QUANTTHINK: QUANTIZED REASONING MODELS CAN LEARN WHEN TO COMMIT

Anonymous authors
Paper under double-blind review

ABSTRACT

Large reasoning models (LRMs) achieve strong performance through extended chains of thought, but their long outputs impose substantial KV-cache and inference overhead. Although KV-cache quantization effectively reduces memory cost, we find that it can paradoxically prolong reasoning by inducing repeated post-solution verification and delaying answer generation even after a correct solution has already been reached. To address this issue, we introduce QuantThink, which learns when reasoning is sufficient to terminate. QuantThink combines (i) a sufficiency-aware trajectory formulation that identifies checkpoints for verification continuation or reasoning termination, and identifies redundant post-solution verification, and (ii) a parameter-efficient reinforcement learning method Q-GRPO that optimizes termination decisions under quantized inference. Across DeepSeek-R1-Distill-Qwen, QwQ, and Phi-4 models, five reasoning benchmarks, and three KV-cache quantizers—KVQuant, KIVI, and RotateKV—QuantThink improves pass@1 by up to 4.32 percentage points while reducing generated tokens by up to 63.1% compared with the KIVI K2V2 baseline.

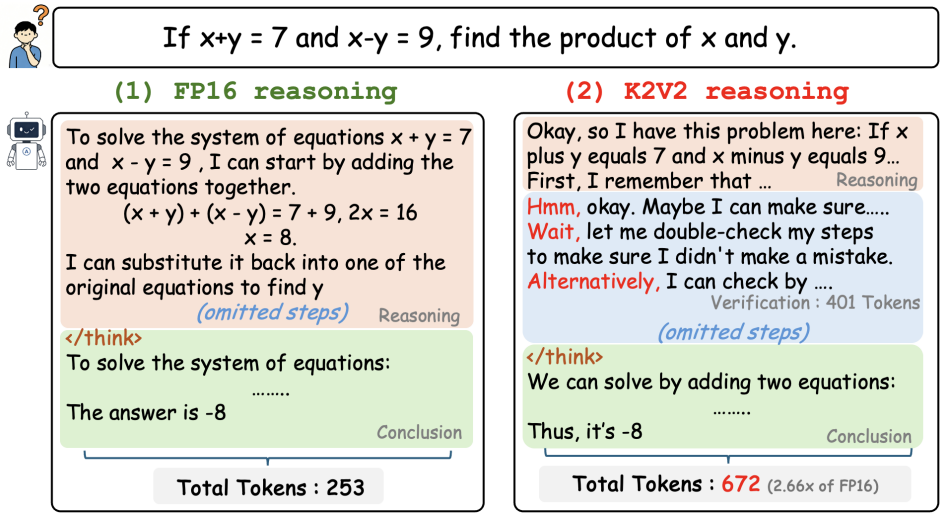

Figure 1: **KV-cache quantization induces CoT inflation.** Relative to FP16 (left), K2V2 (right) increases reasoning from 253 to 672 tokens (2.66×), primarily through post-solution verification. Both yield the same answer, while initial reasoning and conclusion remain similar.

1 INTRODUCTION

Long-form reasoning makes the key–value (KV) cache a central cost of deploying large reasoning models (LRMs). Unlike model weights, the KV cache grows with every generated token and must be retained throughout the decoding process. This cost is especially pronounced for LRMs, which may produce thousands of intermediate tokens before returning an answer (Liu et al., 2025a). Low-bit KV cache quantization reduces the memory required per cached token, enabling longer sequences under same budget. Yet the cost of a request depends on both the KV-cache bit-width and

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

the number of generated tokens. If quantization causes the model to reason substantially longer, part of the intended memory and latency savings may be offset by the increased trajectory length. Consequently, preserving final-answer accuracy alone does not establish that a quantized reasoning model preserves its original efficiency: the same correct answer may require substantially more decoding steps. Evaluating quantization for LRMs therefore requires examining how cache compression changes the computation performed before the model concludes.

Existing KV-cache quantization methods primarily aim to reduce memory usage while minimizing degradation in perplexity and task accuracy. However, we find that KV-cache quantization can also cause substantial CoT inflation (Figure 1), an effect largely overlooked by prior work. As shown in Figure 2 (a), KV-cache quantization substantially inflates CoT length, with the added tokens closely matching the increase in redundant verification while the non-verification portion remains nearly unchanged. Independent analyses of long-CoT reasoning have also identified repeated verification as a source of overthinking (Zhang et al., 2026b). Repeated self-correction is not always beneficial and can even worsen an answer without external feedback (Huang et al., 2024). Efficient-reasoning methods address unnecessary length through prompting, length penalties, early stopping, or reinforcement learning, but generally treat excessive length as an intrinsic property of the reasoning policy. Verification training (Chen et al., 2026) and structured multi-turn decomposition (Zeng et al., 2025) illustrate two ways to reduce that cost without modifying KV-cache precision. Even when a model's hidden states encode intermediate-answer correctness, it may continue reasoning rather than use that signal to stop (Zhang et al., 2025a). We therefore study which part of reasoning changes when KV cache quantization lengthens a trajectory, and whether that change can be corrected without imposing a uniform preference for shorter answers. A fixed token budget cannot distinguish necessary reasoning from repeated checking, and may truncate difficult problems before a correct conclusion is available. Addressing this failure requires more than detecting longer outputs: we must localize the redundant computation, determine when it can be safely reduced, and train the model to make these decisions without additional inference-time control. This leads to two connected questions. **Q1:** How can we identify redundant verification in a quantized reasoning trajectory? **Q2:** How can the model learn to commit at that point without inference-time control or length cheating?

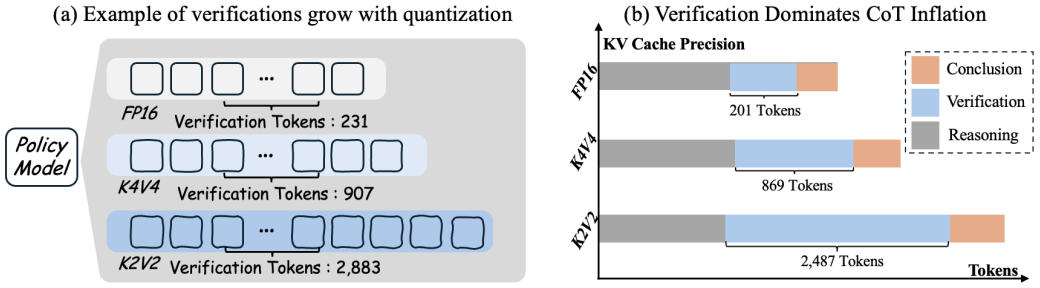

Figure 2: The additional tokens introduced by lower-bit KV-cache quantization closely match the number of redundant verification tokens, while the non-verification portion changes comparatively little.

