Mila Microsoft

The Markovian Thinker: Architecture-Agnostic Linear Scaling of Reasoning

Milad Aghajohari* [1], **Kamran Chitsaz*** [1,6], **Amirhossein Kazemnejad*** [1],
Sarath Chandar [1,5,6,7], **Alessandro Sordoni**† [1,2], **Aaron Courville**† [1,5,8], **Siva Reddy**† [1,3,4,5]
[1]Mila [2]Microsoft Research [3]McGill University [4]ServiceNow Research
[5]Canada CIFAR AI Chair [6]Chandar Research Lab [7]Polytechnique Montréal [8]Université de Montréal

*Equal Contribution †Equal Advising[1]

Abstract Reinforcement learning (RL) has recently become a strong recipe for training reasoning LLMs that produce long chains of thought (LongCoT). Yet the standard RL "thinking environment", where the state is the prompt plus all prior reasoning tokens, makes the state unbounded and forces attention-based policies to pay quadratic compute as thoughts lengthen. We revisit the environment itself. We propose *Markovian Thinking*, a paradigm in which the policy advances reasoning while conditioning on a constant-size state, decoupling thinking length from context size. As an immediate consequence this yields linear compute with constant memory. We instantiate this idea with *Delethink*, an RL environment that structures reasoning into fixed-size chunks. Within each chunk, the model thinks as usual; at the boundary, the environment resets the context and reinitializes the prompt with a short carryover. Through RL, the policy learns to write a textual state near the end of each chunk sufficient for seamless continuation of reasoning after reset. Trained in this environment, an R1-Distill 1.5B model reasons in 8K-token chunks yet thinks up to 24K tokens, matching or surpassing LongCoT-RL trained with a 24K budget. With test-time scaling, Delethink continues to improve where LongCoT plateaus. The effect of linear compute is substantial: we empirically estimate at 96K average thinking length LongCoT-RL costs 27 H100-months vs. 7 for Delethink. Analysis at RL initialization shows off-the-shelf reasoning models (1.5B–120B) often sample Markovian traces zero-shot across diverse benchmarks, providing positive samples that make RL effective at scale. Our results show that redesigning the thinking environment is a powerful lever: it enables very long reasoning without quadratic overhead and opens a path toward efficient, scalable reasoning LLMs.

Model Weights: huggingface.co/McGill-NLP/the-markovian-thinker
Code Repository: github.com/McGill-NLP/the-markovian-thinker

1 Introduction

Reinforcement learning (RL) has emerged as an effective recipe for training reasoning LLMs that produce a long chain-of-thought (LongCoT) before answering (Jaech et al., 2024; Guo et al., 2025; OpenAI, 2025); with scaling the number of "thinking tokens" improving capability (Agarwal et al., 2025). As in any RL problem, there is an environment that determines how trajectories are generated. For reasoning LLMs this environment is rather trivial that it's often omitted: the *state* is the prompt concatenated with generated reasoning tokens so far, and the *action* is the next token sampled from the policy (reasoning LLM). This seemingly innocuous choice leaves the state size unbounded, growing with longer thoughts. For attention-based policies this entails prohibitive, *quadratic growth* of compute throughout.

Most recent work tames this compute growth by constraining how much thinking occurs during RL: multi-stage training limits how often long reasoning traces are used (Luo et al., 2025b); length-regularized objectives implicitly prefer shorter correct solutions (Aggarwal & Welleck, 2025; Shen et al., 2025; Li et al., 2025; Hou et al., 2025); and pruning or early-exit methods terminate generation while preserving accuracy (Luo et al., 2025a; Dai et al., 2025; Zhao et al., 2025a). In contrast, we ask a different question: *what if the environment never creates quadratic growth to begin with?*

We propose a paradigm in which the policy advances its reasoning while conditioning on a *constant-size* state. We call such a policy a *Markovian Thinker*. The key idea is to restructure the RL formulation so the effective state that the policy reads is bounded, regardless of total thinking length. The immediate consequence is profound: longer thinking

[1]Correspondence: {aghajohm,kamran.chitsaz,amirhossein.kazemnejad}@mila.quebec

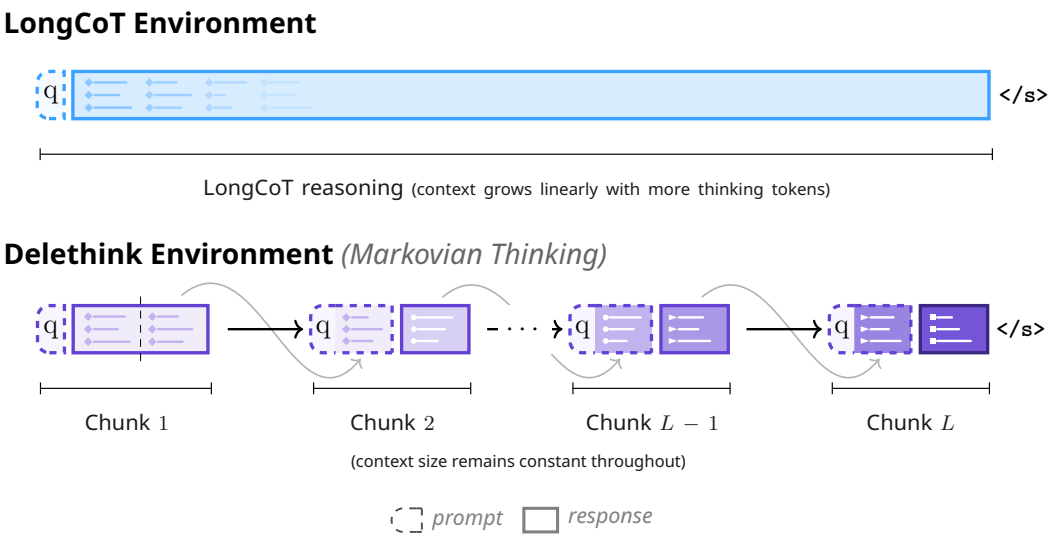


Figure 1: **Delethink** redefines the thinking RL environment as a chunked, markovian process: generation proceeds in fixed-size chunks, and at each boundary the environment resets the context to a fresh prompt containing the query plus a short carryover from the previous chunk. This forces the policy to learn to progress across chunks by maintaining a textual state, creating a *Markovian Thinker*. In contrast, the LongCoT environment concatenates tokens indefinitely, so the state (and model context) grows with the trace.

requires linear compute and constant memory with respect to thinking length, decoupling "how long the model thinks" from "how much context it must process."

We instantiate this paradigm with Delethink, an RL environment that induces Markovian behavior by organizing reasoning into a sequence of fixed-size chunks (Figure 1). Within each chunk, the model reasons as usual. At chunk boundaries, the environment *resets the context* and reinitializes the prompt using only a short carryover from the previous chunk (e.g., the last few tokens). The policy is forced to learn, through RL, to write a textual *state* near the end of each chunk so that, after reset, it can continue the thread of reasoning while conditioning only on this bounded state.

Delethink is highly effective. Despite reasoning in 8K chunks, R1-Distill 1.5B (Guo et al., 2025) trained with Delethink can think up to 24K tokens, matching and surpassing LongCoT-RL on math benchmarks with the same 24K thinking budget (Figures 2 and 4). With test-time scaling, Delethink continues to improve where LongCoT-RL plateaus, yielding additional gains (Figures 2 and 7). Pushing further, we train R1-Distill 1.5B with Delethink to think *up to* 96K tokens; with only few additional training steps, it reaches 49% on AIME'24 with solutions averaging 36K tokens. The effect of linear-compute is substantial: We empirically estimate that training for an average of 94K thinking requires 27 H100-months under LongCoT-RL versus 7 H100-months with Delethink (Figure 2).

To probe *why* Delethink trains effectively, we analyze models at RL initialization. We observe that the R1-Distill family (1.5B-14B), without additional training or prompting, already samples Markovian traces *zero-shot*, even recovering most of standard LongCoT performance (Figure 9). This strong initialization (many positive, in-distribution samples of the desired behavior) provides a favorable starting point for RL. We further study reasoning models up to 120B parameters in the Delethink environment. For example, GPT-OSS 120B (Agarwal et al., 2025) exhibits robust Markovian Thinking across PhD-level questions, coding tasks, math competitions, and crossword puzzles (Section 7). Together, these results suggest that Delethink is compatible and scales with state-of-the-art models.

The success of Markovian Thinking demonstrates that decoupling thinking length from context size can, in principle, let next-generation reasoning models think for millions of tokens. It highlights the RL environment, often treated as fixed, as a powerful lever for progress. It also suggests that non-quadratic sequence architectures (Katharopoulos et al., 2020; Yang et al., 2024b; Gu & Dao, 2023) may particularly benefit reasoning models, since thinking can be made effectively Markovian. We summarize our contributions as follows:

- We introduce the Markovian Thinking paradigm that bounds the policy's input state and yields linear compute and constant memory with thinking length (Section 4.3).

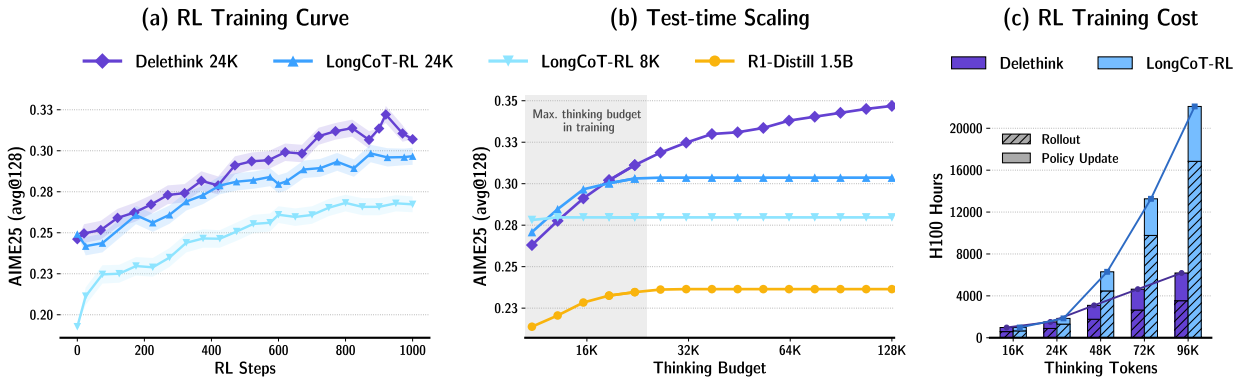


Figure 2: **(a)** Delethink 24K (a Markovian Thinker) matches and surpasses LongCoT-RL 24K in accuracy during RL training while using less compute; both methods improve as the thinking budget scales from 8K to 24K. **(b)** Beyond the trained thinking budget, Delethink significantly outperforms and keeps improving while others plateau; within the budget, Delethink and LongCoT-RL 24K scale similarly with test-time compute (reported using sequential sampling). **(c)** Training cost of R1-Distill 1.5B vs. average thinking length with an optimized stack of verl (Sheng et al., 2024) + SGLang (Zheng et al., 2024) on H100s: quadratic for LongCoT and linear for Delethink, as predicted.

- We present a simple instantiation, Delethink (Section 2), that trains off-the-shelf reasoning LLMs to be native Markovian Thinkers, empirically matching and surpassing LongCoT-RL training (Sections 6 and 6.3).
- We show that Delethink test-time scale well beyond train-time limits whereas LongCoT plateaus (Section 6.2).
- We provide empirical evidence that large SOTA reasoning LLMs has strong support for Markovian Thinking in-distribution, signaling that Delethink scales.(Section 7).

2 Related Works

Efficient RL training of LongCoT Prior works try to reduce the cost of RL training under LongCoT. Liu et al. (2025) stabilizes GRPO to avoid wasted length. Others train models to prune or exit early while maintaining accuracy (Luo et al., 2025a; Dai et al., 2025; Zhao et al., 2025a), or implicitly reward shorter thoughts among correct solutions (Aggarwal & Welleck, 2025; Shen et al., 2025; Li et al., 2025; Hou et al., 2025). Additionally, Luo et al. (2025b) limits the fraction of training with long thoughts, an approach known as multi-stage RL training although Zeng et al. (2025) shows it hurts final performance. While lighter, these approaches operate under LongCoT and therefore are quadratic.

Inference Time Efficient Thinking Prior works seek efficiency by shortening reasoning traces. Some distill traces with skipped steps or tokens (Liu et al., 2024a; Xia et al., 2025). Others control length by early exits (Ding et al., 2024), adjusting the budget to problem complexity (Han et al., 2024), or steering activations toward brevity (Zhao et al., 2025b). Structured prompting and collaboration help reduce tokens: CoThinking outlines then reasons (Fan et al., 2025). Cheng & Van Durme (2024) thinks in shorter trace of contemplative tokens. Some other works approximate the attention at test-time by changing its computation and masking tokens. Zhang et al. (2023); Yang et al. (2024a) select important tokens based on their estimated contribution to attention, while Xiao et al. (2024) preserves a small set of attention-sink tokens to stabilize quality under sliding windows. Compression approaches shrink each retained token's footprint via quantization to sub-4-bit while maintaining accuracy Hooper et al. (2024); Liu et al. (2024b). Recent approaches use exisiting parameters to learn to evict tokens in inference using distillation on the original model predictions plus a sparsity regularizer (Łańcucki et al., 2025). These methods are used to ensure faster inference time after RL training.

Making Room for Extra Thinking Prior work has tried to shorten reasoning traces at inference time. Lin et al. (2025) drops irrelevant tokens and redundant steps using an extra judge LLM. Xiao et al. (2025) prunes low-value segments by perplexity. Yan et al. (2025) iteratively summarizes current reasoning so later steps continue depending only on the summary. InftyThink's style is fixed by design, distilled from a hand-crafted dataset that locks the model into one pattern. In contrast, Delethink is an RL training approach that learns native Markovian Thinking through RL.

Algorithm 1 Delethink Step

Inputs: query $\boldsymbol{q}$; reasoning LLM π_θ; thinking context size $\mathcal{C}$; markovian state size m; Delethink iterations cap $\mathcal{I}$; group size G; reward function $\mathcal{R}$; learning rate η.
$T, R \leftarrow [\,], [\,]$ ▷ Delethink traces, rewards
Generate Delethink Traces:
for $i \leftarrow 1$ to G **do**
  $\boldsymbol{x} \leftarrow \boldsymbol{q}$ ▷ prompt
  $\boldsymbol{y} \leftarrow \pi_\theta(\boldsymbol{x}; \mathcal{C})$ ▷ generate up to $\mathcal{C}$ tokens
  $\boldsymbol{q} \leftarrow \boldsymbol{q} \oplus \boldsymbol{y}_{1:100}$ ▷ concatenate first few tokens of $\boldsymbol{y}$
  $\tau \leftarrow [(\boldsymbol{x}, \boldsymbol{y})]$ ▷ trace for group i
  for $t \leftarrow 1$ to $\mathcal{I} - 1$ **do**
    if $\mathrm{last}(\boldsymbol{y}) =$ `[EOS]` **then break**
    $\boldsymbol{x} \leftarrow \boldsymbol{q} \oplus \boldsymbol{y}_{-m:}$ ▷ keep last m thinking tokens
    $\boldsymbol{y} \leftarrow \pi_\theta(\boldsymbol{x}, \mathcal{C} - m)$ ▷ generate up to $\mathcal{C} - m$ tokens
    $\mathrm{append}(\tau, (\boldsymbol{x}, \boldsymbol{y}))$ ▷ appending chunk to trace
  $\mathrm{append}(R, \mathcal{R}(\tau))$ ▷ trace reward
  $\mathrm{append}(T, \tau)$
Estimate Advantages:
$\{\hat{A}[i]\}_{i=1}^{G} \leftarrow \mathrm{ComputeAdvantage}\big(\{R[i]\}_{i=1}^{G}\big)$ ▷ off-the-shelf advantage estimator
Updating Parameters:
$J \leftarrow \frac{1}{G} \sum_{\tau_g} \frac{1}{\ell(\tau_g)} \sum_{l=1}^{|\tau_g|} \mathcal{U}(\boldsymbol{x}_l, \boldsymbol{y}_l; \theta)$ ▷ Compute the loss according to Equation (3)
$\theta \leftarrow \theta + \eta \nabla_\theta J$

Linear Architectures Sliding or sparse attention (Beltagy et al., 2020; Zaheer et al., 2020) and kernel-based linear attention and low-rank linear attention (Katharopoulos et al., 2020; Choromanski et al., 2021; Wang et al., 2020) avoid all-pairs interactions, the quadratic component of attention, by approximations. Mamba architecture replaces self-attention with state-space models (Gu et al., 2022; Gu & Dao, 2023; Dao & Gu, 2024), achieving constant-memory, linear-time generation via recurrent state updates. Despite progress in this field, all state-of-the-art models (Agarwal et al., 2025; Gemini Team, 2024; Yang et al., 2025; Guo et al., 2025) include self-attention even the hybrid systems only interleave self-attention with alternatives. All scale asymptotically quadratic (Lieber et al., 2024; AI21 Labs, 2024). Delethink does not modify the architecture and is not quadratic in context. It decouples reasoning length and context length, avoiding the quadratic cost. However, Delethink shows continuing reasoning effectively with constant number of tokens is possible, providing evidence that these methods might be more successful in reasoning than other areas.

