as described in Section 3.1. In practice, we find that moderate sampling budgets provide a good balance between estimation stability and computational cost, as also reflected in our ablation results. We report the estimated cost across different model scales in Table 11.

These costs correspond to a one-time data construction process. Once constructed, the dataset can be reused across training runs and model variants, amortizing the cost of data construction. The trained STOP module introduces negligible overhead during inference. These costs are reported to provide transparency and should be interpreted as approximate estimates depending on implementation and hardware configurations.

C Detailed Experimental Settings

In this appendix, we provide the complete experimental details to ensure reproducibility, covering infrastructure, datasets, input formats, training hyperparameters, and baseline implementations.

C.1 Infrastructure and Sampling Configuration

Infrastructure. All experiments were conducted on NVIDIA H100 (80GB) GPUs. We utilized the `vLLM` framework (Kwon et al., 2023) to support efficient batched inference during the evaluation phases.

Sampling Configuration. To ensure consistency across all pruning methods, we adopted a unified generation configuration. Specifically, the temperature was set to 0.6, top-p to 0.95, and top-k to 40. The maximum generation length was set to 16,384 tokens for the 1.5B and 7B models, and 32,768 tokens for the 8B and 20B models. For `gpt-oss` models, the reasoning effort was set to "medium".

C.2 Evaluation Protocol

We strictly adhered to established evaluation protocols to ensure fair comparison and reproducibility. The **GPQA-Diamond** subset, consisting of 198 high-difficulty questions, was reserved exclusively as a held-out test set. Consequently, all remaining GPQA questions were used solely during the training stage. This rigorous separation guarantees zero information leakage from the training corpus to the evaluation benchmarks.

C.3 Prompt Templates and Input Format

To ensure rigorous reproducibility, we detail the exact prompt templates and input construction used in our experiments. We utilized the standard zero-shot Chain-of-Thought (CoT) format.

Prompt Templates & Input Format

GPQA:

```
Please show your choice in the answer field with only the choice letter, e.g., "ANSWER": "C".
```

Math Tasks (AIME, HMMT, BRUMO):

```
Please reason step by step, and put your final answer within \boxed{}.
```

STOP Module Input Mechanism:
To achieve zero-overhead verification, we do not re-encode the full text. Instead, the **STOP** token is appended directly to the **pre-computed KV cache** of the generated prefix. Conceptually, this provides the module with the following effective context, allowing it to attend to the full history:

```
[User Prompt] [Generated Reasoning Prefix] [STOP]
```

C.4 STOP Module Training Details

We developed a custom training pipeline utilizing the Hugging Face `Accelerate` and `PEFT` libraries. All experiments were conducted on 8 NVIDIA H100 GPUs using a LoRA-only approach. We froze the base model parameters and strictly trained low-rank adapters attached to **all linear layers** within the transformer blocks. Specifically, we targeted the full set of projections: `q_proj`, `k_proj`, `v_proj`, `o_proj`, `gate_proj`, `up_proj`, and `down_proj`. The specific hyperparameters, including the varying LoRA configurations for different model scales, are detailed in Table 12.

C.5 Baseline Descriptions

We provide additional details on the baseline implementations used in Section 4:

- **SlimSC (Hong et al., 2025) (Type I):** Computes the pairwise Jaccard similarity between the current generation and previously explored reasoning paths. It prunes trajectories that exhibit high semantic redundancy to ensure diversity.

- **LaBoR (Liao et al., 2025) (Type II):** Relies on a separate, trained Process Reward Model (PRM) to score generated prefixes. We used the official checkpoints released by the authors where available.

- **DeepConf (Fu et al., 2025) (Type III):** Estimates confidence by computing perplexity and entropy directly from the model logits of the generated tokens, serving as a non-learnable internal baseline.