Building on prefix-level studies of reasoning beyond the first correct answer (Caldarella et al., 2026), we characterize this behavior under KV-cache quantization by decomposing each trajectory into an initial reasoning stage R, implicit answer-ready decision points A, verification episodes V, and a final conclusion C. At each answer-ready point, the model either continues checking the current candidate through another verification episode or enters the conclusion phase. We treat computation between successive answer-ready points as verification and use the earliest correct checkpoint to identify verification that continues after a correct answer is available. As shown in Fig 2(b), we find that the first point at which a correct answer can be produced remains nearly unchanged between FP16 and quantized trajectories, while the mean number of verification episodes increases substantially. In offline K2V2 rollouts, the model repeatedly recomputes or checks an already-derived answer instead of concluding. We call this behavior verification persistence: successive verification

episodes prolong the response without necessarily advancing the solution. This distinction matters because verification can still be useful before a solution is established. The relevant efficiency target is therefore the additional checking that persists after a correct conclusion becomes available, rather than verification as a whole.

In light of this observation, we introduce QuantThink, a quantization-aware, parameter-efficient reinforcement learning method that optimizes the verify-or-commit transition under low-bit KV-cache inference. QuantThink shares S-GRPO's serial grouping of early-exit branches from a single trajectory (Dai et al., 2026a), but addresses a quantization-specific failure mode: repeated verification after a correct answer is already available under low-bit KV-cache inference. QuantThink branches at verification boundaries, where the model can either begin another check or commit to a conclusion. Correct-exit rewards decay with the number of verification episodes after the earliest correct checkpoint, including episodes leading to incorrect exits. Incorrect commitments receive a negative reward, so the objective favors earlier correct commitment while discouraging premature stopping. Together, these signals teach the model to avoid additional verification once a correct conclusion is available. A separate length advantage accounts for the complete response through <EOS>, so shifting computation from the reasoning span into the answer span does not evade the length objective. We optimize these signals on quantized rollouts using LoRA while freezing the pretrained backbone, reducing gradient and optimizer-state storage relative to full-parameter updates. At inference, the adapted model uses ordinary quantized decoding without an external stopping controller.

We evaluate QuantThink on four reasoning models from the DeepSeek-Distill, QwQ, and Phi-4 families across GSM8K, AIME 2024, AMC 2023, MATH-500, and LiveCodeBench, using KVQuant, KIVI, and RotateKV. Relative to quantized baselines, QuantThink improves macro-average pass@1 by a **4.32 percentage points** on 7B and reduces generated tokens by **63.1%** on 1.5B. We further evaluate latency, verification count, premature commitment, rescued trajectories, and transfer across bit widths and quantizers. These analyses validate verification persistence as the source of inflation and show that QuantThink mitigates it without inference-time control.

2 Related Work

Low-bit KV cache quantization. KV cache quantization reduces the memory and bandwidth costs of long-context generation. Representative methods improve low-bit fidelity through non-uniform quantization and outlier retention (KVQuant) (Hooper et al., 2024), outlier-aware rotations and attention-sink protection (RotateKV) (Su et al., 2025), or asymmetric per-channel and per-token quantization (KIVI) (Liu et al., 2024). These methods focus primarily on perplexity, accuracy, and system efficiency. In contrast, we study how quantization changes reasoning trajectories, particularly whether a model exits redundant verification and commits to its answer. More recent quantizers target long-CoT inference through progressive mixed-precision allocation (Liu et al., 2026a) or query-aware channel protection (Zhang et al., 2026a); our focus is the resulting reasoning behavior rather than the cache representation itself. Attention-aware rotations have also been evaluated for low-bit KV quantization of reasoning models (Zhou et al., 2026). Separate cache-compression work assigns precision by thought importance (Ramachandran et al., 2026), compacts reasoning blocks (Liu et al., 2026b), or identifies cache entries most useful to the final answer (Wang et al., 2026b). These methods change what is stored; we instead train when the quantized model should stop verifying.

Early stopping for efficient reasoning. Early-stopping methods reduce inference cost by terminating a chain of thought once further reasoning is unlikely to improve the answer. GRPO learns from group-relative outcome rewards but does not explicitly model commitment timing (Shao et al., 2024). JET constructs truncated rollouts and uses reinforcement learning to train correct early commitment (Han et al., 2026). S-GRPO samples exit positions within one trajectory and assigns decaying rewards to correct exits in their temporal order (Dai et al., 2026a). Under KV-cache quantization, trajectories often reach the 32,768-token limit through repeated verification. S-GRPO's uniformly sampled exits can therefore fall deep within this redundant tail, retaining unnecessary computation. QuantThink instead branches at verification boundaries and rewards correct commitment near the first correct checkpoint, while accounting for the full response length; Table 11 isolates these contributions relative to S-GRPO. Other training approaches shorten reasoning through length-harmonizing fine-tuning (Luo et al., 2026), adaptive thinking-mode selection (Zhang et al.,

2025b), or step-aware group rewards (Li et al., 2026). Length-focused RL can instead target the shortest correct sample (Yi et al., 2026), while redundancy-aware objectives suppress low-value reasoning (Cai et al., 2026) or distinguish useful from repeated verification (Dai et al., 2026b). DEER instead operates without training, probing trial answers at reasoning transitions and exiting when token-level confidence exceeds a fixed threshold (Yang et al., 2026). Training-free methods instead mine reasoning-completion patterns (Wei et al., 2025), assess thought sufficiency (Xiang et al., 2026), check semantic convergence (Min et al., 2026), or detect redundant reflection (Sun et al., 2026).

3 PRELIMINARIES

3.1 KV-CACHE QUANTIZATION

During decoding, each Transformer layer caches previous key and value tensors; the cache grows with sequence length. Low-bit quantization stores each tensor $X \in \{K, V\}$ using b-bit integers, with a standard affine reconstruction:

$$\widehat{X} = s_X \left[\mathrm{clip}\big(\mathrm{round}(X/s_X) + z_X, 0, 2^b - 1\big) - z_X \right], \qquad X \in \{K, V\}, \tag{1}$$

Here s_X and z_X are the scale and zero point, chosen per token, channel, or group. Approximation in $\widehat{K}$ and $\widehat{V}$ changes attention outputs and thus subsequent token probabilities, without changing model weights.

3.2 RL-BASED EARLY EXIT

RL-based early exit trains a model to answer once its reasoning is sufficient. For trajectory y and candidate boundary $t \in \mathcal{T}(x)$, it appends </think> to $y_{\le t}$ and decodes an answer $\mathcal{E}(y_{\le t})$. Prefix-truncation analyses use similar forced answers to assess sufficiency (Wang et al., 2026a). The earliest correct exit is

$$t^*(x) = \min \left\{ t \in \mathcal{T}(x) : \mathrm{Acc}(\mathcal{E}(y_{\le t}), x) = 1 \right\}. \tag{2}$$

Here $\mathrm{Acc}(\cdot, x)$ checks answer correctness. Training rewards correct earlier exits and penalizes incorrect ones, so the model learns to stop without an inference-time controller. Position- or length-based ranking, however, does not target verification episodes prolonged by KV-cache quantization.