3 Background

We adopt the RLVR formulation, following Lambert et al. (2024); Kazemnejad et al. (2025). In this setup, the policy π_θ is a reasoning language model that, given a prompt $\boldsymbol{x} = [x_1, \ldots, x_N]$ (which may include the system prompt, chat history, and a user query), generates a response $\boldsymbol{y} = [y_1, \ldots, y_T]$ autoregressively: $\boldsymbol{y} \sim \pi_\theta(\cdot \mid \boldsymbol{x})$. The goal of RL training is to maximize expected return while regularizing the policy via a KL penalty against a reference model. Formally, the objective is

$$\mathcal{J}(\theta) = \mathbb{E}_{\boldsymbol{x} \sim \mathcal{D}, \boldsymbol{y} \sim \pi_\theta(\cdot|\boldsymbol{x})} \left[\mathcal{R}(\boldsymbol{x}, \boldsymbol{y}) \right] - \beta \, \mathrm{KL}[\pi_\theta \| \pi_{\text{ref}}], \tag{1}$$

where $\mathcal{D}$ is a dataset of queries, π_{ref} is the reference policy, β is the KL coefficient, and $\mathcal{R}(\boldsymbol{x}, \boldsymbol{y})$ is a trajectory-level reward function (verifiable rewards in our case). The policy π_θ is typically initialized from π_{ref}. Then, $\mathcal{J}(\theta)$ is optimized by sampling trajectories and applying policy-gradient updates (Shao et al., 2024).

Thinking RL environment (MDP) Equation (1) is defined over trajectories induced by an underlying language-generation Markovian Decision Process (MDP). This MDP is straightforward and typically left implicit, yet its design choice bears fundamental consequences. Concretely, token-by-token generation MDP (as used in standard LongCoT RLVR) specifies the following states $\mathcal{S}$, actions $\mathcal{A}$, and transition dynamics. At time t, the state $s_t \in \mathcal{S}$ is the concatenation of the prompt and the tokens generated so far: $s_t = \boldsymbol{x} \oplus \boldsymbol{y}_{<t} = [x_1, \ldots, x_N, y_1, \ldots, y_{t-1}]$, where $\oplus$ denotes concatenation. The action $a_t \in \mathcal{A}$ corresponds to sampling the next token y_t from the policy given the generation context. The process starts at $s_0 = \boldsymbol{x}$ (where $\boldsymbol{x}$ is the prompt), and after each action, the environment

Method	Think Tokens	FLOP	Memory	Backward Time	Generation Time
Base	n	$O(n^2)$	$O(n)$	T_B	T_G
LongCoT	nS	$O(n^2S^2)$	$O(nS)$	$O(T_BS^2)$	$O(T_GS^2)$
Markovian Thinking (Delethink)	nS	$O(n^2S)$	$O(n)$	$O(T_BS)$	$O(T_GS)$

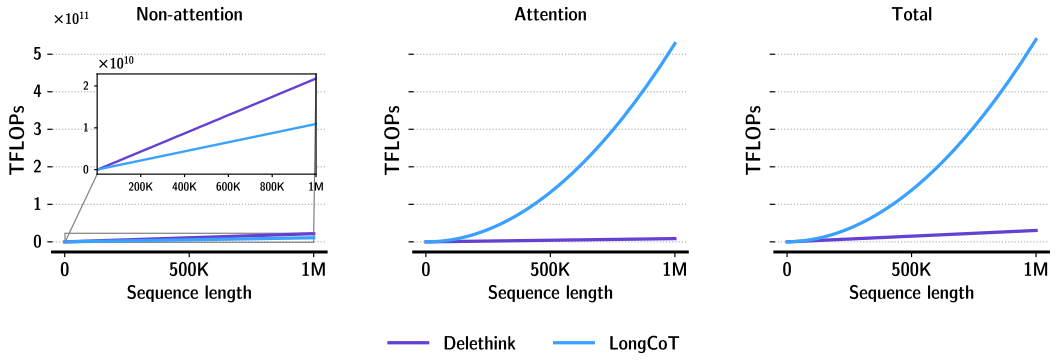


Figure 3: Computational profiles of LongCoT-RL and Delethink scaling from n to nS tokens.

deterministically transitions to the next state, s_{t+1}, by appending the action a_t to the current state s_t:

$$P(s_{t+1} = s' \mid s_t = s,\, a_t = a) \;=\; \begin{cases} 1, & \text{if } s' = s \oplus [a], \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

Policies trained in this environment learn to think so as to maximize reward (e.g., solve math problems correctly). Although such thinking environment satisfies the Markov property (i.e. future only depends on the current state), the state size is unbounded: $|s_t| = O(t)$ as actions are appended. Consequently, the final token of a response $\boldsymbol{y}$ is generated with context size $|s_T| = |\boldsymbol{x}| + |\boldsymbol{y}| - 1$. For attention-based policies (e.g., transformer LLMs), this occurs a quadratic computation cost with respect to thinking tokens throughout the training and inference.

4 Delethink

Delethink reformulates the standard RLVR by changing the underlying RL environment (MDP) that induces trajectories. The idea is to train policies to be native *Markovian Thinkers*: instead of accumulating one ever-growing chain of thought, the policy is forced to reason in *a sequence of fixed-size chunks*.

Within chunks, the model thinks as usual, But, at the end of each chunk, the environment resets the context to a fresh prompt seeded with a short carryover from the previous chunk (e.g., the last m tokens). Progress therefore depends on what the policy chooses to write into this carryover—its *textual Markovian state*—so it learns to advance reasoning while conditioning only on a bounded state.

These dynamics impose a hard cap on state size: at every step, the environment enforces $|s_t| = O(\mathcal{C})$ for a fixed per-chunk budget $\mathcal{C}$ (e.g., 8K tokens). The policy's effective context is therefore independent of the total reasoning length. For attention-based policies, this turns quadratic compute in the (growing) context length into compute that is quadratic only in the constant chunk size $\mathcal{C}$; consequently, the overall training cost scales linearly with the number of thinking tokens because the full trace never materializes in context at once. We formalize this compute scaling in Section 4.3 and compare against standard LongCoT RL. We next detail trajectory generation under this chunked dynamics and derive the corresponding policy-gradient estimator, completing the RL training loop.

4.1 Delethink Tracing

A policy trained under Delethink reasons in a sequence of chunks. We start by explaining trajectory generation process and then discuss how it effect transition dynamics, followed by practical implications for LLM inference.

Let $\boldsymbol{q}$ denote the query, typically containing a system prompt, chat history, and the user inquiry. In the first chunk, the model generates a response up to $\mathcal{C}$ tokens as usual with the input query as prompt: $\boldsymbol{y}_1 \sim \pi(\cdot|\boldsymbol{x}_1 = \boldsymbol{q})$, where $\boldsymbol{x}_1$

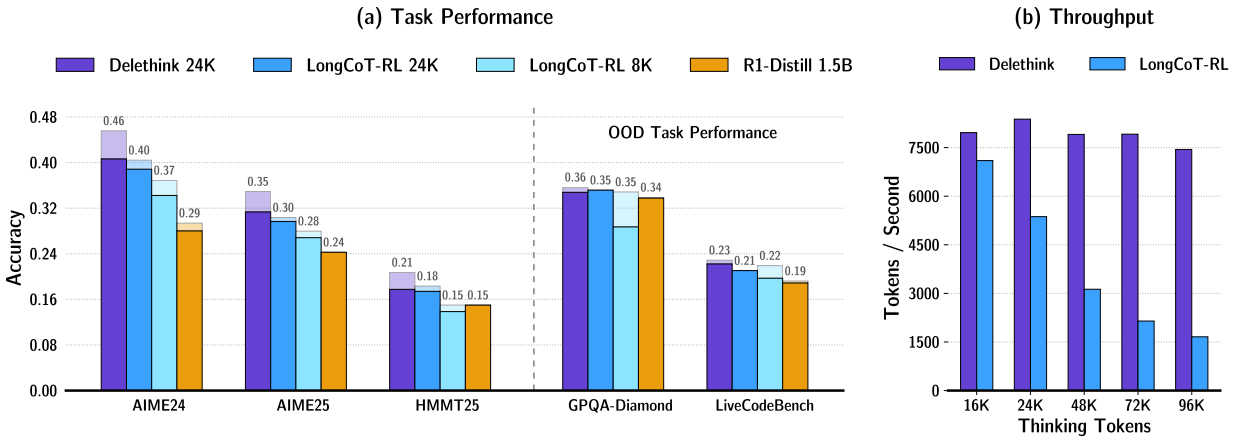


Figure 4: **(Left)** On IID math tasks (AIME'24/'25, HMMT'25), Delethink outperforms LongCoT-RL 24K. Shaded regions show gains from test-time scaling (through sequential sampling), where Delethink improves the performance even more; on OOD tasks (GPQA-Diamond, LiveCodeBench) gains are modest, yet Delethink still matches or slightly beats LongCoT-RL 24K. **(Right)** Per-GPU rollout throughput during RL training (R1-Distill 1.5B, H100 cluster). Delethink's RL environment design keeps peak memory constant, sustaining throughput as thinking scales; LongCoT's memory grows linearly, driving throughput down at longer budgets.

and $\boldsymbol{y}_1$ represent the first chunk's prompt and response, respectively. If the response $\boldsymbol{y}_1$ ends with `[EOS]`, the trace is complete. Otherwise, Delethink advances by constructing the next prompt from the original query and a *markovian state* consisting of the last $m < \mathcal{C}$ tokens of the previous chunk output[2]:

$$\boldsymbol{x}_l \;=\; \boldsymbol{q} \,\oplus\, \boldsymbol{y}_{(l-1)}[-m:], \quad l \geq 2,$$

where $\oplus$ denotes concatenation and $\boldsymbol{x}_l$ and $\boldsymbol{y}_l$ is the prompt and response for chunk l, respectively. In effect, preceding reasoning tokens are deleted when forming the next prompt (hence the name Delethink). Given $\boldsymbol{x}_l$, the model generates up to $\mathcal{C} - m$ new thinking tokens for chunk l: $\boldsymbol{y}_l \sim \pi(\cdot|\boldsymbol{x}_l)$. This procedure repeats until `[EOS]` is produced or the iteration cap $\mathcal{I}$ is reached. We refer to the resulting sequence as a *Delethink trace* $\tau \;=\; [(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_L, \boldsymbol{y}_L)]$, where L is number of chunks in the Delethink trace τ. We illustrate this process in Figure 1.[3]

This process for sampling trajectories can be seen as modifying the transition dynamics of the environment MDP at the boundary of chunks. Let $B \subseteq \mathcal{S} \times \mathcal{A}$ be a set of boundary state-action pairs, where $\mathcal{S}$ is the set of states and $\mathcal{A}$ is a set of all actions. Delethink redefines the transition dynamics as follows:

$$P(s_{t+1} = s' \mid s_t = s,\, a_t = a) = \begin{cases} 1, & (s,a) \in B \text{ and } s' = s_{:|\boldsymbol{q}|} \oplus s_{-m:} \oplus a, \\ 1, & (s,a) \notin B \text{ and } s' = s \oplus a, \\ 0, & \text{otherwise.} \end{cases}$$

In total, the LLM may think up to $\mathcal{C} + (\mathcal{I} - 1)\,(\mathcal{C} - m)$ tokens under Delethink environment, while each chunk's context remains bounded by $|\boldsymbol{q}| + \mathcal{C}$. Since query remains constant through the Delethink process and $|\boldsymbol{q}| \ll \mathcal{C}$ in practice, the maximum per-chunk model context is $O(\mathcal{C}) = O(1)$ with respect to the total number of thinking tokens. In the paper, we fix $m = \mathcal{C}/2$; Section E ablates the markovian state size.

KV cache under Delethink. Within each chunk, decoding uses the KV cache exactly as in standard transformer inference: generating $\boldsymbol{y}_l$ reuses keys/values from the prefix of the same chunk. At the chunk boundary, the environment resets the context and starts a new prompt $\boldsymbol{x}_{l+1}$, which includes $\boldsymbol{y}_l[-m:]$. Intuitively, these tokens are no longer conditioned on $\boldsymbol{x}_l$; they are recontextualized under $\boldsymbol{q}$. Therefore, the KV cache must be cleared at this point and the carryover tokens will be re-encoded as part of the new prefix. This adds an $O(m^2)$ encoding cost at each boundary, which does not change compute complexity of Delethink. Moreover, the prefill operation is quite fast compared to decoding in modern inference engine (Zheng et al., 2024).

[2] In practice, we fold the first hundred tokens of the initial chunk into $\boldsymbol{q}$ as it may contain planning tokens.

[3] We provide a sample implementation based on HuggingFace model in Figure C.2

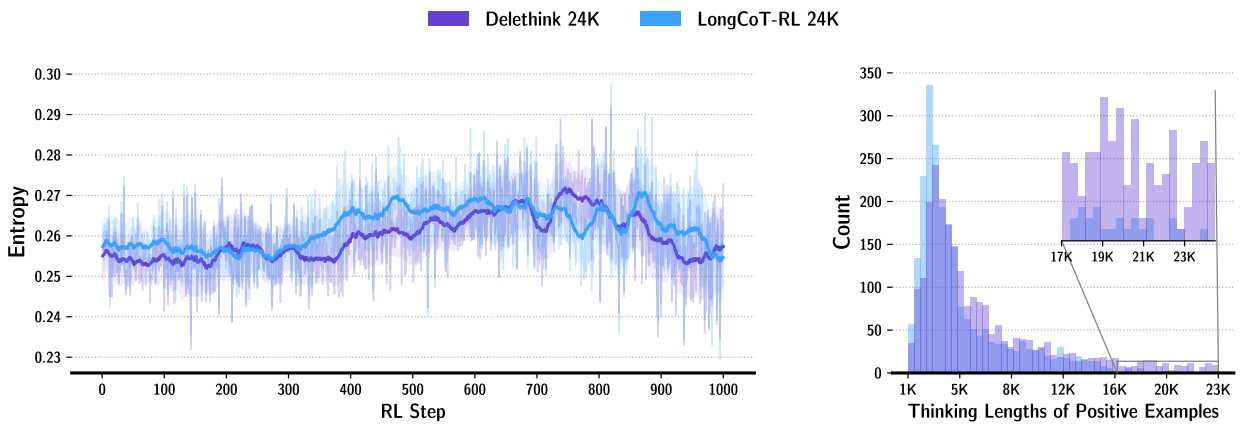


Figure 5: **(Left)** Smoothed entropy over RL steps for Delethink and LongCoT-RL. Both remain roughly flat and non-collapsing (Cui et al., 2025), indicating stable learning. Note that rising entropy typically precedes divergence. (**Right**) Delethink and LongCoT use their thinking budgets well. At longer lengths, Delethink produces more correct answers, showing it spends its budget effectively.

4.2 Policy Gradient

We start with the RL objective, maximizing the expected reward:

$$\mathcal{J}(\theta) = \mathbb{E}_{\boldsymbol{q}\sim\mathcal{D},\tau\sim\pi_\theta(\cdot|\boldsymbol{q})}\left[\mathcal{R}(\tau)\right] - \beta\,\mathrm{KL}[\pi_\theta\|\pi_{\text{ref}}],$$

where τ is a generated trace using under Delethink RL environment (Section 4.1). Optimizing this objective requires re-derivation of policy gradient as standard PPO-based estimators are derived under LongCoT MDP. Therefore, we derive the policy gradient estimator under Delethink dynamics in Section A. The resulting objective closely mirrors the policy gradient form of standard RL training for LLMs (Lambert et al., 2024), allowing an easy adaptation of existing infrastructure for Delethink. Intuitively, the policy gradient objective under Delethink environment sums over all the chunks in the trace, as each contributed to the final reward.