D Ablation: Data Quality vs. Architecture

D.1 Motivation and Setup

A potential confounding factor in our main results is the quality of the training data. Since **STOP** is trained on a high-quality dataset constructed via Monte Carlo rollouts, it is natural to hypothesize that the observed performance gains mainly arise from superior supervision rather than from the **Type IV** architecture itself. To disentangle these two factors, we introduce a controlled baseline, **Type II$^{\text{retrain}}$ (Retrained Early Pruning)**. LaBoR (Liao et al., 2025) propose an Early Pruning strategy based on an external Process Reward Model (PRM), specifically `Qwen2.5-Math-PRM-7B`, but their model is not trained on our MC-estimated soft labels. For a fair comparison, we adopt the same architecture and fine-tune it on the *same dataset* of prefix–success probability pairs used to train **STOP**. This comparison isolates the architectural effect between an internal, learnable method (**Type IV**) with access to full hidden states and an external reward model (**Type II**) that relies only on token-level outputs, thereby ruling out data quality as the sole source of improvement. **Note:** Because the backbone of Type II is specialized for mathematics, we exclude the GPQA (Science) benchmark from this ablation, as the external PRM lacks sufficient domain knowledge for scientific reasoning.

D.2 Detailed Analysis

Table 13 reports results across models and benchmarks. We observe that **Type II-retrain** consistently outperforms the standard Type II baseline, which is typically trained on public PRM datasets or heuristic labels. This result confirms that MC-estimated soft labels provide a stronger and more informative supervision signal than conventional binary labels, even for external reward models. More importantly, despite being trained on identical data, **STOP** consistently outperforms **Type II-retrain** across different model scales. For example, at the

Table 13: **Ablation Study: Architecture vs. Data.** Comparison of avg@8 and token efficiency. **Type II** refers to the standard external PRM baseline (Early Pruning). **Type II$^{\text{retrain}}$** denotes the same external architecture retrained on our MC-estimated data. **STOP** (**Type IV**) outperforms both, demonstrating that architectural access to internal states yields gains beyond data quality alone. Note: Type II variants are not evaluated on GPQA due to the domain limitation of the math-specialized PRM backbone.

Model	Dataset	Full Paths (Baseline)		Type II		Type II$^{\text{retrain}}$		Type IV	
		avg@8|64 (↑)	Tokens (↓)	avg@8|64 (↑)	Tokens (% ↓)	avg@8|64 (↑)	Tokens (% ↓)	avg@8|64 (↑)	Tokens (% ↓)
DS-Qwen-2.5-1.5B	AIME24	30.10	782.3k	32.50	325.9k (-58.34%)	37.50	318.2k (-59.33%)	**37.92**	**204.3k** (-73.88%)
	AIME25	22.76	784.8k	24.17	325.0k (-58.59%)	24.16	323.2k (-58.82%)	**26.67**	**206.6k** (-73.68%)
	BRUMO25	30.99	774.6k	31.67	325.6k (-57.96%)	32.50	320.5k (-58.62%)	**33.75**	**204.4k** (-73.61%)
	HMMT25	15.05	856.4k	15.00	337.2k (-60.63%)	16.67	333.8k (-61.03%)	**17.92**	**215.5k** (-74.84%)
	GPQA-D	33.08	550.9k	-	-	-	-	**48.42**	**179.4k** (-67.43%)
DS-Qwen-2.5-7B	AIME24	54.69	666.2k	54.58	312.5k (-53.09%)	59.17	308.6k (-53.68%)	**61.67**	**189.0k** (-71.63%)
	AIME25	39.67	703.0k	39.17	317.6k (-54.82%)	37.08	315.5k (-55.13%)	**42.50**	**197.5k** (-71.91%)
	BRUMO25	50.99	656.6k	51.25	312.1k (-52.46%)	53.33	309.1k (-52.92%)	**56.67**	**190.2k** (-71.03%)
	HMMT25	23.91	808.9k	23.33	330.8k (-59.11%)	24.17	328.8k (-59.35%)	**27.08**	**211.6k** (-73.84%)
	GPQA-D	45.95	443.8k	-	-	-	-	**55.75**	**165.9k** (-62.61%)
DS-Qwen-3-8B	AIME24	76.93	1361k	78.75	398.4k (-70.73%)	77.92	396.5k (-70.87%)	**79.17**	**279.0k** (-79.51%)
	AIME25	70.68	1427k	72.50	408.4k (-71.39%)	**73.33**	407.5k (-71.44%)	72.92	**290.9k** (-79.62%)
	BRUMO25	75.00	1320k	75.83	394.9k (-70.10%)	75.00	396.1k (-70.01%)	**78.75**	**277.5k** (-78.98%)
	HMMT25	51.04	1601k	50.83	427.8k (-73.28%)	52.08	427.7k (-73.28%)	**54.58**	**311.7k** (-80.53%)
	GPQA-D	56.87	652.6k	-	-	-	-	**63.32**	**193.5k** (-70.35%)
GPT-OSS-20B	AIME24	75.26	594.2k	76.25	299.8k (-49.55%)	74.16	302.5k (-49.09%)	**77.50**	**184.4k** (-68.98%)
	AIME25	70.99	673.4k	69.17	311.7k (-53.71%)	69.58	310.4k (-53.91%)	**75.42**	**191.1k** (-71.62%)
	BRUMO25	68.02	575.6k	66.25	298.8k (-48.09%)	67.50	297.9k (-48.24%)	**70.00**	**183.6k** (-68.11%)
	HMMT25	48.13	910.8k	45.42	336.9k (-63.01%)	48.75	333.3k (-63.41%)	**52.92**	**216.1k** (-76.27%)
	GPQA-D	65.55	277.2k	-	-	-	-	**77.46**	143.4k (-48.26%)