4 METHOD

QuantThink is a quantization-aware early-stopping framework with two components: trajectory-level redundancy identification and commitment optimization. For **Q1**, we decompose quantized trajectories into reasoning (R), answer-ready checkpoints (A), verification (V), and conclusion (C), then probe dictionary-identified boundaries to locate the earliest correct commitment point A_c and identify verification that continues beyond it. For **Q2**, we organize early-exit branches of the same trajectory into a commitment group and apply parameter-efficient reinforcement learning to optimize correct commitment and complete-response length through <EOS>.

4.1 COMMITMENT-AWARE TRAJECTORY SEGMENTATION

To answer **Q1**, we formalize a quantized reasoning trajectory using four functional components: reasoning (R), answer-ready checkpoints (A), verification (V), and conclusion (C). R contains the main derivation toward a candidate solution. An A is an instantaneous decision boundary rather than a text span: the model may either commit or initiate another verification episode. A continued branch forms a V segment that re-examines, recomputes, or validates the current solution and ends at the next checkpoint. Finally, C is the conclusion span that begins with </think>, contains the decoded final answer, and ends with <EOS>. Among the checkpoints, A_0 is the first candidate commitment point, whereas A_c is the earliest checkpoint whose simulated commitment produces a correct final answer. Thus, a trajectory with repeated verification follows $R \to A_0 \to V_1 \to \cdots \to A_c \to \cdots \to A_m \to C$. When the first candidate point already supports a correct commitment, A_0 and A_c coincide.

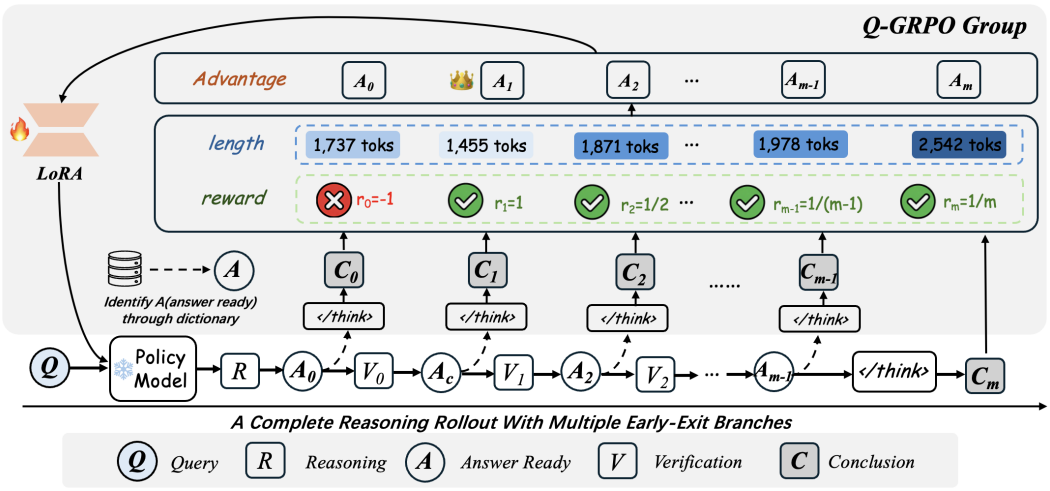

Figure 3: Overview of QuantThink: an offline verification dictionary identifies the A checkpoints, whose early-exit branches form a commitment group for computing correctness- and length-aware advantages that update only the LoRA adapters while the pretrained policy backbone remains frozen.

To construct the segmentation rule, we first collect rollouts that exhibit repeated verification and extract recurring verification cues, such as *Wait* and *Alternatively*, into a fixed offline dictionary $\mathcal{D}_V$. We then split each saved rollout into sentences $y = (s_1, \ldots, s_n)$ and use dictionary matching to locate verification segments. Let k_V be the first sentence whose opening matches $\mathcal{D}_V$, and let k_C be the position of `</think>` that begins the conclusion span. If no verification cue occurs, we set $k_V = k_C$, yielding an empty V span. In this case, $A_c = A_0$ only if the simulated commitment at A_0 produces a correct answer; otherwise, no correct commitment point is identified. We then define

$$\mathcal{D}_V = \{\textit{Wait}, \textit{Alternatively}, \ldots\}, \quad R = s_{1:k_V-1}, \quad V = s_{k_V:k_C-1}, \quad C = s_{k_C:n}. \tag{3}$$

4.2 Quantization-Aware Group Relative Policy Optimization

To answer **Q2**, we train **Q-GRPO** on boundary-specific early-exit branches. At each A_j, we retain the quantized reasoning prefix, append `</think>`, and decode the final answer $\hat{a}_j$ through `<EOS>`. This produces one candidate conclusion $C_j = (\texttt{</think>}, \hat{a}_j, \texttt{<EOS>})$ at each checkpoint. The branches test whether the model can answer after different numbers of verification episodes. Experiments on AIME under KIVI K2V2 KV-cache quantization show that, for DeepSeek-R1-Distill-Qwen-7B, committing at A_c achieves the same accuracy as later commitment points while requiring fewer generated tokens, whereas committing before A_c leads to lower accuracy Fig 15.

During group construction, the rollout policy $\pi_{\theta_{\text{old}}}$ is frozen and generates each branch conclusion. The externally inserted `</think>` is included in the policy loss as an intervention-based stopping surrogate, not as an action sampled from the old policy. A positive branch advantage thus increases its probability at that checkpoint, alongside the generated conclusion tokens. The shared prefix $y_{\le A_j}$ provides conditioning context but is masked from the policy loss.

Verification-to-commit reward. Let c_j indicate whether the answer in C_j is correct, and let c be the index of the earliest correct commitment point A_c. Since each successive checkpoint adds one verification episode, a correct commitment at A_j retains $j - c$ redundant verification episodes. The verification-to-commit reward is

$$r_j^{\text{vc}} = \begin{cases} -1, & c_j = 0 \\ \dfrac{1}{1+j-c}, & c_j = 1, \end{cases} \tag{4}$$

A commitment before A_c produces an incorrect answer in the sampled group and receives -1. Commitment at A_c yields the earliest correct answer without any subsequent verification, giving

the maximum reward of 1. A correct commitment after A_c remains positively rewarded, with rewards decaying as $1/2, 1/3, \ldots$ for successive redundant verification transitions. The group-relative transition advantage is

$$A_j^{\mathrm{vc}} = r_j^{\mathrm{vc}} - \frac{1}{m+1}\sum_{k=0}^{m} r_k^{\mathrm{vc}}. \tag{5}$$

End-to-end length advantage. Group-relative policy optimization can itself exhibit response-length biases (Liu et al., 2025b). Previous methods that optimize only CoT length are vulnerable to reward hacking: the model can emit `</think>` at the beginning and shift its reasoning into the subsequent final-answer span. Q-GRPO makes such shifting less attractive by measuring the complete generated response:

$$\ell_j = |y_{\le A_j}| + |C_j|, \tag{6}$$

where C_j is the complete conclusion decoded after committing at A_j; prompt tokens are excluded. For the correct commitments $\mathcal{C}(x) = \{j : c_j = 1\}$, let $\ell_{\min}$ and $\ell_{\max}$ be their minimum and maximum end-to-end lengths. We define the length advantage as

$$A_j^{\mathrm{len}} = \begin{cases} \dfrac{\ell_{\max} - \ell_j}{\ell_{\max} - \ell_{\min}}, & j \in \mathcal{C}(x) \text{ and } \ell_{\max} > \ell_{\min}, \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