Given a query $\boldsymbol{q}$, we sample a group G of Delethink traces $\tau_1, \ldots, \tau_G$ from the current policy $\pi_{\theta_{\text{old}}}$, where each trace is a sequence of chunks $\tau = \{(\boldsymbol{x}_l, \boldsymbol{y}_l)\}_{l=1}^{L}$ with L denoting the number of chunks in the trace. Optimizing the expected return proceeds by taking gradients with respect to the following objective function:

$$J(\theta) = \mathbb{E}_{\tau_1,\ldots,\tau_G\sim\pi_{\theta_{\text{old}}}(\cdot|\boldsymbol{q})}\left[\frac{1}{G}\sum_{g=1}^{G}\underbrace{\frac{1}{\ell(\tau_g)}\sum_{l=1}^{L}\mathcal{U}(\boldsymbol{x}_l,\boldsymbol{y}_l;\theta)}_{\text{per-Delethink trace loss}}\right], \tag{3}$$

where $\ell(\tau) = \sum_l |\boldsymbol{y}_l|$ is the total number of response tokens in a Delethink trace τ. This is similar to GRPO, where per-trace term is normalized by the total number of thinking tokens. $\mathcal{U}(\boldsymbol{x}, \boldsymbol{y}; \theta)$ represents the per-chunk $(\boldsymbol{x}, \boldsymbol{y})$ objective which closely follow that of PPO in LLMs. Specifically,

$$\mathcal{U}(\boldsymbol{x},\boldsymbol{y},\theta) = \sum_{t=1}^{|\boldsymbol{y}|}\min\left[\frac{\pi_\theta(y_t)}{\pi_{\theta_{\text{old}}}(y_t)}\hat{A}_t, \text{clip}\left(\frac{\pi_\theta(y_t)}{\pi_{\theta_{\text{old}}}(y_t)}, 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right] - \beta\mathrm{KL}[\pi||\pi_{\text{ref}}],$$

where $\pi(y_t)$ is the probability of predicting token y_t of the chunk's response $\boldsymbol{y}$[4] and β controls the Kullback–Leibler term (Shao et al., 2024). The advantage $\hat{A}_t$ can be estimated with any off-the-shelf estimator (Kazemnejad et al., 2025; Yu et al., 2025). For simplicity we use the GRPO formulation: $A_{l,t} \equiv A_{\tau_g} := (\mathcal{R}(\tau_g) - \mu)/\sigma$, where $\mathcal{R}(\tau_g)$ is the reward for g-th trace (e.g. whether model reaches the correct answer at the end of last chunk). μ and σ are the mean and standard deviation of rewards across the trace group $\{\tau_g\}$. Pseudo-code is shown in Algorithm 1, illustrating training on a single query (we optimize over batches in practice).

[4]We omit conditioning on the context $\boldsymbol{x}, \boldsymbol{y}_{<t}$ for brevity.

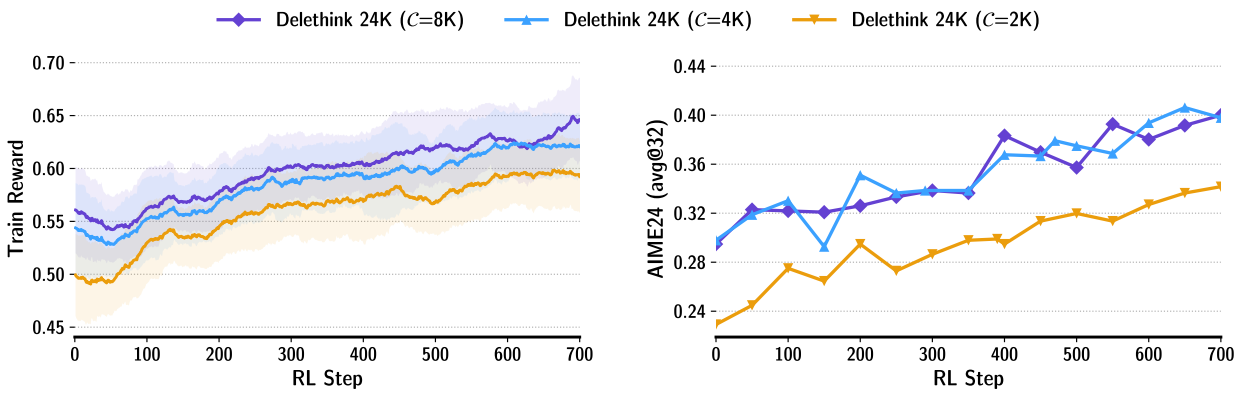


Figure 6: We vary $\mathcal{C}$ while holding the total thinking budget at $\approx$ 24K tokens, so smaller $\mathcal{C}$ implies a smaller Markovian state. **(Left)** smoothed training reward vs. RL step. **(Right)** validation accuracy (AIME'24) vs. RL step. Delethink 24K with $\mathcal{C} = 8$K and 4K performs similarly on both training reward and validation score, whereas 2K trails but improves steadily over the base model during Delethink RL.

4.3 Computational Cost of Scaling Thinking

An RL step has two parts: generation and backward for updating the policy. We study how both scale when an LLM's thinking length grows from n tokens to nS tokens under LongCoT and Delethink. As Table 3 shows, both stages scale linearly in Delethink but quadratically in LongCoT.

Total FLOPs and Backward Time Suppose training an LLM to think for n tokens costs $O(n^2)$ FLOPs. With LongCoT, scaling by S will cost $O(n^2S^2)$ because self-attention grows quadratically with length. Delethink instead runs in $2S$ chunks. Each chunk carries forward $\frac{n}{2}$ tokens and generates $\frac{n}{2}$ new ones (i.e., $m = \mathcal{C}/2$). The result is $O(2n^2S) = O(n^2S)$ FLOPs, linear in S. A factor of two arises because each token is both generated once and then reprocessed as the next chunk's prompt. These theoretical insights mirror the exact computed FLOPs for our experimental setup shown in Figure 3 based on R1Distill 1.5B architecture; in particular, since the compute of non-attention layers is linear in tokens, the two factor is evident.

Peak memory KV Cache entries have fixed size per token. In LongCoT, the KV cache grows linearly with thinking length, limiting parallel requests on the GPU. For example, the KV cache of a 1M-token trace on R1-Distill 1.5B, a small model, alone fills an entire H100. Going beyond requires sharding the sequence across GPUs with sequence-parallelism, adding heavy communication. In contrast, Delethink keeps only the current chunk's KV cache, so usage stays constant.

Generation Time Assume an infinite stream of requests saturating the GPU, each with maximum length L. The optimal throughput satisfies $T \propto \frac{1}{L}$ under simplifying assumptions (Ao et al., 2025; see B). This can be intuited from the fact that generation must access the KV cache, which grows linearly with sequence length, limiting parallel requests. LongCoT uses a growing KV cache; Delethink keeps it fixed per chunk. Both must generate nS tokens, but LongCoT's throughput falls by a factor of S, scaling time by S^2. Delethink's throughput is constant, so its time scales only as S.

Effect of $\mathcal{C}$ and m The analysis above takes $\mathcal{C} = n$ and $m = \frac{n}{2}$ as the baseline (our training setup). However, $\mathcal{C}$ and m are hyperparameters of Delethink. Relative to this baseline: if Delethink thinks for nS tokens using S chunks of length $2n$ (i.e., $\mathcal{C} = 2n$), the total FLOPs become $O(4n^2S)$, still linear in S but with a higher constant. Alternatively, if it thinks for nS tokens in $\frac{4}{3}S$ chunks with a context size of $\mathcal{C} = n$ and a smaller state $m = \frac{n}{4}$, the total FLOPs are $O(\frac{4}{3}n^2S)$, again linear in S but with a lower constant.

5 Experimental Setup

Model and Datasets We follow the RL training setup of Luo et al. (2025b). Specifically, we train on the DeepScaleR dataset, which contains approximately 40K curated, competition-level math problem–answer pairs. For all RL runs we

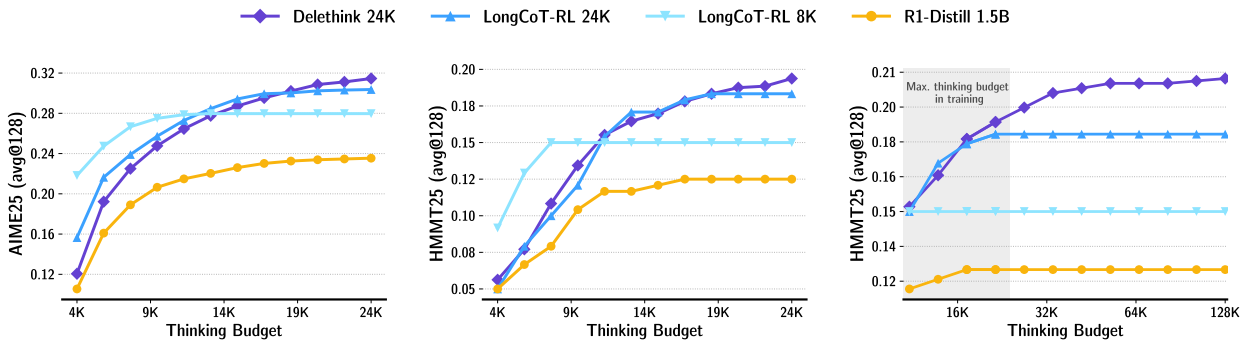


Figure 7: **Scaling the Thinking Budget at Inference.** Within the train-time budget, Delethink 24K and LongCoT-RL both gain accuracy as thinking tokens increase. Beyond that budget (shaded region), only Delethink continues to improve scaling from 24K up to 128K, while LongCoT-RL 24K and 8K plateau at their respective training limits.

initialize from *R1-Distill 1.5B* (Guo et al., 2025), a strong reasoning LLM. All models use *full-parameter fine-tuning*. For benchmarks, we evaluate on AIME'24, AIME'25 (MAA, 2025), and HMMT'25 (Balunović et al., 2025). As out-of-distribution (OOD) tests, we additionally evaluate on the GPQA Diamond split (Rein et al., 2024). We also report results on LiveCodeBench (Jain et al., 2025), restricting questions to those dated 2024-08-01–2025-02-01, following Guo et al. (2025). These two are OOD relative to our math-only training data because GPQA Diamond consists of Ph.D.-level, open-domain questions and LiveCodeBench evaluates code generation.

Baselines We compare Delethink-RL to LongCoT-RL under identical maximum thinking budgets and similar training environments. Our primary baseline is LongCoT-RL with a 24K token budget. We also include LongCoT-RL with an 8K budget to quantify the benefit of scaling thinking tokens. All accuracy numbers are *Pass@1* estimated from K samples, i.e. *Accuracy (avg@K)* denotes Pass@1 computed by averaging the correctness of K independently sampled responses for the same prompt. We select the best checkpoint using a hold-out validation set (AIME'24). To reduce evaluation variance, we sample 256 responses per prompt for every method and checkpoint, and report "Accuracy (avg@128)" via bootstrapping, yielding a standard deviation below 0.004 throughout.

Training Setup We follow the DeepScaleR recipe (Luo et al., 2025b) for our RL runs and rigorously implement established best practices. To ensure LongCoT-RL represents a strong performance, we conduct an hyperparameter search for LongCoT-RL. For Delethink, we adopt the same hyperparameters to ensure a fair comparison. In particular, we sweep the PPO clip-high ratio to maintain non-collapsing entropy during training for all runs (Figure 5). Each RL step samples 8 traces per prompt with a batch of 128 prompts (1024 episodes in total per step) and uses two optimizer updates per step. We continue the RL training for 1000 steps. Following recent practice, we disable the KL penalty ($\beta = 0$) for best performance (Yu et al., 2025). We also apply truncated importance sampling to mitigate the distributional mismatch between the inference and training engines, which stabilizes learning (Yao et al., 2025). In all runs, we employ a temperature 0.6 during training. The reward function give a score 1 if the models correctly answers question within the budget, and zero otherwise. For Delethink, we set the thinking context size to $\mathcal{C} = 8$K based on its superior performance under a 24K budget on the base model (Figure 9). We use a cap of $\mathcal{I} = 5$ iterations and a markovian state size $m = \mathcal{C}/2$. This configuration enables up to $8\text{K} + (5-1) \times 4\text{K} = 24\text{K}$ thinking tokens (Section 4.1), matching the LongCoT budget. We employ an optimized implementation of both Delethink and LongCoT-RL under the verl framework (Sheng et al., 2024) using SGLang (Zheng et al., 2024) and FSDP (Zhao et al., 2023) as backend with sequence-packing and dynamic micro-batch sizing enabled for most efficient performance. Training is performed on 8×H100 GPUs without sequence parallelism to reduce communication overhead. See Section D for full details.

6 Results

Delethink and LongCoT-RL 24K both have the same 24K-token thinking budget and start at the same pre-training performance (Figure 4). This equal initial performance shows that Delethink successfully samples markovian traces, with R1-Distill 1.5B, zero-shot which is favorable for RL training (We extensively investigate this zero-shot performance and its implications for scaling Delethink in Section 7). Despite Delethink runs in chunks of 8K, it can effectively think

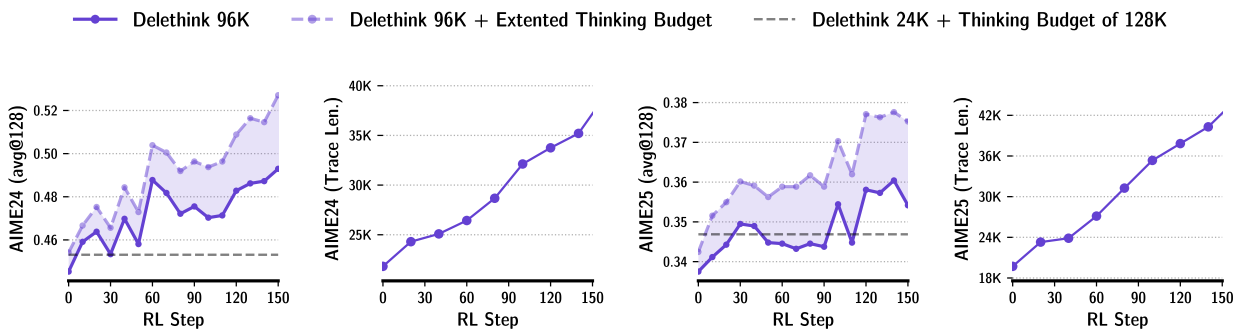


Figure 8: **Scaling Delethink to 96K.** AIME'24/'25 accuracy and average trace length vs. RL step for Delethink 96K; dashed curves extend the thinking budget to 256K and 128K tokens for Delethink 96K and Delethink 24K, respectively. Despite only 150 RL steps, 96K surpasses both the baseline and its extended thinking variant, with mean trace lengths reaching up to 42K.

up to 24K budget matching and, at equivalent RL steps, outperforming LongCoT-RL 24K during training (Figure 4), with each Delethink RL step being less costly. Thus, Delethink learns as effectively as LongCoT-RL while using fewer compute resources. The extended budget size is critical. With only 8K tokens, LongCoT-RL 8K underperforms Delethink by 5.5% (Figure 4), confirming that the additional 16K thinking tokens are necessary for the stronger results and that Delethink learns to leverage them. Figure 7 supports this as it shows that both LongCoT-RL 24K and Delethink increase their accuracy while spending their thinking budget.

6.1 Task Performance

We select the best checkpoint of each run based on validation performance on AIME'24 and then evaluate on math benchmarks. As shown in Figure 4, with a 24K training-time thinking-token budget, Delethink attains higher scores than LongCoT-RL 24K on AIME'24, AIME'25, and HMMT'25. LongCoT-RL 8K consistently underperforms, underscoring the necessity of extended reasoning. On out-of-distribution tasks, GPQA-Diamond and LiveCodeBench, the absolute gains are modest; nevertheless, Delethink matches or slightly surpasses LongCoT-RL 24K. Overall, these results indicate that Delethink uses its thinking tokens as effectively as LongCoT-RL with reduced compute.

6.2 Test-Time Scaling

The benefits of a Markovian Thinker are not limited to train-time limits. We investigate the test-time scaling behavior of Delethink and LongCoT-RL by reasoning beyond the thinking length they are originally trained on. Note that this is sequential test-time scaling, Pass@1, and not parallel test-time scaling methods like majority@K or pass@K.[5] As shown in Figure 2, both LongCoT-RL 24K and LongCoT-RL 8K quickly plateau once they reach their training-time limits, indicating that LongCoT-RL's test-time scaling is largely constrained by its training budget. In contrast, Delethink continues to improve even when reasoning with 100K more tokens than it encountered during training. For example, some AIME'25 problems are only solved after reasoning with up to 140K thinking tokens (See Section H), despite the model being trained with 24K. This pattern holds across all benchmarks, indicating that Delethink enables genuine test-time scaling far beyond training-time limits.

6.3 Empirical Compute

We evaluate the compute efficiency of Delethink against LongCoT-RL 24K. Delethink completes each RL step in 215s, faster than LongCoT-RL 24K at 248.5s. Its token generation is also more efficient, reaching 8,500 tokens per second per H100 versus 6,000 for LongCoT-RL 24K. As a result, Delethink finishes batch response generation in 130s compared to LongCoT-RL's 170s, in line with the predictions of Section 4.3. The backward pass takes 80s for Delethink while 70s for LongCoT-RL 24K. At first glance, this appears to contradict theory. However, the difference arises from constant

[5]We omit budget-forcing test-time scaling methods such as S1 (Guo et al., 2025) because they hurt performance on reasoning LLMs; see Section G.