1.5B scale, **STOP** achieves higher avg@8 on AIME 25 (26.67% vs. 24.16%) and BRUMO 25 (33.75% vs. 32.50%), while at the 7B scale it surpasses Type II$^{\text{retrain}}$ on AIME 24 (61.67% vs. 59.17%). In addition, while Type II is restricted to mathematical tasks due to its specialized backbone, **STOP**, implemented via LoRA, naturally generalizes to the scientific domain on GPQA during training, demonstrating greater flexibility. The only exception is a minor difference on DS-Qwen-3-8B for AIME 25 (72.92% vs. 73.33%), which lies within normal variance; in all other settings, **STOP** shows clear and consistent advantages.

D.3 Discussion: The Advantage of Internal Signals

The superiority of **STOP** (**Type IV**) can be attributed to its ability to mitigate the *information bottleneck* inherent in external evaluation. An external PRM (**Type II**) judges reasoning quality solely from generated text, which is a discrete and low-dimensional projection of the model's internal reasoning process and often discards subtle signals of uncertainty and coherence. In contrast, **STOP** is integrated directly into the generator and has access to dense internal representations, including hidden states and attention patterns. These internal signals preserve rich information about confidence and logical consistency that is largely lost during decoding. By leveraging such first-person internal signals, **STOP** evaluates the potential of a prefix more accurately than a third-person external reward model.

E Derivation and Validation of the Scaling Law

In Section 5.1, we introduced the Interaction Scaling Law to describe the relationship among the optimal pruning ratio γ, the compute budget C, and task complexity. In this appendix, we first examine the empirical optimization surfaces that validate this formulation (Appendix E.1), and then provide detailed reference tables for practical deployment (Appendix E.2).

E.1 Empirical Observations on Optimal Retention

We study how the optimal retention ratio γ^*, defined as the peak of the performance envelope under a fixed compute budget, varies across benchmarks and prefix lengths L_{prefix}. Visualizations of these empirical surfaces are presented in Figure 9.

Scientific Reasoning (GPQA). For GPQA with $L_{\text{prefix}} = 512$ and 1024, the optimal strategy shifts toward more aggressive pruning as the compute budget increases. With short contexts ($L_{\text{prefix}} = 512$), γ^* is around $1/8$ at low budgets ($\sim$ 24k tokens), reflecting a balance between exploration and exploitation. As the budget increases to 195k tokens, the performance peak moves to smaller values ($\gamma \approx 1/16$), indicating that **STOP** effec-

tively discards low-quality candidates when sufficient samples are available. For medium contexts ($L_{\text{prefix}} = 1024$), conservative retention ($\gamma = 1/2$) consistently underperforms. The optimal γ^* starts near $1/8$ and rapidly decreases toward $\gamma \approx 1/28$ as compute increases.

This pruning pattern arises from the concise reasoning structure of GPQA. GPQA solutions typically require few steps, so the fixed prefix captures a large portion of the full reasoning trajectory. As a result, the prefix contains high information density and provides a strong pruning signal, enabling **STOP** to aggressively filter candidates with low risk of removing correct solutions.

Mathematical Reasoning (AIME). In contrast, AIME shows a strong dependence on prefix length, reflecting the higher sunk cost of long mathematical derivations. For $L_{\text{prefix}} = 2048$, increasing the compute budget shifts the optimal γ^* from conservative values ($\gamma \approx 1/2$) toward more aggressive pruning ($\gamma \approx 1/4$). Compared with GPQA, AIME consistently requires higher retention because mathematical reasoning is deeply sequential, and a fixed prefix represents only an initial portion of the full solution, leading to greater downstream uncertainty.