Q-GRPO objective. Q-GRPO combines transition quality with complete-response efficiency:

$$A_j^{\text{Q-GRPO}} = A_j^{\mathrm{vc}} + \lambda A_j^{\mathrm{len}}, \tag{8}$$

We broadcast the sequence-level advantage to the optimized tokens in C_j, including `</think>`, and optimize the clipped policy objective:

$$\mathcal{J}(\theta) = \mathbb{E}\left[\frac{1}{m+1}\sum_{j=0}^{m}\frac{1}{|C_j|}\sum_{t=1}^{|C_j|}\min\Big(\rho_{j,t}A_j^{\text{Q-GRPO}}, \mathrm{clip}(\rho_{j,t}, 1-\eta, 1+\eta)A_j^{\text{Q-GRPO}}\Big)\right], \tag{9}$$

The token-level probability ratio is

$$\rho_{j,t} = \frac{\pi_\theta(C_{j,t} \mid x, y_{\le A_j}, C_{j,<t})}{\pi_{\theta_{\mathrm{old}}}(C_{j,t} \mid x, y_{\le A_j}, C_{j,<t})}, \tag{10}$$

where η is the clipping range. This training-only supervision leaves inference unchanged: the learned policy performs standard quantized decoding without an external stopping module or an additional decoding pass.

5 Experiments

5.1 Experimental Setup

Training data and optimization. We train on decontaminated DeepMath-103K. All RL baselines and QuantThink use online KIVI K2V2 rollouts, where CoT inflation is more pronounced, and are evaluated under KIVI K2V2 in the main comparison. K4V4 tests transfer across precisions. Reference solutions supply final answers for correctness checks. We update LoRA adapters while keeping the pretrained backbone frozen.

Models. We evaluate QuantThink on four open-weight reasoning models: DeepSeek-R1-Distill-Qwen-1.5B, DeepSeek-R1-Distill-Qwen-7B (Guo et al., 2025), Phi-4-reasoning-14B (Abdin et al., 2025), and QwQ-32B (Team, 2025).

Benchmarks. We use GSM8K (Cobbe et al., 2021), AMC 2023, AIME 2024, MATH-500 (Lightman et al., 2023) from the MATH benchmark (Hendrycks et al., 2021), and the August 2024–January 2025 slice of LiveCodeBench (Jain et al., 2024). Generalization across KV-cache quantizers is evaluated with KVQuant (Hooper et al., 2024), KIVI (Liu et al., 2024), and RotateKV (Su et al., 2025).

Table 1: FP16 reference and generalization across KV-cache quantizers and bit-widths on MATH-500 using DeepSeek-R1-Distill-Qwen-7B. KVQuant retains the largest 1% of outliers in FP16.

	Base		GRPO		DEER		JET		S-GRPO		QuantThink	
Configuration	Pass@1	Tokens	Pass@1	Tokens	Pass@1	Tokens	Pass@1	Tokens	Pass@1	Tokens	Pass@1	Tokens
Full Precision												
FP16	89.8	5,661	90.6	5,317	91.0	2,431	91.2	2,591	92.4	2,452	91.6	2,722
K4V4												
KVQuant	85.8	5,812	89.2	5,117	89.8	2,552	90.0	2,241	89.8	2,389	**90.0**	**2,204**
RotateKV	85.0	5,601	88.4	4,980	86.6	3,021	87.2	2,947	88.6	2,642	**89.2**	**2,403**
KIVI	86.2	5,789	86.8	5,076	88.4	2,601	87.8	2,362	90.0	2,365	**90.4**	**2,287**
K2V2												
KVQuant	78.6	10,957	82.4	7,884	81.8	4,757	80.8	4,635	81.4	4,891	**82.2**	**4,452**
RotateKV	54.8	13,829	58.2	9,582	50.4	6,376	55.2	5,729	52.2	5,528	**57.2**	**5,225**
KIVI	84.6	7,846	85.4	7,293	85.2	4,527	84.0	5,143	86.2	3,531	**88.8**	**3,226**

Baselines. We compare QUANTTHINK with the full-precision model, the unmodified quantized model, standard GRPO (Shao et al., 2024), DEER (Yang et al., 2026), S-GRPO (Dai et al., 2026a), and Just-Enough Thinking (JET) (Han et al., 2026).

Metrics. We report pass@1, average generated tokens, end-to-end latency, and peak GPU memory under three NVIDIA RTX 3090 GPUs, using matched batch size and decoding settings, for K2V2 and K4V4. Peak memory (GB) is the maximum total allocated GPU memory across the three devices during generation, averaged over evaluated batches. N denotes the number of questions in dataset.

$$\mathrm{RVC} = \frac{1}{N}\sum_i |\mathcal{V}_{\mathrm{red}}^{(i)}|, \quad \mathcal{V}_{\mathrm{red}}^{(i)} = \{V_j^{(i)} : j > c_i\}, \quad \text{CoT-C} = \frac{1}{N}\sum_i \mathbf{1}[g(y_i^{\mathrm{think}}) = 1]. \tag{11}$$

Here, c_i is the index of the earliest correct checkpoint A_c in response i, so RVC counts verification episodes after that checkpoint. If no correct checkpoint exists, the question contributes zero to RVC, which is therefore interpreted alongside pass@1 rather than in isolation. The blinded LLM judge g identifies CoTs ending at a complete semantic boundary. Lower RVC and higher CoT-C are better. We track RVC because redundant verification drives much of the quantization-induced token inflation. CoT-C guards against apparent token savings from emitting `</think>` immediately and skipping reasoning. Figure 14 illustrates this failure mode.

5.2 EXPERIMENTAL RESULTS

We assess three outcomes: the pass@1–efficiency trade-off under aggressive quantization, transfer across quantizers and bit widths, and whether shorter responses reflect less redundant verification without truncated reasoning. We therefore report both task outcomes and behavioral diagnostics.

Transfer across quantization settings. Table 1 reports MATH-500 results for DeepSeek-R1-Distill-Qwen-7B under three quantizers and two bit widths, with FP16 as a reference. Although training uses KIVI K2V2, QuantThink improves pass@1 and reduces tokens relative to the corresponding base in all six quantized settings. The K2V2 token savings range from 58.9% to 62.2% across KIVI, KVQuant, and RotateKV. Thus, the learned stopping behavior also transfers to quantization mechanisms absent from training.

Overall pass@1–efficiency trade-off. Table 2 compares five benchmarks under KIVI K2V2, where reasoning inflation is pronounced. FP16 shows the pre-quantization reference, and the quantized base shows the cost each method can recover. "Overall" is the macro-average across benchmarks. This gives each benchmark equal weight despite their different numbers of questions.