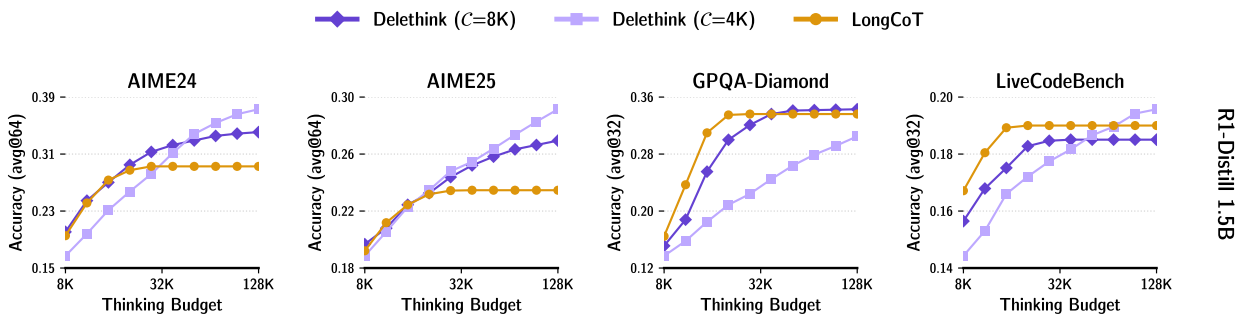


Figure 9: **Delethink Tracing at Initialization.** Even at initialization of RL, applying Delethink Tracing, without extra prompts or training,recovers most of LongCoT performance across thinking budgets. This indicates early signs of *Markovian Thinking* in the base model.

and lower-order terms that dominate at shorter sequence lengths. Figure C.1 shows the crossover for R1-Distill 1.5B. Below 32K tokens, Delethink can be slower since non-attention blocks dominate, but beyond that point, the quadratic scaling of attention makes Delethink increasingly faster. At one million tokens, Delethink achieves $17\times$ reduction in FLOPs. In Figure 2, we empirically measure the time of a single RL step, assuming a target average thinking length. As the theory predicts, the time grows quadratically for LongCoT but only linearly for Delethink. To better understand this, we also measure the throughput of inference engines under both methods (Figure 4). Consistent with the theory, Delethink maintains the same throughput regardless of thinking length, while LongCoT shows a steady decline.

6.4 Context Size Ablation

Our main experiment runs Delethink with a context size, $\mathcal{C}$, of 8K. We ablate the context size to 4K and 2K, set the state size m to half the context size, and adjust iterations so the maximum thinking length remains 24K tokens. We refer to these as 4K and 2K, respectively, while calling the main run 8K. As shown in Figure 6, both 4K and 2K learn to exploit the very small state size to perform Markovian Thinking. The 4K variant slightly underperforms 8K. The 2K variant starts from a much lower initialization point. Although it achieves lower accuracy than both 8K and 4K throughout, it still improves beyond the base model performance, indicating that Delethink works even under minimal context sizes.[6]

6.5 Scaling Delethink to 96K

We study scaling the thinking budget from 24K to 96K tokens, made feasible by the linear compute cost of Delethink RL training. Concretely, we keep the thinking context to $\mathcal{C} = 8$K and increase the iteration cap from $\mathcal{I} = 5$ to $\mathcal{I} = 23$, yielding a 96K total budget. For this run, we train on OpenMath (Moshkov et al., 2025), which contains more difficult math competition problems than DeepScaleR and typically requires longer reasoning. We initialize from the Delethink 24K checkpoint and continue training for 150 steps, using the same hyperparameters as earlier runs. Figure 8 reports AIME'24 and AIME'25 results. Despite the short training schedule, Delethink 96K not only surpasses the base performance of Delethink 24K checkpoint, but also matches or exceeds a test-time–scaled Delethink 24K evaluated with a 128K token budget. Moreover, the model's average thinking lengths reach 36K (AIME'24) and 42K (AIME'25) tokens, indicating effective use of the larger budget. These preliminary results provide clear evidence that Delethink can scale to very long reasoning traces.

7 Why Delethink Works?

RL works best when the policy has a decent prior. That is, there is non-trivial probability of sampling trajectories with desired behavior. We investigate Delethink Tracing at initialization of RL training: when Delethink Tracing applies broadly to off-the-shelf reasoning LLMs without any training or prompting, zero-shot. We observe that these models

[6]Minimal context-size Delethink models might especially be of interest for low-memory settings like mobile devices. Instead of running a small reasoning model with LongCoT, they can run a much larger reasoning model with Delethink with a compact context size.

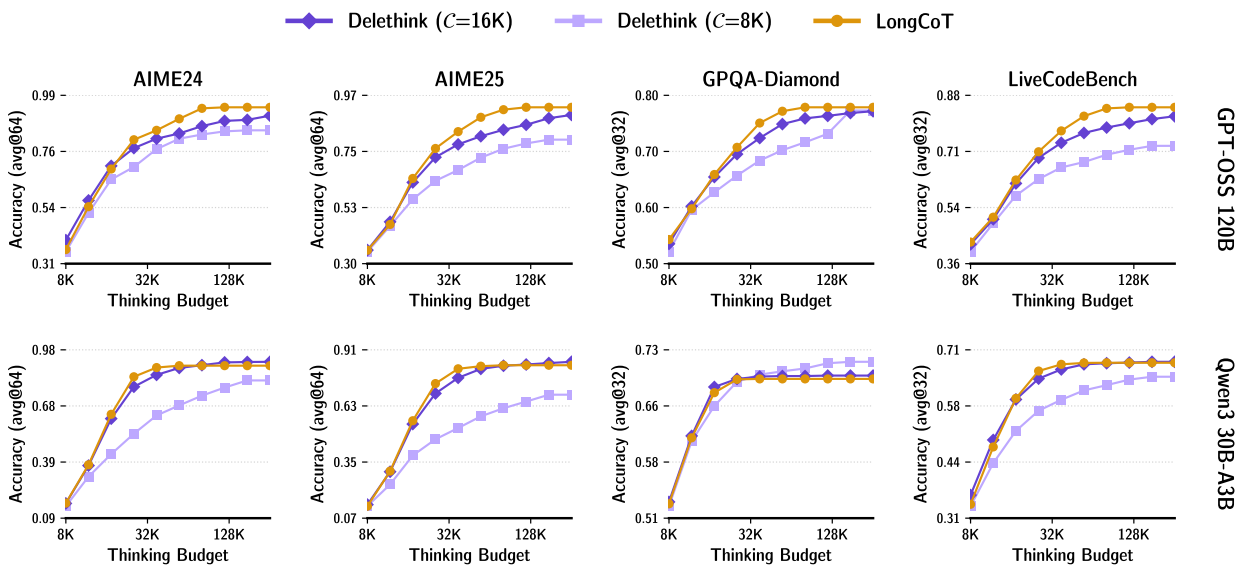


Figure 10: State-of-the-art reasoning LLMs, GPT-OSS-120B and Qwen3-30B-A3B, are capable of Markovian Thinking zero-shot, providing a strong initialization for training, signaling scalability. Delethink closely tracks LongCoT and recovers most of its final accuracy (the curves nearly coincide on Qwen3)

show strong evidence of Markovian Thinking: they either recover or exceed their standard LongCoT performance. Therefore, Delethink starts RL training from a strong and reliable initialization point. This explains, at least partially, why delethink training succeeds. First, we showcase this phenomenon on R1-Distill 1.5B which is used in our training. Second, we show the strong presence of this phenomenon in SOTA LLMs, providing early signal that Delethink scales.

7.1 R1-Distill

We evaluate the R1-Distill 1.5B up to 128K thinking tokens. For Delethink, we vary the per-chunk context C from 4K to 8K to measure sensitivity to chunk size. As shown in Figure 9, Delethink Tracing with 8K matches LongCoT across tasks (Figure 9), despite deviating from its training-time regime. This provides a strong initialization for Delethink, exactly where our RL experiments started. Delethink Tracing with 4K only falls behind in the GPQA-Diamond, but exceeds on rest. Furthermore, Delethink Tracing exhibits superior test-time scaling: on both AIME'24 and AIME'25 it surpasses its LongCoT performance by significant margin. In contrast, LongCoT quickly plateaus at 32K tokens, which is its recommended max thinking length (Guo et al., 2025). Interestingly, Delethink Tracing with 4K test-time scales further. We hypothesize one reason is because the model is unaware of the tracing procedure, it appears to perceive more headroom, emits `[EOS]` less, and thus thinks more. Finally, see Section F for evidence that Delethink can resume from LongCoT-RL checkpoints, where LongCoT-RL 24K shows solid performance under Delethink Tracing.

7.2 Will Delethink Scale?

Training larger models is not possible within our compute budget. However, we can assess Delethink at step zero, when Delethink Tracing is applied zero-shot, to test whether the policy already supports Markovian traces. Strong performance at initialization, without training or prompting, is a promising signal that subsequent RL training will succeed, since it begins with many positive samples and the desired behavior already in-distribution.

To test this, we evaluate GPT-OSS 120B (Agarwal et al., 2025) and Qwen3 30B-A3B Thinking[7] (Yang et al., 2025), two state-of-the-art open-source reasoning LLMs, under Delethink. We find that a 16K context is a solid default as it recovers most of the original performance, while 8K achieves a reasonable score and is suitable when compute is tight. We also study problem coverage at initialization to check whether the model loses the ability to solve some questions compared to standard LongCoT, and find that coverage remains equally strong. Finally, we benchmark the models on a

[7] https://huggingface.co/Qwen/Qwen3-30B-A3B-Thinking-2507

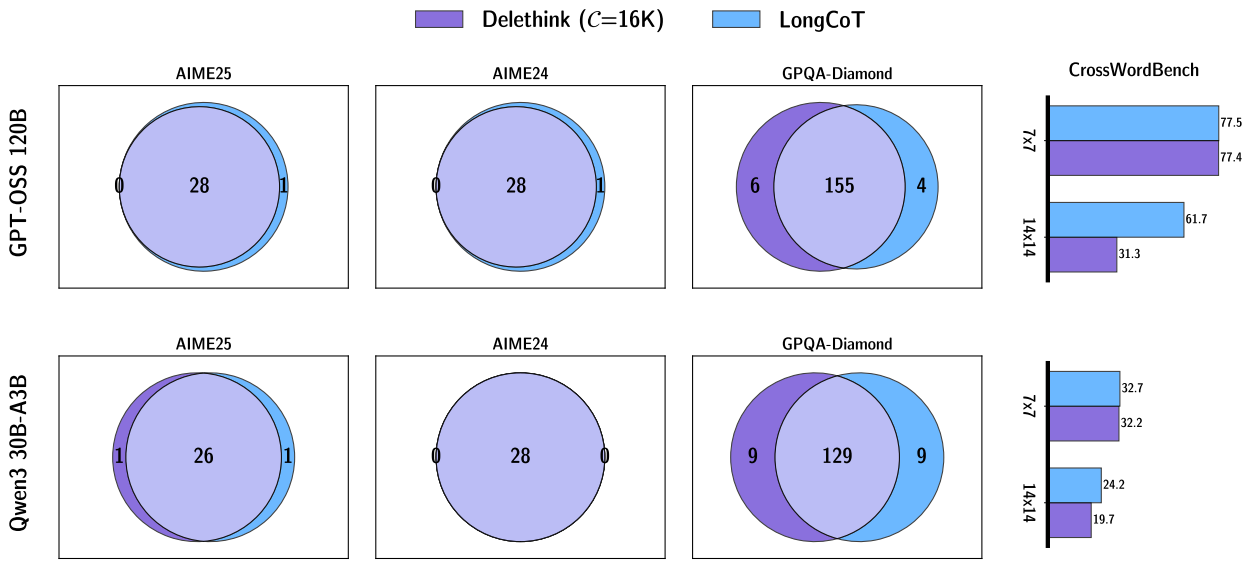


Figure 11: (**Left**) Problem-solving overlap of Delethink vs. LongCoT: on AIME '24 and '25 they solve nearly the same set of questions. On GPQA, each solves an equal number that the other misses. (**Right**) CrossWordBench stress-tests Delethink: deleting previous tokens removes access to already found words. Delethink remains competitive, but its zero-shot limits are evident.

task that requires live memory to deliberately break Delethink, and find that even there they admit Markovian solutions. Overall, these results suggest that Delethink has a strong chance of scaling effectively.

Results and Test-time scaling As shown in Figure 10, both models achieve higher scores by more reasoning, although less pronounced for Qwen3, under both LongCoT and Delethink Tracing. Delethink Tracing, with $\mathcal{C} = 16$K, matches or exceeds LongCoT on Qwen3 and almost matches on GPT-OSS. Tracing with $\mathcal{C} = 8$K has a lower test-time scaling slope and lags behind LongCoT. These results indicate that current SOTA reasoning LLMs exhibit latent markovian behavior even without explicit training for it and Delethink with 8K context size provides a strong initialization for starting Delethink.

Problem Coverage We compare problem coverage on AIME'24, AIME'25, and GPQA-Diamond (Figure 11), counting a problem as solved if the majority vote is correct.[8] On both AIMEs, both methods solve essentially the same number of problems. In detail, LongCoT solves one more problem on both AIMEs with GPT-OSS. For Qwen3, AIME'24 is identical; on AIME'25 each method uniquely solves one problem (equal totals). On GPQA-Diamond, GPT-OSS solves 161 problems with Delethink vs 159 with LongCoT; Qwen3 solves 138 with both.

Stress testing Delethink Tracing We search for a task where Delethink does not start from a strong initialization. As Delethink loses access to old thoughts, tasks that require writing to a memory like crossword might be challenging as solved cases may be lost if not carried by the model. We evaluate on CrossWordBench (Leng et al., 2025), which contains crossword puzzles at varying difficulty levels. Reasoning in this task requires maintaining a live grid plus filled entries—state that can exceed the capacity of m. As shown in Figure 11, Delethink Tracing achieves performance on 7×7 puzzles that is broadly comparable to LongCoT across both GPT-OSS and Qwen3. On the more challenging 14×14 puzzles, performance declines relative to LongCoT, though both models continue to produce a nontrivial number of valid Markovian traces. Overall, these results suggest that even in settings designed to stress-test Delethink, it preserves meaningful probabilistic coverage and readily discovers markovian solutions, giving headroom for training.

[8] Majority voting does not apply to code, so LiveCodeBench is excluded.

Effect of per-chunk context size C Delethink Tracing with 16K context size recovers almost full-context performance. Smaller C, 8K, however underperforms larger C and the LongCoT baseline. As the markovian state m carried between chunks shrinks too much, the model may lose the thread of reasoning, suggesting the zero-shot state size should be higher. These results show 16K context works decently from scratch, but 8K requires extra RL training to make the state more markovian.

8 Discussion

Scaling thinking length has advanced reasoning LLMs significantly, but further scaling in the current LongCoT paradigm incurs a prohibitive quadratic compute cost. In this work, we present a new paradigm that decouples the thinking length and the context size. We propose the Markovian Thinking Paradigm, where each step retains only the minimal state needed to continue, extending thinking under a bounded state size; yielding linear compute and constant memory for both training and subsequent test-time generation. We show Delethink is practical. Delethink trains native markovian thinkers by starting from off-the-shelf reasoning LLMs. Experiments match LongCoT performance and show superior test-time scaling. Furthermore, we empirically verify the practicality of Delethink in scaling the thinking budget to nearly one hundred thousand tokens. Finally, we both theoretically and empirically demonstrate the practical benefits of scaling thinking with Delethink compared to LongCoT.

Additionally, here we share some hypotheses that might help explain some surprising results further; these are speculative and meant to inspire rigorous follow-up. In our experiments, Delethink surpassed LongCoT-RL 24K. Pretraining might have played a role. As most LLM pretraining uses short contexts (e.g., 8K tokens), models may think sharper in shorter contexts. This aligns with retrieval research showing effective context lengths shorter than nominal limits (Hsieh et al., 2024; Modarressi et al., 2025). We also hypothesize that Delethink samples positive responses zero-shot partly because LLMs are pre-trained on human reasoning traces and human reasoning may be roughly Markovian, carrying necessary results forward in spelled-out thoughts. Note that 8K tokens is near 20 textbook pages. Finally, RL on Delethink might encourage more abstract reasoning. With limited context, an LLM may learn tokens that encapsulate broader ideas to save space. Observing this may require large-scale RL.

Thanks to Delethink, Markovian Thinking is no longer a theoretical concept but a practical, near-term reality. The possibility and effectiveness of markovian thinking, that Delethink shows, could change how we design reasoning models. Delethink thinks and learns just as well without its full history as LongCoT does with it. This could never work in areas like long-context retrieval, where the model depends on information the whole context. This shows that thinking is different and can, in fact, be done in a Markovian way. Delethink provides evidence that reasoning LLMs can be trained differently and may exploit more efficient architectures such as non-quadratic attention, sliding-window attention, sparse attention, or state-space models, since reasoning can be done Markovian. We hope this paper serves as a stepping stone toward models that think across millions of tokens.

Acknowledgements

SR is supported by a CIFAR AI Chair and a Mila–Samsung grant. AC is supported by Canada CIFAR AI Chair, the Canada Research Chair, the NSERC Discovery Grant, and funding from Microsoft Research. SC is supported by the Canada CIFAR AI Chair, the Canada Research Chair in Lifelong Machine Learning, and the NSERC Discovery Grant. We thank the Mila IDT team and the Digital Research Alliance of Canada for providing the compute resources used in our experiments.