When the context length increases to $L_{\text{prefix}} = 4096$, we observe a further shift toward selectivity. Contrary to the expectation that longer contexts require conservative retention, the optimal γ^* decreases to the range $\gamma \in [1/6, 1/8]$. This behavior indicates that a longer prefix provides richer evidence for evaluating trajectory quality. With more reasoning history available, the **STOP** module identifies flawed paths with higher confidence, allowing more aggressive pruning than in the $L_{\text{prefix}} = 2048$ setting without sacrificing correct solutions.

Alignment with the Unified Formula. These results support the coupled structure of the Interaction Scaling Law. Across all tasks, γ^* consistently decreases as the compute budget C increases. At the same time, the optimal pruning level is modulated by the interaction between task domain and available context. Overall, the scaling law adapts to differences in reasoning density across domains and prefix lengths, and it aligns well with the observed empirical optimization landscapes.

E.2 Recommended Retention Guidelines

Based on the derived scaling law, we provide reference tables for selecting optimal pruning strategies. To **improve visual clarity and facilitate quick lookup**, we present the guidelines in two separate tables, each corresponding to a different compute budget regime.

These tables are intended primarily as **illustrative references** for representative task lengths. For other tasks, whether they are similar to GPQA or Math and have different response characteristics, practitioners can directly substitute the task length (L_{task}), prefix length (L_{prefix}), and compute budget (C) into the derived formula (Eq. 7) to obtain the exact optimal retention ratio.

Tables 14 and 15 report the recommended **inverse retention ratio** (γ^{-1}) for representative short-horizon tasks ($L_{task} \approx 8{,}650$) and long-horizon tasks ($L_{task} \approx 11{,}950$), respectively.

F Detailed Latency and Throughput Benchmarking

In this appendix, we present a detailed analysis of the system efficiency discussed in Section 5.2. We conduct controlled micro-benchmarks on a single NVIDIA H100 GPU using **DS-Qwen-2.5-7B**. The evaluation uses a batch size of 16 and a fixed prefix length of 2,048 tokens to simulate realistic inference conditions.

F.1 Metric Definitions

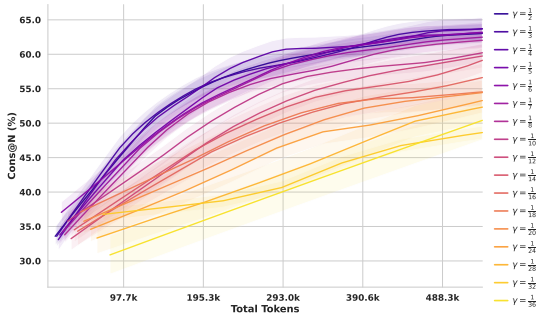
We adopt the following metrics to evaluate computational overhead:

- **Generation Time (T_{gen}):** The wall-clock time required for autoregressive decoding of reasoning tokens, excluding any verification operations.
- **Verification Latency (T_{verify}):** The explicit computation time required by the pruning signal generator to produce scores for a batch.
- **System Throughput:** The effective inference speed measured in tokens per second (tok/s). Unlike latency metrics, throughput captures implicit system-level overheads, including CPU–GPU synchronization and pipeline inefficiencies caused by context switching.

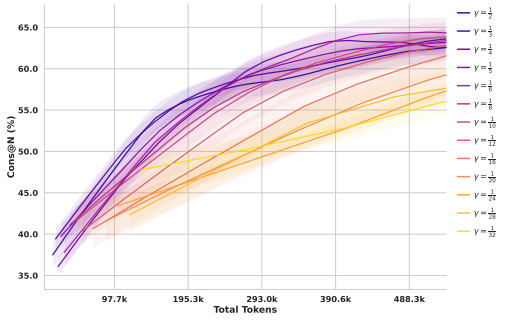
F.2 Quantitative Analysis

Table 16 reports the detailed timing breakdown across different pruning paradigms. The results reveal a clear mismatch between explicit verification latency and the realized system throughput, especially for heuristic-based methods.