QuantThink improves overall pass@1 by 4.30, 4.32, and 2.90 points on the 1.5B, 7B, and 14B models while cutting generated tokens by 63.1%, 55.1%, and 46.9%, respectively, relative to KIVI K2V2. Against the strongest early-exit baseline, S-GRPO, the 7B model gains 2.34 points and uses 17.2% fewer tokens. The gain is therefore not solely recovery from the untrained quantized base. On 1.5B, QuantThink exceeds S-GRPO by 1.34 points while using 29.0% fewer tokens. The margin narrows on 14B to 0.92 points and 3.3% fewer tokens, indicating that the benefit varies with model scale. Standard GRPO even lengthens the quantized 7B and 14B responses, illustrating why optimizing answer outcomes alone need not resolve CoT inflation.

Table 2: Main results for FP16 and KIVI K2V2. Except for the FP16 reference, all methods are evaluated under KIVI K2V2. Colored values in the Overall columns show changes relative to KIVI K2V2; red denotes an increase and blue a decrease in the corresponding metric.

	GSM8K		AIME 2024		LiveCodeBench		MATH-500		AMC 2023		Overall	
Method	Pass@1	Tokens	Pass@1	Tokens	Pass@1	Tokens	Pass@1	Tokens	Pass@1	Tokens	Pass@1	Tokens
DeepSeek-R1-Distill-Qwen-1.5B												
FP16 Base	87.0	1,587	30.0	11,046	16.8	17,782	79.6	3,617	62.5	10,824	55.18	8,971
KIVI K2V2	84.5	2,186	23.3	15,456	12.7	24,033	74.8	5,346	57.5	14,728	50.56	12,350
GRPO	85.2	2,087	23.3	16,873	13.4	24,681	75.6	4,873	60.0	13,619	51.50+0.94	12,427+0.6%
DEER	84.1	1,293	20.0	10,081	13.1	16,067	76.2	3,714	57.5	7,647	50.18-0.38	7,760-37.2%
JET	85.5	1,061	16.7	10,943	14.0	15,487	76.8	2,806	60.0	7,531	50.60+0.04	7,566-38.7%
S-GRPO	86.0	1,012	26.7	8,877	14.8	13,733	77.6	2,259	62.5	6,223	53.52+2.96	6,421-48.0%
QuantThink	87.7	1,047	26.7	6,454	16.1	8,587	78.8	2,034	65.0	4,663	**54.86**+4.30	**4,557**-63.1%
DeepSeek-R1-Distill-Qwen-7B												
FP16 Base	92.4	1,833	56.7	13,232	37.6	15,232	89.8	5,661	77.5	9,293	70.80	9,050
KIVI K2V2	89.1	2,454	50.0	18,106	34.3	22,523	84.6	7,846	72.5	12,991	66.10	12,784
GRPO	90.0	2,344	53.3	20,437	35.1	23,781	85.4	7,293	70.0	13,647	66.76+0.66	13,500+5.6%
DEER	88.7	1,244	46.7	11,873	34.8	12,031	85.2	4,527	75.0	8,383	66.08-0.02	7,612-40.5%
JET	90.3	1,396	43.3	12,608	35.0	14,852	84.0	5,143	75.0	7,089	65.52-0.58	8,218-35.7%
S-GRPO	91.2	1,206	50.0	10,312	35.5	13,574	86.2	3,531	77.5	6,033	68.08+1.98	6,931-45.8%
QuantThink	92.0	1,142	53.3	8,842	38.0	10,688	88.8	3,226	80.0	4,792	**70.42**+4.32	**5,738**-55.1%
Phi-4-reasoning-14B												
FP16 Base	95.0	2,129	63.3	11,099	53.8	10,872	92.4	3,844	92.5	5,527	79.40	6,694
KIVI K2V2	93.2	2,538	60.0	14,483	51.9	12,801	90.2	5,083	90.0	6,811	77.06	8,343
GRPO	93.8	2,391	60.0	16,327	52.5	13,589	91.0	4,627	87.5	7,123	76.96-0.10	8,811+5.6%
DEER	94.0	982	63.3	10,037	52.2	10,143	89.8	2,453	92.5	4,049	78.36+1.30	5,533-33.7%
JET	94.2	1,198	56.7	10,379	52.8	8,637	91.2	2,754	92.5	4,187	77.48+0.42	5,431-34.9%
S-GRPO	94.8	1,024	63.3	8,648	53.0	7,886	91.6	2,071	92.5	3,281	79.04+1.98	4,582-45.1%
QuantThink	96.1	1,078	66.7	8,656	54.0	6,822	93.0	2,169	90.0	3,429	**79.96**+2.90	**4,431**-46.9%

Where the token savings come from. CoT-C guards against a shortcut observed in early-exit trajectories: a policy can emit `</think>` immediately, skip the visible derivation, and rely on the model's direct-answer ability in the conclusion. This saves tokens but can lower pass@1 rather than teach a better stopping point. Token counts alone therefore cannot distinguish genuine removal of redundant verification from bypassing reasoning. Table 3 reports pass@1 and generated tokens on MATH-500 alongside behavioral and inference-cost measures. A successful method should stop at a complete semantic boundary, remove unnecessary checks, and preserve the answer.

Table 3: Behavioral and inference-cost analysis of DeepSeek-R1-Distill-Qwen-7B on MATH-500. Except for FP16 Base, all methods use KIVI K2V2. Pass@1 and Tokens match Table 2. RVC is averaged per question as in Eq. equation 11; Peak Mem is peak total allocation across three RTX 3090 GPUs at batch size 8 (32,768-token cap). Latency is mean per-question end-to-end wall-clock time from prompt to `<EOS>`.

Method	Pass@1 ↑	RVC ↓	CoT-C ↑	Peak Mem (GB) ↓	Tokens ↓	Latency (s) ↓
FP16 Base	89.8	1.65	1.0	32.3	5,661	32.8
KIVI K2V2	84.6	4.70	0.92	20.1	7,846	20.9
GRPO	85.4	4.20	0.94	20.3	7,293	20.4
DEER	85.2	1.21	0.48	19.1	4,527	18.3
JET	84.0	1.72	0.78	19.7	5,143	14.6
S-GRPO	86.2	1.03	0.68	18.8	3,531	11.8
QuantThink	88.8	0.78	0.98	18.2	3,226	10.2

On MATH-500, QuantThink raises pass@1 from 84.6 to 88.8 while reducing mean generated tokens from 7,846 to 3,226 relative to KIVI K2V2. It also exceeds S-GRPO by 2.6 points with 8.6% fewer tokens. RVC falls from 4.70 to 0.78, CoT-C rises from 0.92 to 0.98, peak memory falls from 20.1 to 18.2 GB, and latency from 20.9 to 10.2 seconds.

5.3 Ablation Study

Figure 4 first shows how stopping behavior changes during optimization: the stop position approaches the earliest correct checkpoint A_c as end-to-end response length falls, while pass@1 stays high. We then ablate the two reward components on DeepSeek-R1-Distill-Qwen-7B under KIVI K2V2. Removing A^{vc} or A^{len} isolates the contribution of commitment timing and complete-response length, respectively.