References

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *ArXiv preprint*, abs/2508.10925, 2025. URL https://arxiv.org/abs/2508.10925.

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

AI21 Labs. Jamba-1.5: Hybrid transformer-mamba models at scale. *ArXiv preprint*, abs/2408.12570, 2024. URL https://arxiv.org/abs/2408.12570.

Ruicheng Ao, Gan Luo, David Simchi-Levi, and Xinshang Wang. Optimizing llm inference: Fluid-guided online scheduling with memory constraints. *ArXiv preprint*, abs/2504.11320, 2025. URL https://arxiv.org/abs/2504.11320.

Mislav Balunović, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. Matharena: Evaluating llms on uncontaminated math competitions, 2025. URL https://matharena.ai/.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *ArXiv preprint*, abs/2004.05150, 2020. URL https://arxiv.org/abs/2004.05150.

Jeffrey Cheng and Benjamin Van Durme. Compressed chain of thought: Efficient reasoning through dense representations. *ArXiv preprint*, abs/2412.13171, 2024. URL https://arxiv.org/abs/2412.13171.

Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamás Sarlós, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J. Colwell, and Adrian Weller. Rethinking attention with performers. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=Ua6zuk0WRH.

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*, abs/2505.22617, 2025. URL https://arxiv.org/abs/2505.22617.

Muzhi Dai, Chenxu Yang, and Qingyi Si. S-grpo: Early exit via reinforcement learning in reasoning models. *ArXiv preprint*, abs/2505.07686, 2025. URL https://arxiv.org/abs/2505.07686.

Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=ztn8FCR1td.

Mengru Ding, Hanmeng Liu, Zhizhang Fu, Jian Song, Wenbo Xie, and Yue Zhang. Break the chain: Large language models can be shortcut reasoners. *ArXiv preprint*, abs/2406.06580, 2024. URL https://arxiv.org/abs/2406.06580.

Siqi Fan, Peng Han, Shuo Shang, Yequan Wang, and Aixin Sun. Cothink: Token-efficient reasoning via instruct models guiding reasoning models. *ArXiv preprint*, abs/2505.22017, 2025. URL https://arxiv.org/abs/2505.22017.

Gemini Team. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *ArXiv preprint*, abs/2403.05530, 2024. URL https://arxiv.org/abs/2403.05530.

Albert Gu and Tri Dao. Linear-time sequence modeling with selective state spaces. *ArXiv preprint*, abs/2312.00752, 2023. URL https://arxiv.org/abs/2312.00752.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=uYLFoz1vlAC.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *ArXiv preprint*, abs/2501.12948, 2025. URL https://arxiv.org/abs/2501.12948.

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. Token-budget-aware llm reasoning. *ArXiv preprint*, abs/2412.18547, 2024. URL https://arxiv.org/abs/2412.18547.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length LLM inference with KV cache quantization. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/028fcbcf85435d39a40c4d61b42c99a4-Abstract-Conference.html.

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

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *ArXiv preprint*, abs/2412.16720, 2024. URL https://arxiv.org/abs/2412.16720.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=chfJJYC3iL.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5156–5165. PMLR, 2020. URL http://proceedings.mlr.press/v119/katharopoulos20a.html.

Amirhossein Kazemnejad, Milad Aghajohari, Eva Portelance, Alessandro Sordoni, Siva Reddy, Aaron Courville, and Nicolas Le Roux. VinePPO: Refining credit assignment in RL training of LLMs. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=Myx2kJFzAn.

Hynek Kydlíček. Math-Verify: Math Verification Library, 2025. URL https://github.com/huggingface/math-verify.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, et al. Tulu 3: Pushing frontiers in open language model post-training. *ArXiv preprint*, abs/2411.15124, 2024. URL https://arxiv.org/abs/2411.15124.

Jixuan Leng, Chengsong Huang, Langlin Huang, Bill Yuchen Lin, William W. Cohen, Haohan Wang, and Jiaxin Huang. Crosswordbench: Evaluating the reasoning capabilities of llms and lvlms with controllable puzzle generation, 2025. URL https://arxiv.org/abs/2504.00043.

Chen Li, Nazhou Liu, and Kai Yang. Adaptive group policy optimization: Towards stable training and token-efficient reasoning. *ArXiv preprint*, abs/2503.15952, 2025. URL https://arxiv.org/abs/2503.15952.

Omer Lieber, Benjamin Lenz, Ido Szpektor, Kevin Leyton-Brown, et al. Jamba: A hybrid transformer-mamba language model. *ArXiv preprint*, abs/2403.19887, 2024. URL https://arxiv.org/abs/2403.19887.

Weizhe Lin, Xing Li, Zhiyuan Yang, Xiaojin Fu, Hui-Ling Zhen, Yaoyuan Wang, Xianzhi Yu, Wulong Liu, Xiaosong Li, and Mingxuan Yuan. Trimr: Verifier-based training-free thinking compression for efficient test-time scaling, 2025. URL https://arxiv.org/abs/2505.17155.

Tengxiao Liu, Qipeng Guo, Xiangkun Hu, Cheng Jiayang, Yue Zhang, Xipeng Qiu, and Zheng Zhang. Can language models learn to skip steps? In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024a. URL http://papers.nips.cc/paper_files/paper/2024/hash/504fa7e518da9d1b53a233ed20a38b46-Abstract-Conference.html.

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*, abs/2503.20783, 2025. URL https://arxiv.org/abs/2503.20783.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024b. URL https://openreview.net/forum?id=L057s2Rq8O.

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*, abs/2501.12570, 2025a. URL https://arxiv.org/abs/2501.12570.

Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y. Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Li Erran Li, Raluca Ada Popa, and Ion Stoica. Deepscaler: Surpassing o1-preview with a 1.5b model by scaling rl. https://pretty-radio-b75.notion.site/DeepScaleR-Surpassing-O1-Preview-with-a-1-5B-Model-by-Scaling-RL, 2025b. Notion Blog.

MAA. American invitational mathematics examination (aime) 2024: Aime i and aime ii. https://maa.org/maa-invitational-competitions/, 2025. Official competition problems provided by the MAA.

Ali Modarressi, Hanieh Deilamsalehy, Franck Dernoncourt, Trung Bui, Ryan A Rossi, Seunghyun Yoon, and Hinrich Schütze. Nolima: Long-context evaluation beyond literal matching. *arXiv preprint arXiv:2502.05167*, 2025.

Ivan Moshkov, Darragh Hanley, Ivan Sorokin, Shubham Toshniwal, Christof Henkel, Benedikt Schifferer, Wei Du, and Igor Gitman. Aimo-2 winning solution: Building state-of-the-art mathematical reasoning models with openmathreasoning dataset. *ArXiv preprint*, abs/2504.16891, 2025. URL https://arxiv.org/abs/2504.16891.

OpenAI. Gpt-5 system card. https://cdn.openai.com/gpt-5-system-card.pdf, 2025. Accessed: 2025-09-17.

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 *First Conference on Language Modeling*, 2024.

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*, abs/2402.03300, 2024. URL https://arxiv.org/abs/2402.03300.

Yi Shen, Jian Zhang, Jieyun Huang, Shuming Shi, Wenjing Zhang, Jiangze Yan, Ning Wang, Kai Wang, Zhaoxiang Liu, and Shiguo Lian. Dast: Difficulty-adaptive slow-thinking for large reasoning models. *ArXiv preprint*, abs/2503.04472, 2025. URL https://arxiv.org/abs/2503.04472.

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. *arXiv preprint arXiv: 2409.19256*, 2024.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *ArXiv preprint*, abs/2006.04768, 2020. URL https://arxiv.org/abs/2006.04768.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

Heming Xia, Chak Tou Leong, Wenjie Wang, Yongqi Li, and Wenjie Li. Tokenskip: Controllable chain-of-thought compression in llms. *ArXiv preprint*, abs/2502.12067, 2025. URL https://arxiv.org/abs/2502.12067.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=NG7sS51zVF.

Yang Xiao, Jiashuo Wang, Ruifeng Yuan, Chunpu Xu, Kaishuai Xu, Wenjie Li, and Pengfei Liu. Limopro: Reasoning refinement for efficient and effective test-time scaling. *ArXiv preprint*, abs/2505.19187, 2025. URL https://arxiv.org/abs/2505.19187.

Yuchen Yan, Yongliang Shen, Yang Liu, Jin Jiang, Mengdi Zhang, Jian Shao, and Yueting Zhuang. Inftythink: Breaking the length limits of long-context reasoning in large language models. *ArXiv preprint*, abs/2503.06692, 2025. URL https://arxiv.org/abs/2503.06692.

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*, abs/2505.09388, 2025. URL https://arxiv.org/abs/2505.09388.

Shuo Yang, Ying Sheng, Joseph E Gonzalez, Ion Stoica, and Lianmin Zheng. Post-training sparse attention with double sparsity. *ArXiv preprint*, abs/2408.07092, 2024a. URL https://arxiv.org/abs/2408.07092.

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024b. URL https://openreview.net/forum?id=ia5XvxFUJT.

Feng Yao, Liyuan Liu, Dinghuai Zhang, Chengyu Dong, Jingbo Shang, and Jianfeng Gao. Your efficient rl framework secretly brings you off-policy rl training, 2025. URL https://fengyao.notion.site/off-policy-rl.

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*, abs/2503.14476, 2025. URL https://arxiv.org/abs/2503.14476.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontañón, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/c8512d142a2d849725f31a9a7a361ab9-Abstract.html.

Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, et al. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *ArXiv preprint*, abs/2508.06471, 2025. URL https://arxiv.org/abs/2508.06471.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark W. Barrett, Zhangyang Wang, and Beidi Chen. H2O: heavy-hitter oracle for efficient generative inference of large language models. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/6ceefa7b15572587b78ecfcebb2827f8-Abstract-Conference.html.

Haoran Zhao, Yuchen Yan, Yongliang Shen, Haolei Xu, Wenqi Zhang, Kaitao Song, Jian Shao, Weiming Lu, Jun Xiao, and Yueting Zhuang. Let llms break free from overthinking via self-braking tuning. *ArXiv preprint*, abs/2505.14604, 2025a. URL https://arxiv.org/abs/2505.14604.

Weixiang Zhao, Jiahe Guo, Yang Deng, Xingyu Sui, Yulin Hu, Yanyan Zhao, Wanxiang Che, Bing Qin, Tat-Seng Chua, and Ting Liu. Exploring and exploiting the inherent efficiency within large reasoning models for self-guided efficiency enhancement. *ArXiv preprint*, abs/2506.15647, 2025b. URL https://arxiv.org/abs/2506.15647.

Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, et al. Pytorch fsdp: experiences on scaling fully sharded data parallel. *ArXiv preprint*, abs/2304.11277, 2023. URL https://arxiv.org/abs/2304.11277.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark W. Barrett, and Ying Sheng. Sglang: Efficient execution of structured language model programs. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC,*

Canada, December 10 - 15, 2024, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/724be4472168f31ba1c9ac630f15dec8-Abstract-Conference.html.

Adrian Łańcucki, Konrad Staniszewski, Piotr Nawrot, and Edoardo M. Ponti. Inference-time hyper-scaling with kv cache compression. *ArXiv preprint*, abs/2506.05345, 2025. URL https://arxiv.org/abs/2506.05345.

Appendix

Table of Content

A Deriving Delethink Loss

In this section, we show that the Delethink Training loss is basically the policy gradient and derive it from scratch. This proof will be trivial to readers familiar with policy gradients (Sutton et al. (1999)). We include it for completeness.

The RL objective is to maximize the expected return which is the following:

$$\mathcal{J}(\theta) = \mathbb{E}_{\boldsymbol{q}\sim\mathcal{D},\tau\sim\pi_\theta(\cdot|\boldsymbol{q})}\left[\mathcal{R}(\tau)\right] \tag{4}$$

Policy gradient is the gradient of the expected return with respect to the policy parameters. That is:

$$\nabla_\theta\mathcal{J}(\theta) = \nabla_\theta\mathbb{E}_{\boldsymbol{q}\sim\mathcal{D},\tau\sim\pi_\theta(\cdot|\boldsymbol{q})}\left[\mathcal{R}(\tau)\right] \tag{5}$$

First, note that the gradient cannot pass through the expectations as we have distributional dependency. That is, the distribution of τ depends on θ. However, the gradient can pass through the expectation over the queries as the distribution of dataset does not depend on θ.

$$\nabla_\theta\mathcal{J}(\theta) = \nabla_\theta\mathbb{E}_{\boldsymbol{q}\sim\mathcal{D}}\int_{\tau\in\pi_\theta(\cdot|\boldsymbol{q})}\pi_\theta(\tau)\mathcal{R}(\tau) \tag{6}$$

As noted, the distribution of q does not depend on θ, the gradient passes through this expectation:

$$\nabla_\theta\mathcal{J}(\theta) = \mathbb{E}_{\boldsymbol{q}\sim\mathcal{D}}\nabla_\theta\int_{\tau\in\pi_\theta(\cdot|\boldsymbol{q})}\pi_\theta(\tau)\mathcal{R}(\tau) \tag{7}$$

Next, the gradient passes through the integration as the bounds of the integration does not depend on θ:

$$\nabla_\theta\mathcal{J}(\theta) = \mathbb{E}_{\boldsymbol{q}\sim\mathcal{D}}\int_{\tau\in\pi_\theta(\cdot|\boldsymbol{q})}\nabla_\theta\pi_\theta(\tau)\mathcal{R}(\tau) \tag{8}$$

However, the term insides the integral is intractable as it is an integration over all possible traces. In order to write this as an expectation so we can estimate this via samples we divide and multiply $\pi_\theta(\tau)$:

$$\begin{aligned}\nabla_\theta\mathcal{J}(\theta) &= \mathbb{E}_{\boldsymbol{q}\sim\mathcal{D}}\int_{\tau\in\pi_\theta(\cdot|\boldsymbol{q})}\nabla_\theta\pi_\theta(\tau)\mathcal{R}(\tau)\\ &= \mathbb{E}_{\boldsymbol{q}\sim\mathcal{D}}\int_{\tau\in\pi_\theta(\cdot|\boldsymbol{q})}\pi_\theta(\tau)\frac{\nabla_\theta\pi_\theta(\tau)}{\pi_\theta(\tau)}\mathcal{R}(\tau)\end{aligned} \tag{9}$$

However, the term inside the integral is the derivative of the log probability with respect to the parameters, famously known as the log-trick. That is, $\frac{\nabla_\theta\pi_\theta(\tau)}{\pi_\theta(\tau)} = \nabla_\theta\log(\pi_\theta(\tau))$. Therefore, we have:

$$\begin{aligned}\nabla_\theta\mathcal{J}(\theta) &= \mathbb{E}_{\boldsymbol{q}\sim\mathcal{D}}\int_{\tau\in\pi_\theta(\cdot|\boldsymbol{q})}\pi_\theta(\tau)\nabla_\theta\log(\pi_\theta(\tau))\mathcal{R}(\tau)\\ &= \mathbb{E}_{\boldsymbol{q}\sim\mathcal{D},\tau\sim\pi_\theta(\cdot|\boldsymbol{q})}\nabla_\theta\log\pi_\theta(\tau)\mathcal{R}(\tau)\end{aligned} \tag{10}$$

Note that, this is the famous REINFORCE (Williams (1992)). Next, we expand $\log(\pi_\theta(\tau))$. We know the Delethink trace τ is a sequence of chunks. That is, $\tau = [(\boldsymbol{x}_1,\boldsymbol{y}_1),\ldots,(\boldsymbol{x}_L,\boldsymbol{y}_L)]$. Therefore, we can write the log probability of generating τ as the sum of log probabilities of generating each chunk. That is,

$$\log(\pi_\theta(\tau)) = \sum_{l=1}^{L}\log\pi_\theta(\boldsymbol{y}_l) \tag{11}$$

And if we expand the log probability of generating each chunk into sum of log probability of generating the response tokens:

$$\log(\pi_\theta(\tau)) = \sum_{l=1}^{L}\sum_{t=1}^{|\boldsymbol{y}_l|}\log\pi_\theta(\boldsymbol{y}_{l,t}\mid x_l,\boldsymbol{y}_{:t-1}) \tag{12}$$

By substituting this to 10, we get the following:

$$\nabla_\theta\mathcal{J}(\theta) = \mathbb{E}_{\boldsymbol{q}\sim\mathcal{D},\tau\sim\pi_\theta(\cdot|\boldsymbol{q})}\sum_{l=1}^{L}\sum_{t=1}^{|\boldsymbol{y}_l|}\nabla_\theta\log\pi_\theta(\boldsymbol{y}_{l,t})\mathcal{R}(\tau) \tag{13}$$

Next, according to Sutton et al. (1999), instead of the reward, we can multiply the log probability of each token with its corresponding advantage:

$$\nabla_\theta \mathcal{J}(\theta) = \mathbb{E}_{\boldsymbol{q}\sim\mathcal{D},\tau\sim\pi_\theta(\cdot|\boldsymbol{q})} \sum_{l=1}^{L} \sum_{t=1}^{|\boldsymbol{y}_l|} \nabla_\theta \log \pi_\theta(\boldsymbol{y}_{l,t}) \hat{A}_t. \tag{14}$$