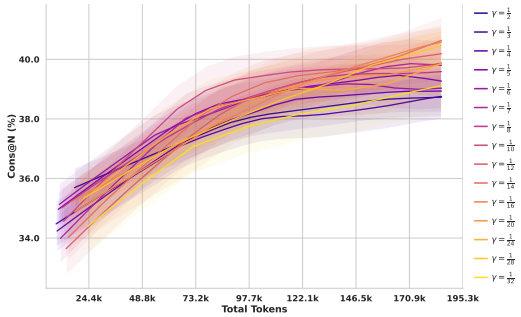
Throughput degradation in heuristic methods. A key observation is the pronounced throughput drop in Type I (SlimSC). Although the cumulative



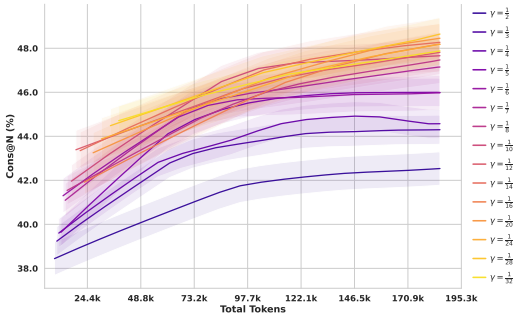
(a) **AIME 2024** ($L_{prefix} = 2048$). Optimal γ shifts to aggressive pruning as budget increases.



(b) **AIME 2024** ($L_{prefix} = 4096$). Longer context enables stable pruning at higher selectivity.



(c) **GPQA** ($L_{prefix} = 512$). Higher compute budgets drive more aggressive pruning.



(d) **GPQA** ($L_{prefix} = 1024$). Scaling behavior remains consistent with longer contexts.

Figure 9: **Empirical optimization surfaces.** Impact of retention ratio γ across increasing compute budgets.

Table 14: **GPQA (Science, Short-Horizon).** Recommended inverse retention ratios (γ^{-1}) for tasks with shorter reference lengths ($L_{\text{task}} \approx 8{,}650$). Pruning is more aggressive (higher values) even at lower budgets.

Prefix Length (L_{prefix})	**Compute Budget C (Total Tokens)**								
	140k	**160k**	**180k**	**200k**	**220k**	**240k**	**260k**	**280k**	**300k**
512	5.23	5.56	5.87	6.16	6.44	6.70	6.95	7.19	7.42
1024	6.90	7.34	7.75	8.13	8.49	8.84	9.17	9.49	9.80
1536	8.11	8.63	9.11	9.56	9.99	10.40	10.79	11.16	11.52
2048	9.10	9.68	10.22	10.73	11.21	11.67	12.10	12.52	12.93
2560	9.95	10.59	11.17	11.73	12.26	12.76	13.23	13.69	14.13

verification latency is small, the method requires frequent similarity computations during chunk-wise generation. These repeated interventions fragment GPU kernel execution, prevent sustained high utilization, and increase the base generation time from 33.20s to 40.64s.

Efficiency and implementation of STOP. In contrast, the proposed STOP module introduces a minimal verification latency of 0.20s. By reusing the resident KV cache, STOP performs verification by processing the sequence T_s in a single forward pass. During standard generation, the LoRA adapter remains disabled to strictly preserve the behavior of the base model and is activated only during the verification step. The prefix KV cache serves as a shared and immutable reference, and verification appends T_s to a temporary view of this cache to compute the score. Once scoring is complete, the temporary branch is discarded. This design removes the need for context rollbacks or cache cleanup operations, ensuring that verification introduces no structural overhead into the generation pipeline. As a result, the total wall-clock time of STOP (34.33s) remains close to that of the baseline.

Memory Footprint and Deployment Complexity. Beyond temporal latency, the spatial overhead of model deployment is a decisive factor. Methods relying on external verifiers (Type II) impose a **dual-model burden**: deploying Type II (External PRM) requires hosting a separate PRM alongside the generator. For example, using a 7B generator with a 7B reward model effectively doubles the VRAM requirement and increases orchestration complexity. In contrast, STOP is implemented as a lightweight LoRA adapter attached directly to

Table 15: **AIME (Math, Long-Horizon).** Recommended inverse retention ratios (γ^{-1}) for tasks with longer reference lengths ($L_{\text{task}} \approx 11{,}950$). Pruning is more conservative (lower values) due to higher reasoning complexity.