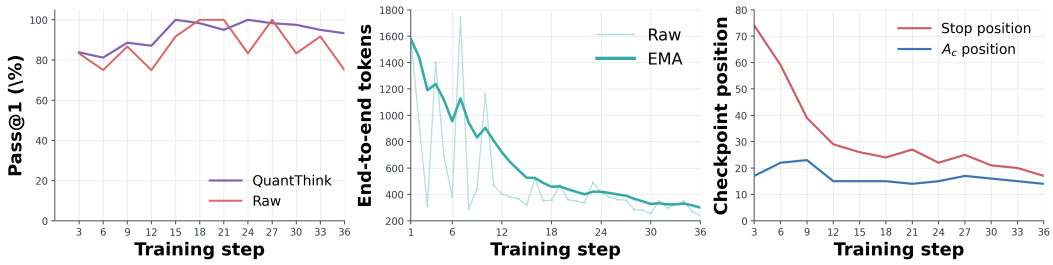

Figure 4: Training dynamics of QuantThink under KIVI K2V2. From left to right, we report Pass@1, mean end-to-end generated tokens (CoT and conclusion), and the learned stopping checkpoint against the earliest correct checkpoint A_c. Pass@1 and checkpoint positions are evaluated every three optimization steps; the length panel shows both raw measurements and their EMA trend.

Table 4: Ablation of the QuantThink reward components on DeepSeek-R1-Distill-Qwen-7B under KIVI K2V2.

	AIME 2024		LiveCodeBench		MATH-500		AMC 2023		Overall	
Method	Pass@1	Tokens	Pass@1	Tokens	Pass@1	Tokens	Pass@1	Tokens	Pass@1	Tokens
DeepSeek-R1-Distill-Qwen-7B										
Reward components										
QuantThink	53.3	8,842	38.0	10,688	88.8	3,226	80.0	4,792	65.03	6,887
w/o. A^{vc}	46.7	11,947	33.0	13,420	85.2	3,909	75.0	6,376	59.98	8,913
w/o. A^{len}	53.3	9,601	37.3	11,459	87.6	3,721	77.5	5,229	63.93	7,503

Hyperparameter sensitivity. Tables 5 and 6 study the length weight λ and clipping coefficient η, respectively. The default setting ($\lambda = 0.1$, $\eta = 0.2$) matches the MATH-500 QuantThink result in Table 3. Appendix Table 10 separately compares dictionary-based boundaries with random, uniform, and BERT-based alternatives under the same number of checkpoints per trajectory.

Table 5: Sensitivity to λ on DeepSeek-R1-Distill-Qwen-7B (KIVI K2V2) with MATH-500

Setting	Pass@1 ↑	Tokens ↓	RVC ↓
Base (no RL)	84.6	7,846	4.70
$\lambda = 0$	87.6	3,721	1.02
$\lambda = 0.05$	88.2	3,562	0.92
$\lambda = \mathbf{0.1}$	**88.8**	3,226	0.78
$\lambda = 0.15$	88.0	3,136	0.75
$\lambda = 0.2$	87.8	2,929	0.67

Table 6: Sensitivity to η on DeepSeek-R1-Distill-Qwen-7B (KIVI K2V2) with MATH-500

Setting	Pass@1 ↑	Tokens ↓	RVC ↓
$\eta = 0.1$	88.6	4,132	0.92
$\eta = \mathbf{0.2}$	**88.8**	3,226	0.78
$\eta = 0.3$	88.6	3,061	0.69
$\eta = 0.5$	87.4	3,980	0.89
$\eta = 0.7$	87.2	3,421	0.80

6 Conclusion

Our results show that much of the additional CoT under low-bit KV-cache quantization appears as verification after a correct answer can already be produced. QuantThink targets this behavior at complete verification boundaries: Q-GRPO compares early-exit branches of one quantized trajectory and rewards correct, timely commitment while measuring length through the final <EOS>. It therefore learns a stopping policy without an inference-time controller or a reward based only on CoT shortness. Across the evaluated models, benchmarks, bit widths, and quantizers, QuantThink reduces generated tokens while maintaining or improving pass@1. On MATH-500, it also reduces redundant verification while retaining a high rate of semantically complete CoTs. The earliest correct checkpoint is identified from sampled conclusions, however, and the current boundary dictionary is rule-based. Testing the stability of these checkpoints and extending boundary detection beyond math-focused reasoning remain important next steps.

Ethics Statement

This work studies inference efficiency in open-weight reasoning models and involves no human subjects or animal experiments. Training uses the decontaminated DeepMath-103K dataset; evaluation uses GSM8K, AMC 2023, AIME 2024, MATH-500, and LiveCodeBench. We use these datasets for mathematical and coding tasks and do not intentionally process personal information. Shorter outputs are not inherently better: premature stopping can reduce accuracy or hide reasoning in the conclusion. We therefore report answer correctness and CoT completeness alongside token and latency savings.

Reproducibility Statement

Section 4 specifies trajectory segmentation, early-exit branch construction, rewards, and the Q-GRPO objective; Algorithm 1 summarizes the training procedure. Section 5 reports the models, datasets, KIVI K2V2 training setting, evaluation metrics, and hardware. The appendix includes the verification-cue visualization, segmentation comparisons, reward example, and repeated-run results. These details support independent implementation and evaluation. Repo and more detailed information at `https://anonymous.4open.science/r/QuantThink-0DB2/`

AI use statement

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

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

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

References

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

Ruichu Cai, Haopeng Du, Qingwen Lin, Yutong Chen, Zijian Li, and Boyan Xu. Entra: Entropy-based redundancy avoidance in large language model reasoning. *arXiv preprint arXiv:2601.07123*, 2026.

Simone Caldarella, Davide Talon, Rahaf Aljundi, Elisa Ricci, and Massimiliano Mancini. Thinking past the answer: Evaluating harmful overthinking in large reasoning models. *arXiv preprint arXiv:2606.02835*, 2026.

Zigeng Chen, Xinyin Ma, Gongfan Fang, Ruonan Yu, and Xinchao Wang. Verithinker: Learning to verify makes reasoning model efficient. *Advances in Neural Information Processing Systems*, 38: 130202–130227, 2026.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Muzhi Dai, Chenxu Yang, and Qingyi Si. S-grpo: Early exit via reinforcement learning in reasoning models. *Advances in Neural Information Processing Systems*, 38:48178–48204, 2026a.

Xinbang Dai, Zheyu Xin, Huikang Hu, Lin Ren, Rihui Jin, Guohui Xiao, Guilin Qi, Kuicai Dong, Zhaocheng Du, and Yuyang Zhang. Evothink: Evolving thinking in large reasoning models via self-pruning and aha-moment preference optimization. *arXiv preprint arXiv:2607.19962*, 2026b.