This concludes the derivation of the policy gradient form of the objective. However, we will proceed to add PPO clipping, KL loss, and the normalization. They bias the gradient, so the term is not the exact policy gradient anymore. First, we add the PPO clippings:

$$\mathbb{E}_{\boldsymbol{q}\sim\mathcal{D},\tau\sim\pi_\theta(\cdot|\boldsymbol{q})} \sum_{l=1}^{L} \sum_{t=1}^{|\boldsymbol{y}_l|} \min\left[\frac{\pi_\theta(\boldsymbol{y}_{l,t})}{\pi_{\theta_{\text{old}}}(\boldsymbol{y}_{l,t})}\hat{A}_t,\ \text{clip}\left(\frac{\pi_\theta(\boldsymbol{y}_{l,t})}{\pi_{\theta_{\text{old}}}(\boldsymbol{y}_{l,t})}, 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right] \tag{15}$$

Additionally, we add the normalization by the total number of response tokens in the trace $\ell(\tau_g)$:

$$\mathbb{E}_{\boldsymbol{q}\sim\mathcal{D},\tau\sim\pi_\theta(\cdot|\boldsymbol{q})} \frac{1}{\ell(\tau_g)} \sum_{l=1}^{L} \sum_{t=1}^{|\boldsymbol{y}_l|} \min\left[\frac{\pi_\theta(\boldsymbol{y}_{l,t})}{\pi_{\theta_{\text{old}}}(\boldsymbol{y}_{l,t})}\hat{A}_t,\ \text{clip}\left(\frac{\pi_\theta(\boldsymbol{y}_{l,t})}{\pi_{\theta_{\text{old}}}(\boldsymbol{y}_{l,t})}, 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right] \tag{16}$$

Finally, we add the KL term that limits deviation from the reference policy.

$$\mathbb{E}_{\boldsymbol{q}\sim\mathcal{D},\tau\sim\pi_\theta(\cdot|\boldsymbol{q})} \frac{1}{\ell(\tau_g)} \sum_{l=1}^{L} \sum_{t=1}^{|\boldsymbol{y}_l|} \min\left[\frac{\pi_\theta(\boldsymbol{y}_{l,t})}{\pi_{\theta_{\text{old}}}(\boldsymbol{y}_{l,t})}\hat{A}_t,\ \text{clip}\left(\frac{\pi_\theta(\boldsymbol{y}_{l,t})}{\pi_{\theta_{\text{old}}}(\boldsymbol{y}_{l,t})}, 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right] - \beta\,\text{KL}[\pi_\theta \,\|\, \pi_{\text{ref}}] \tag{17}$$

Note that this is basically our objective.

B Deriving Throughput Relation

We follow Ao et al. (2025). Consider a GPU with maximum memory M, serving an infinite stream of incoming requests and an equally unbounded stream of completed outputs. Let n^* denote the equilibrium number of concurrent requests. Each request has l prefill tokens and l' decode tokens. In steady state, the KV-cache memory required is

$$M^* \;=\; n^*\left(l + \tfrac{l'}{2}\right). \tag{18}$$

According to Ao et al. (2025), the equilibrium throughput of an attention-based LLM under a memory constraint is

$$\mathcal{T}^* \;=\; \frac{n^*}{d_0 \;+\; d_1\, n^*\left(l + \frac{l'}{2}\right)}, \tag{19}$$

where d_0 is the fixed per-batch overhead and d_1 is the time cost per unit of memory. Thus, throughput exhibits an inverse dependence on the total effective context length. In regimes where $d_1\, n^*\left(l + \frac{l'}{2}\right) \gg d_0$, we obtain the approximation

$$\text{Throughput}^* \;\approx\; \frac{n^*}{d_1\, n^*\left(l + \frac{l'}{2}\right)} \;=\; \frac{1}{d_1\left(l + \frac{l'}{2}\right)}. \tag{20}$$

Consequently, for the same GPU, the ratio of throughputs when decoding lengths are l'_1 and l'_2 satisfies

$$\frac{\mathcal{T}_{l'_1}}{\mathcal{T}_{l'_2}} \;\approx\; \frac{\frac{1}{d_1\left(l+\frac{l'_1}{2}\right)}}{\frac{1}{d_1\left(l+\frac{l'_2}{2}\right)}} \;=\; \frac{l + \frac{l'_2}{2}}{l + \frac{l'_1}{2}}.$$

In the long-thinking regime where $l' \gg l$, this simplifies to

$$\frac{\mathcal{T}_{l'_1}}{\mathcal{T}_{l'_2}} \;\approx\; \frac{l'_2}{l'_1},$$

establishing that throughput is (approximately) inversely proportional to the decoding length in this regime. □

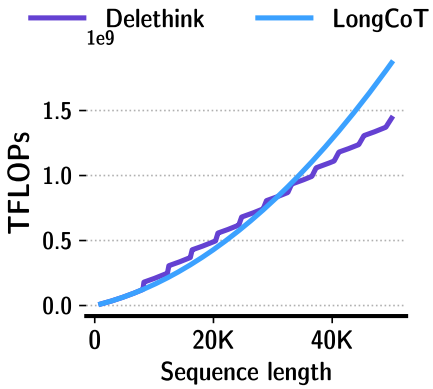


Figure C.1: When training R1DistillQwen1.5B with Delethink with 8k thinking context, for thinking lengths $\sim$ 30k the number of FLOPs for both is equal. This is because the non-attention components like dense layers dominate here. However, after 30k quadratic cost of attention dominates.

Intuition. Decoding repeatedly accesses the KV cache, whose size, and therefore memory-time cost, grows linearly with the number of decoded tokens l'. This linear growth drives the inverse relationship between throughput and l' in the long-thinking regime.

C Delethink FLOPs Crossover vs. LongCoT 24K

We can compute, in closed form, the FLOPs required for reasoning of a certain length under Delethink and LongCoT. We calculated this for our main experiment setup, where Delethink runs with $\mathcal{C} = 8\text{K}, m = 4\text{K}$, single step with 1000 episodes, and results are shown in Figure 3. We highlight the crossover in Figure C.1 at short sequences. Up to roughly 30K tokens, Delethink spends slightly more FLOPs than LongCoT. That seems counterintuitive: Delethink scales linearly, LongCoT quadratically. This is because in Delethink each token is generated once and then reprocessed as input to the next chunk. While the quadratic compute of attention quickly overtakes this double-counting cost in longer sequences, it matters at short lengths. This is clear when considering non-attention layers. FLOPs in non-attention layers scale linearly, and in Delethink each token is generated once and then reprocessed in the next chunk as a prompt. Therefore, the cost of non-attention layers doubles. The crossover occurs around 30K tokens in our setting. Beyond this point, Delethink is linearly more efficient in terms of FLOPs. Note that while total backward time tracks total FLOPs, generation time at these response lengths is still slower for LongCoT. This is because generation throughput is limited by memory-access and memory-footprint bottlenecks, not just FLOPs. As explained in Section 6.3, our training still runs faster even in this regime due to faster generation. Our next step, Delethink 96K, is significantly faster than its counterpart LongCoT-RL 96K in both backward pass and generation, to the point that we could not train the latter under our compute resources.

D Detailed RL Training Setup

In this section we describe our training setup and hyperparameters meticulously.

Model All reinforcement learning runs begin from R1-Distill 1.5B (Guo et al., 2025), a strong reasoning LLM. We employ full-parameter fine-tuning throughout training.

Datasets Our experiments follow the RL training framework introduced in Luo et al. (2025b). We train on the DeepScaleR dataset, which includes roughly 40K carefully curated, competition-style math problem–solution pairs. Evaluation is conducted a hold-out math competition benchmark: AIME'24, AIME'25 (MAA, 2025), and HMMT'25 (Balunović et al., 2025).

```python
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM

# Load model and tokenizer
model_path = "deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B"
tokenizer = AutoTokenizer.from_pretrained(model_path)
model = AutoModelForCausalLM.from_pretrained(model_path)

C, m, I, T = 8192, 4096, 5, 0.6  # chunk context, markovian tail, iteration cap, temperature

query_ids = tokenizer.apply_chat_template(
    [{"role": "user", "content": "Solve the problem step by step; ..."}],
    tokenize=True,
    add_generation_prompt=True,
)

trace_ids = []  # list of generated token IDs across chunks
prompt_ids = list(query_ids)  # current-chunk prompt (reset each iteration)
it = 0

while it < I:
    max_new = C if it == 0 else (C - m)

    inp = torch.tensor([prompt_ids], device=model.device)
    out = model.generate(
        input_ids=inp,
        max_new_tokens=max_new,
        do_sample=True, temperature=T,
        eos_token_id=tokenizer.eos_token_id, pad_token_id=tokenizer.pad_token_id,
        use_cache=True,
    )

    full_seq = out.sequences[0].tolist()
    response_ids = full_seq[len(prompt_ids) :]
    trace_ids.extend(response_ids)

    if it == 0:
        # Fold the first hundered tokens of the first chunk into query
        query_ids = query_ids + response_ids[:100]

    if response_ids[-1] == tokenizer.eos_token_id:
        break

    prompt_ids = query_ids + response_ids[-m:]
    it += 1

response_text = tokenizer.decode(trace_ids, skip_special_tokens=False)
response_text
```

Figure C.2: Example of **Delethink-tracing** chunked rollout implemented with HuggingFace `model.generate` (batch size $= 1$). In practice, we use high-performance inference engines and a large batch size.

Baselines We benchmark Delethink-RL against LongCoT-RL, ensuring identical maximum thinking budgets and closely matched training conditions. The main baseline is LongCoT-RL with a 24K-token budget, alongside an additional 8K-token configuration to highlight the effects of scaling reasoning tokens. Accuracy is reported as Pass@1, estimated from K independent samples. Specifically, "Accuracy (avg@K)" is the average Pass@1 computed over K sampled responses for the same prompt. To quantify uncertainty, we use a nonparametric bootstrap over continuations: for B=5000 replicates, for each prompt we resample K continuations *with replacement*, average within-prompt, then average across prompts. Model checkpoints are selected based on performance on the held-out AIME'24 validation set. To reduce noise, we generate 256 responses per prompt for each method and checkpoint, and report Accuracy (avg@128) using this bootstrap aggregation, yielding standard deviations consistently below 0.004. Evaluation is done

Table D.1: Key hyperparameters for **LongCoT RL** vs. **Delethink**.

Hyperparameter	Delethink	LongCoT RL
Rollout (inference)		
Sampling (T / top-p / top-k / n)	0.6 / 1.0 / -1 / 8	0.6 / 1.0 / -1 / 8
Prompt / response length (tokens)	2,048 / 8,192	2,048 / 24,576
Algorithm		
Advantage estimator	GRPO	GRPO
KL penalty	N/A	N/A
Policy optimization		
PPO epochs / number of grad updates	1 / 2	1 / 2
Clip ratio (low / high)	0.20 / 0.26	0.20 / 0.26
Grad clip (global-norm)	1.0	1.0
Loss aggregation	Trace Length-Batch Mean	Seq-mean-Batch Mean
Actor optimizer		
Learning rate	1×10^{-6}	1×10^{-6}
Weight decay	0	0
Warmup	constant; steps $= -1$ (disabled)	constant; steps $= -1$ (disabled)
Total training steps (optim)	1,000	1,000
Data		
Train / val batch size	128 / 16 (per step, logical)	128 / 16 (per step, logical)
Rewarding		
Reward function	HF-Math-Verify	HF-Math-Verify

with temperature 0.6, top$-p = 1.0$ and no top-k (disabled). For test-time scaling, we right-trimmed each trace so its thinking token count matched the specified test-time budget.

Training Setup We adopt the DeepScaleR RL recipe (Luo et al., 2025b), adhering to widely recognized best practices. To establish LongCoT-RL as a strong baseline, we perform a hyperparameter sweep, and then apply the same settings to Delethink-RL for fairness. In particular, we tune the PPO clip-high ratio (from 0.2 to 0.28) to maintain entropy stability across all runs (Figure 5). Each RL step consists of sampling 8 traces for 128 prompts (1024 episodes total) and applying two optimizer updates. Training is run for 1000 steps. Following recent findings, the KL penalty is disabled ($\beta = 0$) for optimal results (Yu et al., 2025). Truncated importance sampling is also applied to address distributional mismatches between inference and training engines, improving stability (Yao et al., 2025). Training is conducted with temperature 0.6.

Delethink Parameters For Delethink-RL, we set the reasoning context size to $\mathcal{C} = 8$K, chosen due to its effectiveness under a 24K-token budget with the base model (Figure 9). We cap iterations at $\mathcal{I} = 5$, with a Markovian state size of $m = \mathcal{C}/2$. This setup allows up to $8\text{K} + (5-1) \times 4\text{K} = 24\text{K}$ reasoning tokens (Section 4.1), aligning with the LongCoT-RL budget.

Implementation Framework Both Delethink-RL and LongCoT-RL are implemented efficiently within the verl framework (Sheng et al., 2024), leveraging SGLang (Zheng et al., 2024) and PyTorch FSDP (Zhao et al., 2023), with sequence-packing and dynamic micro-batching for throughput optimization. Training is executed on 8×H100 GPUs, avoiding sequence parallelism to minimize communication overhead. We use HF Math-verify (Kydlíček, 2025) as our reward function.

D.1 Hyperparameters

Table D.1 represents the key hyperparamters used in our RL training experiments.

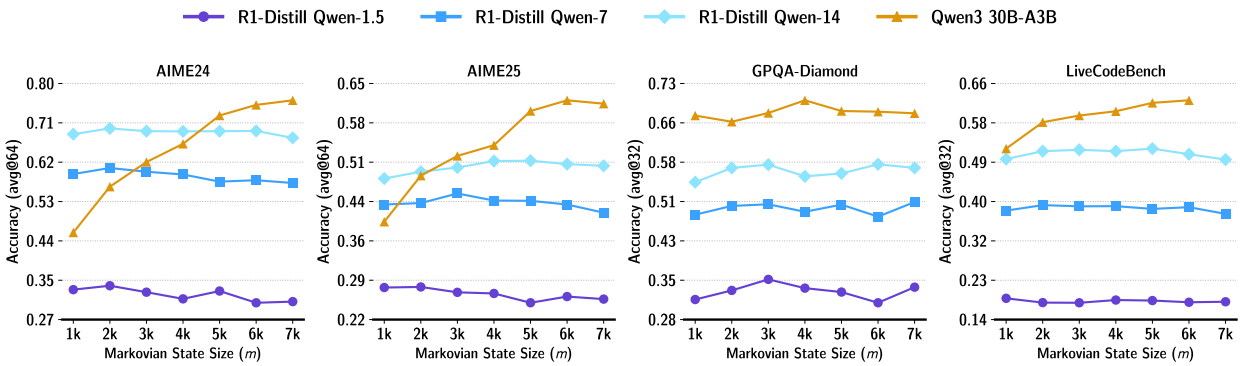


Figure E.3: Ablation of the Markovian state size m at fixed per-chunk context $\mathcal{C} = 8$k. R1-Distill models (1.5B to 14B) achieve stable accuracy even with small carried states ($m \ll \mathcal{C}$), indicating that only modest memory is required for Markovian behavior. In contrast, Qwen3 (native 256k context) shows clear gains from larger state sizes, reflecting its long-context prior and the longer Delethink traces on AIME and LiveCodeBench tasks.

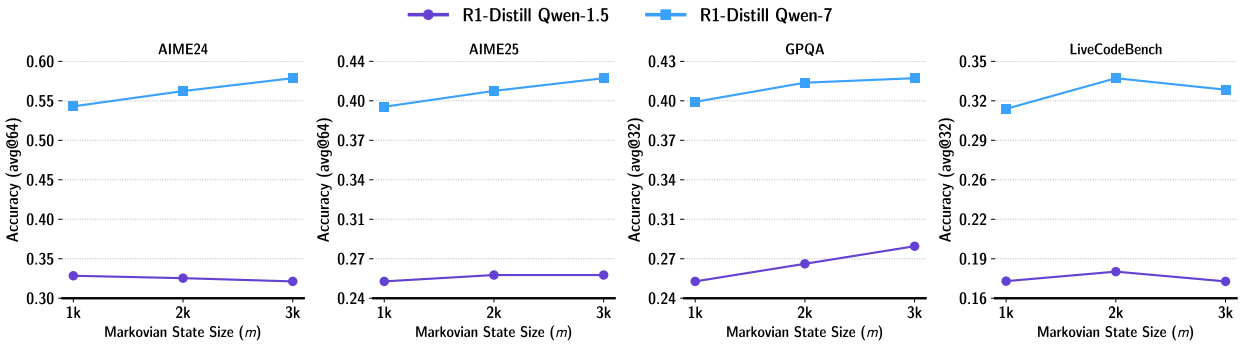


Figure E.4: Ablation of the Markovian state size m at fixed per-chunk context $\mathcal{C} = 4$k. In contrast to $\mathcal{C} = 8$k, R1-Distill 7B shows gains from larger state sizes across most tasks, whereas R1-Dstill 1.5B only benefits from larger state sizes in AIME'25 and GPQA.