Prefix Length (L_{prefix})	**Compute Budget C (Total Tokens)**								
	200k	**250k**	**300k**	**350k**	**400k**	**450k**	**500k**	**550k**	**600k**
1024	1.87	2.07	2.25	2.42	2.57	2.71	2.85	2.98	3.10
2048	2.47	2.73	2.97	3.19	3.39	3.58	3.76	3.93	4.09
3072	2.90	3.21	3.49	3.75	3.99	4.21	4.42	4.62	4.81
4096	3.25	3.60	3.92	4.21	4.48	4.72	4.96	5.18	5.39
5120	3.56	3.94	4.29	4.60	4.89	5.17	5.42	5.66	5.90

Table 16: **Breakdown of Inference Latency and Throughput.** Note the discrepancy between *explicit cost* and *system impact* for heuristic methods. Although Type I (SlimSC) shows a low explicit verification cost (1.74%), the pipeline fragmentation significantly slows down generation, causing a massive **17.71% drop in throughput**. In contrast, STOP operates in-situ, keeping the throughput drop minimal ($< 3\%$) with negligible verification cost (0.59%).

Method	**Gen. Time (s)**	**Verify Latency (s)**	**Total Time (s)**	**Throughput (tok/s)**	**Throughput Drop (↓)**	**Explicit Verify Cost**
Baseline (No Pruning)	33.20	–	33.20	986.9	–	–
Type I (SlimSC)	40.64	0.38	41.02	812.1	**17.71%**	1.74%
Type II (LaBoR)	33.53	1.13	34.68	977.3	0.97%	3.37%
Type IV (**STOP**)	34.13	**0.20**	34.33	**960.1**	**2.71%**	**0.59%**

the frozen generator. This **integrated architecture** adds only a minimal number of parameters, incurring **negligible additional VRAM overhead** for model weights. It eliminates the need for managing secondary inference services, making STOP a "plug-and-play" solution for existing pipelines.

G Extended Attention Analysis

In Section 5.3, we hypothesize that the **STOP** module acts as a process-oriented evaluator. To empirically validate this, we analyze the attention patterns in Figure 10.

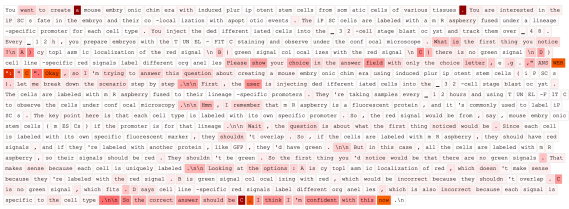
Universal Attention Pattern. Consistent with the findings in Section 5.3, **STOP** exhibits a broad attention pattern across all samples. Regardless of the score, the module consistently tracks structural discourse markers (e.g., “Wait”, “Hmm”, “Therefore”, “but”, “\n\n”) as well as the final answer text. This confirms that the module monitors the structural progression of the reasoning chain.

Distinguishing Quality via Attention Focus. However, a critical distinction determines the quality score. In **High-Scoring Trajectories** (Figures 10a and c), attention prioritizes **logical negations** (e.g., “don’t” and “doesn’t”)—which serve as cognitive pivots—over the final answer options, indicating that **STOP** values the validity of the logical derivation. Conversely, **Low-Scoring Trajectories** (Figures 10b and d) exhibit a pattern of **premature closure**: attention disproportionately fixates on the **answer options themselves** (e.g., the token “C”) while neglecting the reasoning context, serving as a robust signal for identifying guessing behavior.

(a) **High-scoring Case.** The module focuses on the logical negation “don’t” (a cognitive pivot) rather than simply jumping to the answer option.

(b) **Low-scoring Case.** Attention concentrates heavily on the answer option itself (“C”), ignoring the sparse reasoning context.

(c) **High-scoring Case.** Similar to (a), the module attends to the logical marker “doesn’t,” prioritizing the validity of the reasoning process over the final outcome.



(d) **Low-scoring Case.** The module demonstrates premature closure by fixating on the terminal choice (“C”) while bypassing critical logical intermediates.

Figure 10: **Extended Visualization of `[STOP]` Attention Maps.** While **STOP** broadly tracks structural markers (e.g., “Wait”, “Therefore”) in all cases, it distinguishes reasoning quality by focus: **High-scoring paths** (left) prioritize logical pivots (e.g., “don’t”), whereas **Low-scoring paths** (right) exhibit **premature closure** by fixating on the terminal answer options.