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

Jinyi Han, Ying Huang, Ying Liao, Haiquan Zhao, Zishang Jiang, Xinyi Wang, Xikun Lu, Guanghao Zhou, Sihang Jiang, Jiaqing Liang, et al. Your models have thought enough: Training large reasoning models to stop overthinking. In *International Conference on Learning Representations*, volume 2026, pp. 141341–141365, 2026.

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

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun S Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *Advances in Neural Information Processing Systems*, 37:1270–1303, 2024.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *International conference on learning representations*, volume 2024, pp. 32808–32824, 2024.

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, 2024. URL https://arxiv.org/abs/2403.07974.

Weijie Li, Jin Wang, Liang-Chih Yu, and Xuejie Zhang. Step-grpo: Enhancing reasoning quality and efficiency via structured prm-based reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 31734–31742, 2026.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step, 2023. URL https://arxiv.org/abs/2305.20050.

Minghui Liu, Aadi Palnitkar, Tahseen Rabbani, Hyunwoo Jae, Kyle Rui Sang, Dixi Yao, Shayan Shabihi, Fuheng Zhao, Tian Li, Ce Zhang, et al. Hold onto that thought: Assessing kv cache compression on reasoning. *arXiv preprint arXiv:2512.12008*, 2025a.

Tengxuan Liu, Shiyao Li, Jiayi Yang, Tianchen Zhao, Feng Zhou, Xiaohui Song, Guohao Dai, Shengen Yan, Huazhong Yang, and Yu Wang. Pm-kvq: Progressive mixed-precision kv cache quantization for long-cot llms. In *International Conference on Learning Representations*, volume 2026, pp. 107881–107900, 2026a.

Yang Liu, Bin Chong, Chongyang Zhang, Hao Zheng, Jiayu Liang, and Xu Kefu. Thought-aware kv cache compaction for reasoning via adaptive attention matching. *arXiv preprint arXiv:2608.12331*, 2026b.

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

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. *arXiv preprint arXiv:2402.02750*, 2024.

Sanae Lotfi, Polina Kirichenko, Steven Li, and Zechun Liu. Quantized reasoning models think they need to think longer, but they do not, 2026. URL https://arxiv.org/abs/2606.00206.

Haotian Luo, Haiying He, Yibo Wang, Shiwei Liu, Wei Li, Xiaochun Cao, Dacheng Tao, Naiqiang Tan, and Li Shen. O1-pruner: Length-harmonizing fine-tuning for o1-like reasoning pruning. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 14242–14257, 2026.

Dehai Min, Giovanni Vaccarino, Huiyi Chen, Yongliang Wu, Gal Yona, and Lu Cheng. Stop when reasoning converges: Semantic-preserving early exit for reasoning models. *arXiv preprint arXiv:2605.17672*, 2026.

Akshat Ramachandran, Marina Neseem, Charbel Sakr, Rangharajan Venkatesan, Brucek Khailany, and Tushar Krishna. Thinkv: Thought-adaptive kv cache compression for efficient reasoning models. In *International Conference on Learning Representations*, volume 2026, pp. 110072–110106, 2026.

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

Zunhai Su, Zhe Chen, Wang Shen, Hanyu Wei, Linge Li, Huangqi Yu, and Kehong Yuan. Rotatekv: Accurate and robust 2-bit kv cache quantization for llms via outlier-aware adaptive rotations. *arXiv preprint arXiv:2501.16383*, 2025.

Renliang Sun, Wei Cheng, Dawei Li, Haifeng Chen, and Wei Wang. Stop when enough: Adaptive early-stopping for chain-of-thought reasoning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27250–27268, 2026.

Qwen Team. Qwq-32b: Embracing the power of reinforcement learning, March 2025. URL https://qwenlm.github.io/blog/qwq-32b/.

Xinpeng Wang, Nitish Joshi, Barbara Plank, Rico Angell, and He He. Is it thinking or cheating? detecting implicit reward hacking by measuring reasoning effort. In *International Conference on Learning Representations*, volume 2026, pp. 96956–96981, 2026a.

Zihan Wang, Cheng Tang, Lei Gong, Cheng Li, Chao Wang, Wenqi Lou, Xuehai Zhou, et al. Crystal-kv: Efficient kv cache management for chain-of-thought llms via answer-first principle. *arXiv preprint arXiv:2601.16986*, 2026b.

Zihao Wei, Liang Pang, Jiahao Liu, Jingcheng Deng, Shicheng Xu, Zenghao Duan, Jingang Wang, Fei Sun, Xunliang Cai, Huawei Shen, et al. Stop spinning wheels: Mitigating llm overthinking via mining patterns for early reasoning exit. *arXiv e-prints*, pp. arXiv–2508, 2025.

Yang Xiang, Yixin Ji, Ruotao Xu, Dan Qiao, Zheming Yang, Juntao Li, and Min Zhang. When is thinking enough? early exit via sufficiency assessment for efficient reasoning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23541–23556, 2026.

Chenxu Yang, Qingyi Si, Yongjie Duan, Zheliang Zhu, Chenyu Zhu, Qiaowei Li, Minghui Chen, Zheng Lin, and Weipinng Wang. Dynamic early exit in reasoning models. In *International Conference on Learning Representations*, volume 2026, pp. 88170–88210, 2026.

Jingyang Yi, Jiazheng Wang, and Sida Li. Shorterbetter: Guiding reasoning models to find optimal inference length for efficient reasoning. *Advances in Neural Information Processing Systems*, 38: 39011–39043, 2026.

Zihao Zeng, Xuyao Huang, Boxiu Li, Hao Zhang, and Zhijie Deng. Done is better than perfect: Unlocking efficient reasoning by structured multi-turn decomposition. *arXiv preprint arXiv:2505.19788*, 2025.

Anqi Zhang, Yulin Chen, Jane Pan, Chen Zhao, Aurojit Panda, Jinyang Li, and He He. Reasoning models know when they're right: Probing hidden states for self-verification. *arXiv preprint arXiv:2504.05419*, 2025a.

Jiajie Zhang, Nianyi Lin, Lei Hou, Ling Feng, and Juanzi Li. Adaptthink: Reasoning models can learn when to think. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 3716–3730, 2025b.

Tao Zhang, Ziqian Zeng, Hao Peng, Huiping Zhuang, and Cen Chen. Mixkvq: Query-aware mixed-precision kv cache quantization for long-context reasoning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7189–7204, 2026a.

Xinliang Frederick Zhang, Anhad Mohananey, Alexandra Chronopoulou, Pinelopi Papalampidi, Somit Gupta, Tsendsuren Munkhdalai, Lu Wang, and Shyam Upadhyay. Do llms really need 10+ thoughts for "find the time 1000 days later"? towards structural understanding of llm overthinking. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 17005–17030, 2026b.

Zhongzhu Zhou, Donglin Zhuang, Jisen Li, Ziyan Chen, Shuaiwen Leon Song, Ben Athiwaratkun, and Xiaoxia Wu. Oscar: Offline spectral covariance-aware rotation for 2-bit kv cache quantization. *arXiv preprint arXiv:2605.17757*, 2026.