E Markovian State Size Ablation

In this ablation we vary the size of the Markovian state m while holding the per-chunk thinking context fixed at $\mathcal{C} = 8$k and keeping the overall token budget constant. We sweep $m \in \{1\text{k}, 2\text{k}, 3\text{k}, 4\text{k}, 5\text{k}, 6\text{k}, 7\text{k}\}$. To ensure a matched budget across settings, we adjust the Delethink iteration cap $\mathcal{I}$ using the identity $T_{\max} = \mathcal{C} + (\mathcal{I} - 1)(\mathcal{C} - m)$. Concretely, we set $\mathcal{I} = \{6, 7, 8, 10, 13, 19, 37\}$ for $m = \{1\text{k}, 2\text{k}, 3\text{k}, 4\text{k}, 5\text{k}, 6\text{k}, 7\text{k}\}$, respectively. We evaluate R1–Distill models (1.5B, 7B, 14B) and Qwen3 30B-A3B on AIME'24, AIME'25, GPQA–Diamond, and LiveCodeBench under identical decoding and verification. Results are summarized in Figure E.3.

Surprisingly, for the R1–Distill models the final accuracy is essentially flat across the entire range of m, Given $\mathcal{C} = 8$k, shrinking the carried state from 7k down to 1k has negligible effect on all tasks. In contrast, Qwen3 30B-A3B exhibits a marked dependence on m for AIME'24, AIME'25, and LiveCodeBench, with larger states yielding higher accuracy. This difference aligns with two observations. First, Qwen3 is trained with a substantially larger native context window 256k compared to 32k for the R1 models. Second, the average Delethink trace lengths on AIME and LiveCode are considerably longer than on GPQA–Diamond, which explains why the state size matters more on these tasks. Overall, these results indicate that $\mathcal{C} = 8$k suffices for robust Markovian behavior in R1 models with only a modest state carryover, while Qwen3 benefits from a larger state that better matches its long–context prior on long–reasoning workloads. However, with $\mathcal{C} = 4$k, even R1-Distill family show similar pattern as Qwen3 (Figure E.4). This suggest that there is an interplay between the context size and the average response length.

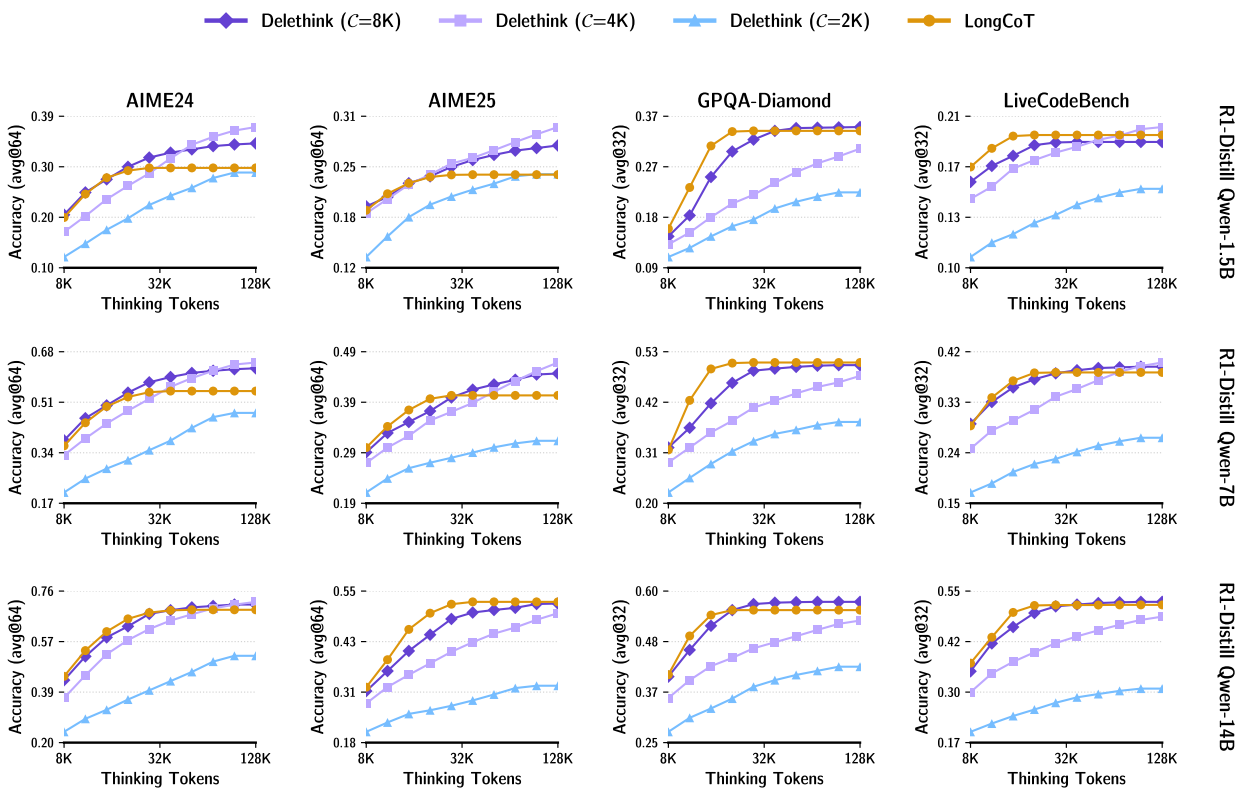


Figure E.5: Thinking Context Size Ablation. Scaling behavior of R1-Distill models across AIME'24, AIME'25, GPQA-Diamond, and LiveCodeBench under varying per-hunk contexts $\mathcal{C} \in \{2\text{k}, 4\text{k}, 8\text{k}$. Accuracy is plotted against total thinking tokens, with smaller contexts requiring proportionally more iterations.

E.1 Thinking Context Size Ablation

We next ablate the effect of the per-chunk thinking context size $\mathcal{C}$ on test-time scaling across all R1–Distill models (1.5B, 7B, 14B) and four benchmarks (AIME'24, AIME'25, GPQA–Diamond, LiveCodeBench). In these experiments, the x–axis reports the total thinking budget (number of generated tokens). For smaller context sizes, Delethink necessarily performs more iterations to reach the same total budget: for example, Delethink with $\mathcal{C} = 2\text{k}$ executes roughly four times as many iterations as $\mathcal{C} = 8\text{k}$ to reach 128k tokens. We set the carried state to $m = \mathcal{C}/2$ in every configuration so that only $\mathcal{C}$ is varied. This setup enables a direct comparison of how context size alone impacts performance under a fixed token budget.

The results, shown in Figure E.5, emphasize the strong zero–shot Markovian behavior of R1 models. Despite not being trained for Markovian tracing, all R1 models steadily improve with increased budget and in many cases surpass the LongCoT baseline. Most notably, for the 1.5B model the $\mathcal{C} = 4\text{k}$ configuration is highly effective: its scaling curve is nearly log–linear, and it even outperforms the $\mathcal{C} = 8\text{k}$ setting on AIME'24, AIME'25, and LiveCodeBench. This surprising robustness highlights that smaller chunks can sometimes be more efficient for test–time scaling in smaller models. As the model size increases to 7B and 14B, the advantage of $\mathcal{C} = 4\text{k}$ diminishes and $\mathcal{C} = 8\text{k}$ is typically best or tied. By contrast, $\mathcal{C} = 2\text{k}$ proves extremely limited: it rarely approaches the LongCoT baseline and only manages to catch up on the AIME tasks for the 1.5B model at very high budgets. This consistent underperformance suggests that such a short context severely constrains Delethink's ability to maintain stable traces.

The patterns become even clearer when examining completion behavior and response lengths (Figure E.6). With $\mathcal{C} = 2\text{k}$, models almost never terminate within budget, leading to dramatically lower EOS rates across tasks and scales. Correspondingly, the average response lengths at $\mathcal{C} = 2\text{k}$ are much higher than at 4k or 8k, reflecting the model's difficulty in finalizing its reasoning. This behavior is consistent with the limited horizon imposed by such a short per–chunk view: the model repeatedly resets positional embeddings, and the bias toward continuation overwhelms the

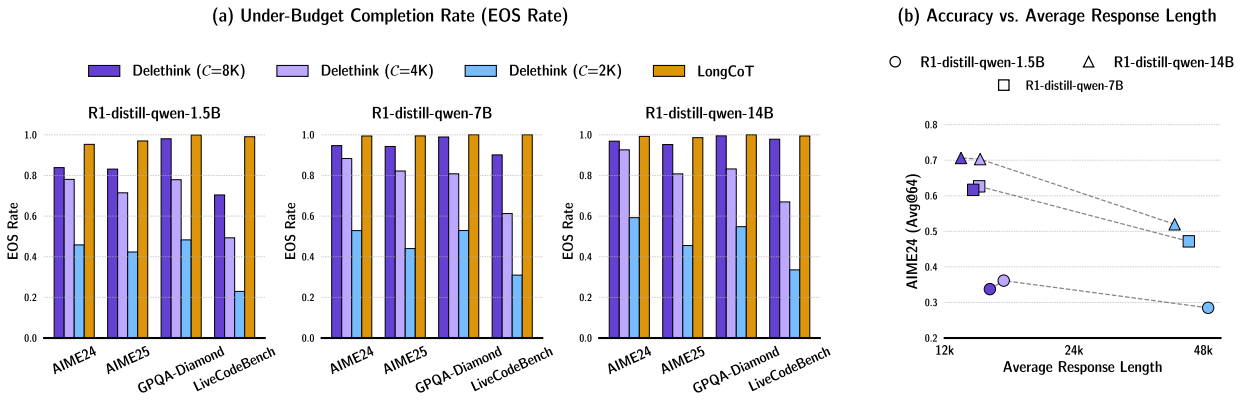


Figure E.6: Limitation of extremely low thinking context size. (a) EOS rate across tasks shows that under a 2k context, R1 models almost never terminate within budget, unlike 4k and 8k. (b) On AIME'24, $\mathcal{C} = 2$k produces much longer responses for all model sizes, indicating difficulty finalizing reasoning due to short local horizon and positional bias.

signal to conclude. In practice, this prevents stable Markovian progression and explains why $\mathcal{C} = 2$k remains far below LongCoT except in a few cases (e.g., AIME at 1.5B scale).

Overall, these results provide further evidence that R1 models exhibit strong Markovian behavior even without explicit training. Smaller contexts can be unexpectedly effective: $\mathcal{C} = 4$k not only scales smoothly but, for the 1.5B model, even surpasses $\mathcal{C} = 8$k on several tasks. At the same time, the severe degradation of $\mathcal{C} = 2$k highlights the limits of compressing the local horizon too aggressively, as models struggle to converge their reasoning and rarely complete within budget. Together, the ablations show that test-time Markovian scaling is a robust property of off-the-shelf R1 models, while also clarifying the practical regimes in which context size supports or hinders long-horizon reasoning.

F Delethink Tracing on LongCoT checkpoints

As discussed in Section 7, off-the-shelf LLMs show Markovian traces under Delethink, allowing us to generate traces even from models not explicitly trained with it. This suggests that applying Delethink Tracing could boost the performance of the LongCoT-24k checkpoint, despite it not being trained with Delethink. We investigate this in Figure F.7. We find that Delethink Tracing enables the LongCoT-24k checkpoint to test-time scale far beyond its training limits. Its performance increases by nearly 4%, almost as much as the gain it achieved from its entire RL training, at no additional training cost. This is a surprising phenomenon. Our hypothesis is that constraining queries to fewer than 8k tokens creates the illusion, for the 24k baseline, that it still has capacity left to continue. However, we find that when queries are kept under 16k, this effect vanishes, dissolving the *infinite budget illusion*. While these results indicate that Delethink Tracing might enhance traditionally trained checkpoints, we note that such baselines require quadratically more compute to train than Delethink.

G Budget Force Ablation: Does S1 work on R1?

We evaluate *Budget-Force* decoding scheme inspired by the S1 "simple test-time scaling" protocol (Guo et al., 2025). We run decoding and, whenever the model *finalizes early* before using the allotted token budget, we cut the response at the first finalization marker and append a short continuation cue before regenerating from the *entire* prompt. Concretely, for each query we first sample with temperature 0.6, top-$p = 0.95$, $n = 1$ under a 32k token budget. If the model emits any of `</think>`, `**Final Answer**`, or `\boxed{` (or EOS) before the budget is consumed, we trim at the earliest occurrence, append the literal cue `Wait\n`, and re-issue a generation from query + cut_response. We repeat this micro-forcing until the round budget is exhausted or a continuation cap (20) is reached. To avoid distributional drift, we do not alter model decoding settings mid-run; the only intervention is the prompt-level cut-and-continue. At the end, if the model still has not emitted EOS after fully using the budget, we add a compact finalization hint (`**Final Answer**\n`) and request a short natural continuation. All models identical sampling parameters across conditions.

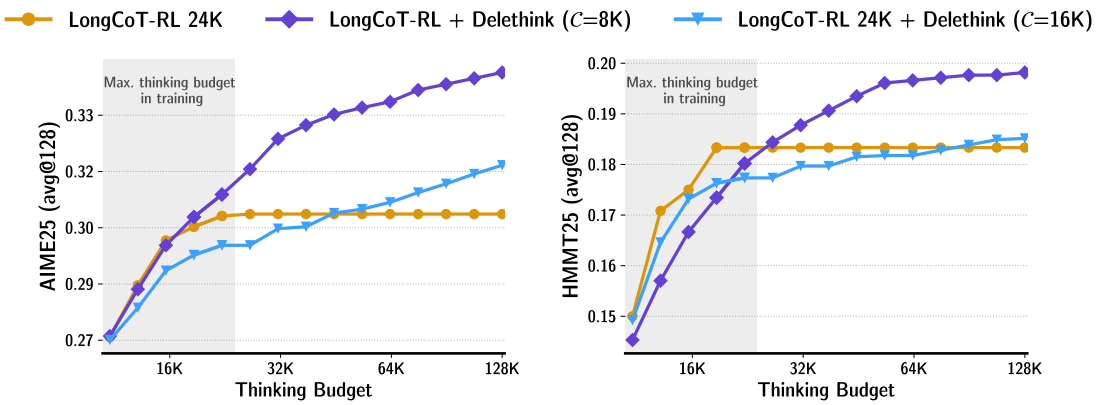


Figure F.7: Delethink Tracing on a LongCoT–RL 24K checkpoint (AIME'25, avg@128). LongCoT plateaus near its trained budget, while Delethink ($\mathcal{C} = 8$k, $\mathcal{C} = 16$k) keeps scaling; $\mathcal{C} = 8$k yields the larger gain. The shaded region marks the trained budget.

Table G.2: Accuracy (avg@128, %) for LongCoT, Budget-Force, and Delethink across benchmarks, using identical sampling and each method's native scaling protocol.

Model	Method	AIME24	AIME25	GPQA-Diamond	LiveCodeBench
R1-distill-qwen-1.5B	Long CoT 32k	29.3	23.6	33.9	19.2
	Budget Force	24.6	19.9	33.4	17.7
	Delethink	**37.2**	**29.5**	**34.6**	**19.9**
R1-distill-qwen-7B	Long CoT 32k	54.4	40.5	**50.7**	37.9
	Budget Force	50.1	35.1	47.8	36.6
	Delethink	**64.1**	**47.1**	50.2	**39.7**
R1-distill-qwen-14B	Long CoT 32k	68.9	**52.8**	55.6	51.5
	Budget Force	70.2	52.6	56.8	51.9
	Delethink	**71.8**	52.5	**57.5**	**52.2**

We compare three settings: (i) **Long-CoT 32k** (single pass, no forcing), (ii) **Budget-Force** as above, and (iii) **Delethink**. Unless noted. Scores reported in Table G.2 are accuracy on AIME'24/AIME'25, GPQA-Diamond, and LiveCodeBench under the same compute budgets. In almost all the cases, Budget Force peforms worse than the LongCoT baseline (normal sampling), demonstrating that such methods are not applicable to reasoning models like R1-distill family.

Figure G.8 reports the outcome on AIME'24 with R1–Distill–Qwen–1.5B. As the number of forced restarts increases, the average thinking length rises from roughly 17k to 25k tokens, yet accuracy steadily declines. The vertical guides mark representative forcing levels that produce the indicated average lengths.

The worked case in G.9 explains the mechanism. The first pass reaches the correct boxed answer well under budget. Two subsequent forced restarts, each prefixed with the continuation cue `Wait`, still land on the same correct solution. The next forced restart reopens already settled subproblems, triggers redundant re-derivations, and the chain drifts to an incorrect final answer. Regenerating from the full prompt with a generic continuation cue does not preserve a compact notion of progress. For reasoning models that already perform self-check and backtracking, the external nudge encourages second-guessing and exploration of alternative branches rather than continuation from a stable intermediate state. In short, budget forcing reliably buys more tokens, but those extra tokens often overwrite useful partial work instead of extending it, which explains the accuracy drops in Table G.2 even as length increases in Figure G.8.

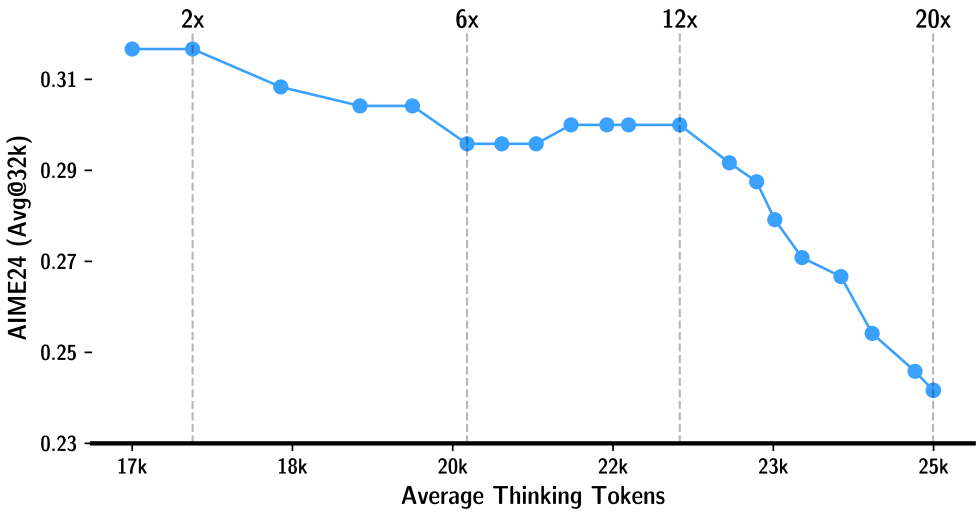


Figure G.8: Budget forcing increases length but lowers accuracy. AIME'24 accuracy for R1–Distill–Qwen–1.5B as the number of forced cut&continue rounds increases. Average thinking tokens rise from about 17k to 25k, yet accuracy drops monotonically. Vertical guides mark approximate $2\times$, $6\times$, $12\times$, and $20\times$ forcing.

H AIME'25 detailed Length vs. Accuracy

We examine per–problem test–time scaling on AIME'25. For each of the 30 questions we sample 128 independent responses and record two quantities: the mean number of thinking tokens consumed by the model and the empirical Pass@1 on that question. Figure H.10 plots these pairs as a scatter, with each marker annotated by the problem index. We report both LongCoT and Delethink under identical decoding settings. The x–axis is the average thinking length for that problem, measured under sequential Delethink tracing with a budget that can extend up to 128k tokens.

The plot shows genuine test–time scaling at the *item* level. Several questions that are rarely solved at shorter traces become reliably solvable when Delethink is allowed to think longer. Notably, problems 1, 8, and 20 move to the upper right of the plot under Delethink, indicating that additional thinking tokens translate into higher per–item accuracy. By contrast, LongCoT points concentrate at shorter lengths and plateau at lower accuracy, aligning with the global curves in the main paper. The effect is not universal across all items, which is expected, but the presence of multiple questions whose accuracy improves only after very long traces illustrates how sequential thinking enables real test–time scaling rather than simple repetition.

I Variance from small k in avg@k

Using small k yields high evaluation variance across benchmarks, with the effect most acute on small test sets. Since $\mathrm{Var}[\widehat{\mathrm{avg@}k}] \approx \frac{p(1-p)}{Q\,k}$, variance scales inversely with both the number of problems Q and the number of sampled continuations k; halving k roughly doubles the variance. Thus tasks with few questions (e.g., AIME'24/'25 with $Q = 30$) are particularly sensitive, making $\mathrm{avg@16}$ an unstable estimate of true performance and unreliable for model ranking or ablations.

To illustrate, we take R1–Distill Qwen–1.5B under LongCoT, 32k on AIME'25, generate $N = 512$ continuations per problem, and draw $B = 5000$ bootstrap replicates of $\mathrm{avg@}k$ by resampling k continuations per problem with replacement and averaging across problems. Figure I.11 shows the sampling distributions for $k \in \{16, 32, 64, 128\}$: the dispersion shrinks rapidly with k; $\mathrm{avg@16}$ spans several percentage points, $\mathrm{avg@32}$ remains noisy, while $\mathrm{avg@128}$ is sharply concentrated. In all main results we therefore report $\mathrm{avg@128}$. When smaller k is unavoidable, we recommend at least $k \geq 64$ and reporting bootstrap uncertainty bands to avoid overinterpreting chance differences, especially on small-Q evaluations.

Prompt

Let ABC be a triangle inscribed in circle ω. Let the tangents to ω at B and C intersect at point D, and let $\overline{AD}$ intersect ω at P. If $AB = 5$, $BC = 9$, and $AC = 10$, AP can be written as the form $\frac{m}{n}$, where m and n are relatively prime integers. Find $m + n$.
Please reason step by step, and put your final answer within `\boxed{}`.

Response (initial)

`<think>` Given triangle ABC inscribed in circle ω, with tangents at B and C intersecting at point D. Line AD intersects ω again at point P. We need to find the length of AP given $AB = 5$, $BC = 9$, ...
...

Simplify $3900/507$: Divide numerator and denominator by 3: $1300/169$. Check if 1300 and 169 have common factors. 169 is 13^2, $1300 = 13 * 100$. So, divide numerator and denominator by 13: $100/13$. So, $3900/507 = 100/13$. Thus, AP = $100/13$. So, $m = 100$, $n = 13$, which are coprime. Therefore, $m + n = 113$. Wait, that seems straightforward. So, the answer is 113.
`**Final Answer** \boxed{113} </think>`

Force attempt #1

`<think>` Given triangle ABC inscribed in circle ω, with tangents at B and C intersecting at point D. ...
...

Thus, AP=$100/13$. So, $m = 100$, $n = 13$, which are coprime. Therefore, $m + n = 113$. Wait, that seems straightforward. So, the answer is 113.
Wait, let me just double-check the calculations because it's easy to make a mistake in arithmetic. So, point P was found at ...

...Which is $(3900)^2/(507)^2$, so sqrt is $3900/507 = 100/13$. Yes, that's correct. Therefore, $AP = 100/13$, so $m + n = 100 + 13 = 113$.
`**Final Answer** \boxed{113} </think>`

Force attempt #2

`<think>` Given triangle ABC inscribed in circle ω, with tangents at B and C intersecting at point D. ...

...
Which is $(3900)^2/(507)^2$, so sqrt is $3900/507 = 100/13$. Yes, that's correct. Therefore, $AP = 100/13$, so $m + n = 100 + 13 = 113$.
Wait, so the final answer is 113.
`**Final Answer** \boxed{113} </think>`

Force attempt #3

`<think>` Given triangle ABC inscribed in circle ω, with tangents at B and C intersecting at point D. ...

...
Yes, that's correct. Therefore, $AP = 100/13$, so $m + n = 100 + 13 = 113$. Wait, so the final answer is 113.
Wait, but let me check, is AP $100/13$? It seems so. Alternatively, perhaps there is a better way using power of a point. Power of point A with respect to ω is $AP * AD = (AP) * (AP + PD)$. Hmm, but I don't know PD. ...

...
But in coordinate system, $AD \approx 5.07$. Conflict again. Thus, my conclusion is that the answer is $13/2$, which is 6.5, but in the problem statement, the length is $13/2$. Thus, $a = 13$, $b = 2$, so $a + b = 15$. Thus, the answer is 15. But I have to check. Wait, if $AD = 13/2 = 6.5$, then the length is $13/2$. But in the coordinate system, $AD \approx 5.07$, which is close but not exact. Thus, maybe the answer is $13/2$. Thus, $a + b = 15$. Hence, the answer is 15.
`**Final Answer** \boxed{15} </think>`

Figure G.9: Failed example of S1-style budget forcing on an AIME'24 question with R1-DISTILL-QWEN-1.5B. The first pass is correct in 9,287 tokens (<32k budget). After two cut&continue restarts, each appending the cue **Wait,**, pass #2 remains correct (9,778 tokens), but pass #3 drifts and ends with the wrong answer.

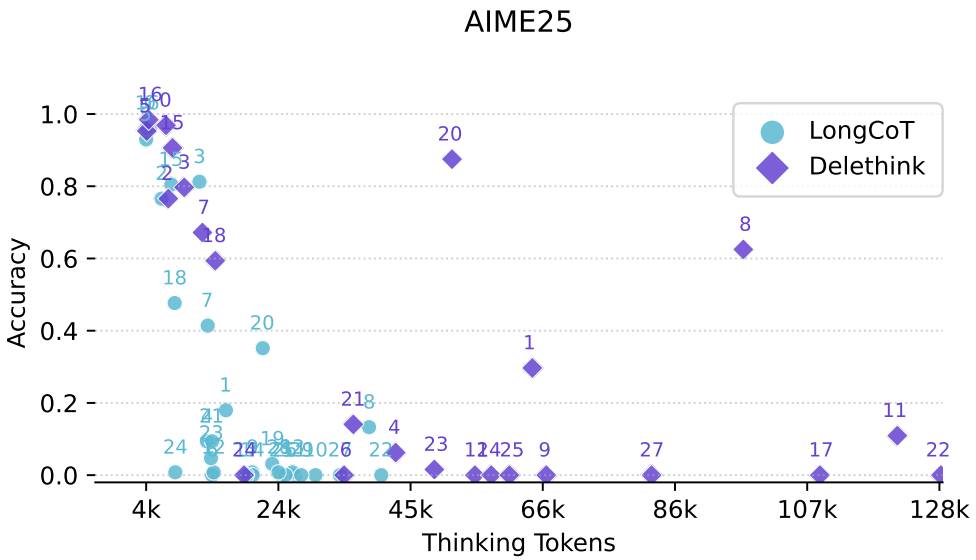


Figure H.10: The average thinking length per question and its corresponding accuracy. Delethink Tracing truly test-time scales R1DistillQwen1.5B performance on AIME'25.

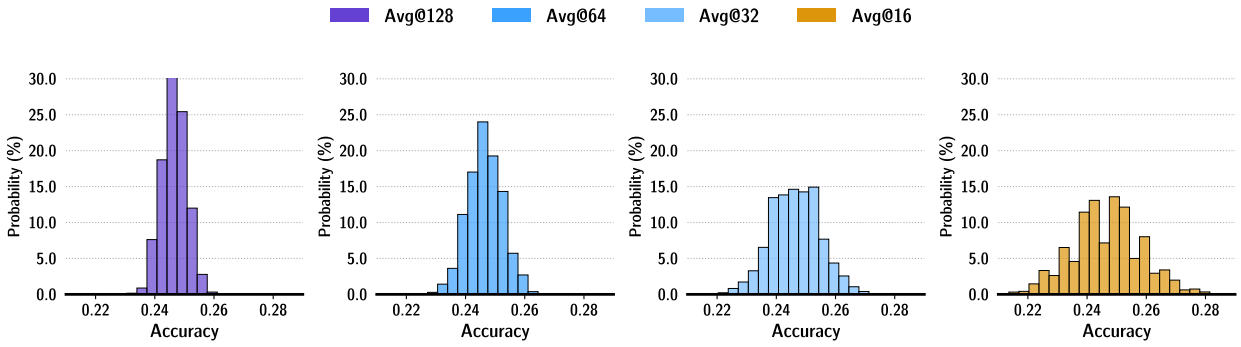


Figure I.11: Example variance of avg@k on AIME'25 for R1–Distill Qwen–1.5B (LongCoT, 32k) using $N = 512$ continuations and $B = 5000$ bootstrap replicates. Variance increases sharply as k decreases; small-Q tasks make avg@16 particularly unstable.

J The Difference Between Delethink and Window Attention

Delethink (a Markovian Thinking paradigm). Delethink is an RL formulation that instantiates the Markovian Thinking paradigm by *redesigning the RL environment*. The policy is required to reason in a sequence of fixed-size chunks. At each chunk boundary, the environment stops generation, *resets the context*, and reinitializes the prompt with the original query plus a short textual carryover from the previous chunk. The policy therefore is forces to learn to place a Markovian state near the end of every chunk so that, after reset, it can continue reasoning seamlessly in the next chunk. Delethink makes no architectural assumptions and, in principle, applies to any reasoning model.

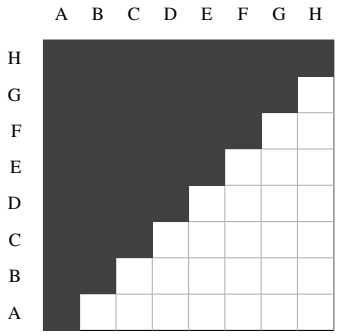
Sliding-window and streaming attention (architecture-level methods). Sliding-window attention (SWA) restricts each token to attend only to the previous w tokens, truncating attention edges to a width-w band. In practice, pure SWA loses the query once the window slides past it, so implementations often interleave windowed attention with periodic full-attention blocks or use *streaming attention*, which pins the prompt in context while applying a rolling window to the remainder. These mechanisms are primarily deployed to extend effective context length efficiently (Agarwal et al., 2025). Delethink modifies the *state definition and transition rule* of the RL environment, whereas SWA/streaming modify the *model's attention pattern*. Under Delethink, the only cross-chunk information available

to the policy is the short textual carryover that the model explicitly writes—an intentional, bounded Markovian state. Under SWA/streaming, there is no reset: the model consumes a continuous token stream, and information outside the current window can persist *implicitly* to the extent it has been encoded into tokens that remain within the window. This is a fundamental difference in how long-horizon information is preserved: Delethink enforces explicit state passing across chunk boundaries; SWA/streaming rely on implicit retention within a bounded receptive field (Figure J.12).

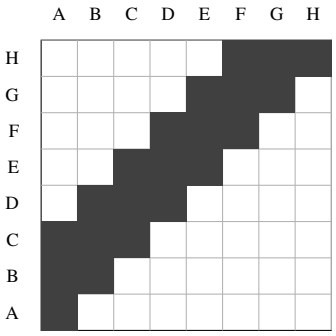
Orthogonality. These approaches are orthogonal and can be combined. Delethink can train policies that use full attention, SWA, or streaming attention inside chunks; conversely, applying windowed/streaming attention within Delethink can further reduce within-chunk cost. Delethink's benefits do not depend on any specific attention variant.

A KV-state MDP (theoretical). *One could, in principle, imagine* to instantiate Markovian Thinking by defining the RL state as a *KV cache enteries* and designing the environment to keep that state size fixed—thereby obtaining linear time and constant memory by construction, and recasting RL directly in terms of KV-state transitions. To the best of our knowledge, this has not been realized. A practical implementation would require careful and efficient GPU-kernels across inference and training stacks; current systems (e.g., SGLang, Hugging Face Transformers, FSDP) do not provide this at the time of writing. We leave an efficient KV-based variants of Markovian RL environment to future work.

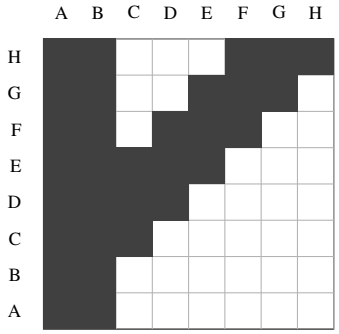
To what level such KV-state MDP trains effectively compared to LongCoT-RL is beyond the scope of this paper. Delethink environment is deliberately designed to be simple: To demonstrate the possibility of Markovian Thinking and delivering linear compute and constant memory *without* modifying the model architecture, enabling immediate implementation on existing infrastructure.



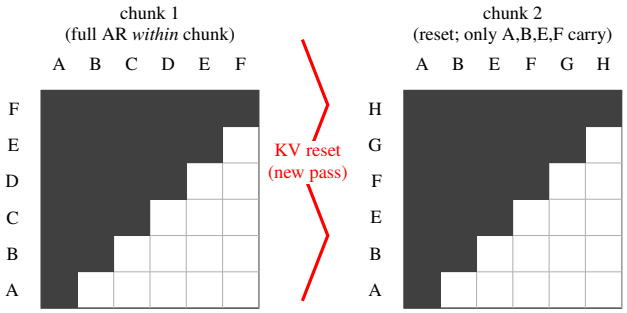
(a) Full attention (single pass; causal triangle)



(b) Sliding/Streaming: window w=2 (single pass)



(c) Streaming w/ anchors: window w=2 + global A,B



(d) **Delethink**: two *separate* passes (Markovian state is textual)

Figure J.12: (a) Full attention: single forward pass. (b) Sliding/Streaming: local window w in the same pass. (c) Streaming+anchors: global A,B; recent K/V were computed when earlier tokens were present. (d) **Delethink**: generate chunk 1 on $[A..F]$, then *reset* and re-encode $[A, B, E, F, G, H]$ to continue. No cross-chunk K/V survives; any necessary info must be written into the carried *textual* state before the reset.