A Appendix

A.1 Illustrative FP16 and KIVI K2V2 Rollouts

Figures 5 and 6 show responses to the same algebra question. Both obtain $x = 8$, $y = -1$, and the correct product -8, but the KIVI K2V2 response continues with additional checks after reaching the answer. This example illustrates the verification-heavy CoT inflation analyzed in the main text.

Table 7: Comparison with the overthinking-marker logit penalty of (Lotfi et al., 2026) on DeepSeek-R1-Distill-Qwen-7B. All rows except FP16 use KIVI K2V2; Marker Penalty is evaluated in this matched configuration.

	GSM8K		AIME 2024		LiveCodeBench		MATH-500		AMC 2023		Overall	
Method	Pass@1	Tokens	Pass@1	Tokens	Pass@1	Tokens	Pass@1	Tokens	Pass@1	Tokens	Pass@1	Tokens
FP16 Base	92.4	1,833	56.7	13,232	37.6	15,232	89.8	5,661	77.5	9,293	70.80	9,050
KIVI K2V2	89.1	2,454	50.0	18,106	34.3	22,523	84.6	7,846	72.5	12,991	66.10	12,784
Marker Penalty	88.5	2154	53.3	11,267	37.0	14,812	86.4	4,713	75.0	6,891	68.04	7,967
QuantThink	92.0	1,142	53.3	8,842	38.0	10,688	88.8	3,226	80.0	4,792	70.42	5,738

Table 7 places a training-free marker penalty and QuantThink under the same KIVI K2V2 evaluation setting. Relative to the quantized base, the marker penalty raises macro-average pass@1 from 66.10 to 68.04 while reducing generated tokens from 12,784 to 7,967. QuantThink reaches 70.42 pass@1 with 5,738 tokens. The two interventions act at different points: the penalty changes selected logits during decoding, whereas QuantThink updates the policy during training and uses ordinary quantized decoding at test time. This comparison concerns inference outcomes, not the different training costs of the methods.

The following paired responses offer a concrete view of the behavior behind the aggregate token counts. Both reach the same algebraic solution, but the quantized trace revisits it through additional checks. A single example illustrates the type of redundancy considered here; the benchmark-level measurements, rather than this example, support the aggregate comparison.

</think>
To solve the system of equations \(x + y = 7 \) and \(x - y = 9 \), I can start by adding the two equations together. This will eliminate the \(y \) variable and allow me to
solve for \(x \).Adding the equations:
\[(x + y) + (x - y) = 7 + 9\] \[2x = 16\] \[x = 8\]
Now that I have the value of \(x \), I can substitute it back into one of the original equations to find \(y \).
Using the first equation:
\[8 + y = 7\] \[y = 7 - 8\] \[y = -1\]
Finally, to find the product of \(x \) and \(y \):
\[x \times y = 8 \times (-1) = -8\]
</think>
To solve the system of equations:
\[
\begin{cases}
x + y = 7 \quad \text{(1)} \\
x - y = 9 \quad \text{(2)}
\end{cases}
\]
Step 1: Add the two equations to eliminate \(y \).
\[
\begin{align*}
(x + y) + (x - y) &= 7 + 9 \\
2x &= 16 \\
x &= \frac{16}{2} \\
x &= 8
\end{align*}
\]
Step 2: Substitute \(x = 8 \) back into equation (1) to find \(y \).
\[
\begin{align*}
8 + y &= 7 \\
y &= 7 - 8 \\
y &= -1
\end{align*}
\]
Step 3: Calculate the product of \(x \) and \(y \).
\[
x \times y = 8 \times (-1) = -8
\]
Final Answer:
\[
\boxed{-8}
\]

FP16 Rollout

Figure 5: FP16 response to the algebra question. The highlighted CoT derives the answer before the conclusion.

</think>
Okay, so I have this problem here: If x plus y equals 7 and x minus y equals 9, I need to find the product of x and y. Hmm, let me think about how to approach this.
First, I remember that when you have two equations with two variables, you can solve for one variable and then substitute it into the other equation. So, maybe I can solve one of the equations for one variable and then plug it into the other equation. Let me write down the equations again to make it clearer.
We have:
1. x + y = 7
2. x - y = 9
Hmm, okay. Maybe I can add these two equations together. If I add the left sides, I get (x + y) + (x - y). Let me compute that: x + y + x - y. The y terms cancel out, so that's 2x. On the right side, 7 + 9 is 16. So, 2x = 16. Then, dividing both sides by 2, I get x = 8. Okay, so x is 8.
Now that I know x is 8, I can plug that back into one of the original equations to find y. Let's use the first equation: x + y = 7. Substituting x with 8, we get 8 + y = 7. Subtracting 8 from both sides, y = 7 - 8, which is y = -1. So, y is -1.
Now, the question asks for the product of x and y. So, that would be x multiplied by y. Plugging in the values, that's 8 * (-1). Hmm, 8 times -1 is -8. So, the product is -8.
Wait, let me double-check my steps to make sure I didn't make a mistake. I added the two equations and got 2x = 16, so x = 8. Then, substituting back, y = -1. Multiplying them gives -8. That seems right.
Alternatively, I can check by plugging x = 8 and y = -1 back into the original equations to make sure they hold true.
First equation: 8 + (-1) = 7. Yep, that's correct because 8 - 1 is 7.
Second equation: 8 - (-1) = 8 + 1 = 9. That's also correct. So, both equations are satisfied with x = 8 and y = -1.
Therefore, the product of x and y is indeed -8.
I think that's it. I don't see any mistakes in my calculations, so I'm confident that the product is -8.
Final Answer
The product of \(x \) and \(y \) is \boxed{-8}.
</think>
Given the equations:
1. \(x + y = 7 \)
2. \(x - y = 9 \)
We can solve for \(x \) and \(y \) by adding the two equations:
\[(x + y) + (x - y) = 7 + 9\]
\[2x = 16\]
\[x = 8\]
Substituting \(x = 8 \) back into the first equation:
\[8 + y = 7\]
\[y = 7 - 8\]
\[y = -1\]

The product of \(x \) and \(y \) is:
\[
x \cdot y = 8 \cdot (-1) = -8
\]

Thus, the product of \(x \) and \(y \) is \(\boxed{-8}\).

KIVI K2V2 Rollout

Figure 6: KIVI K2V2 response to the same question. After deriving -8, the highlighted CoT continues with redundant checks prompted by *Wait* and *Alternatively*.

A.2 Additional Results on QwQ-32B

Table 8: QwQ-32B results from our evaluations. Overall is the unweighted mean across all five benchmarks.

	GSM8K		AIME 2024		LiveCodeBench		MATH-500		AMC 2023		Overall	
Method	Pass@1	Tokens	Pass@1	Tokens	Pass@1	Tokens	Pass@1	Tokens	Pass@1	Tokens	Pass@1	Tokens
Full-precision Base	96.2	1,921	66.7	10,821	63.6	6,435	93.8	4,508	100.0	6,792	84.06	6,095
KIVI K2V2	93.3	2,183	60.0	11,037	58.3	8,913	92.6	4,719	92.5	8,438	79.34	7,058
GRPO	94.1	1,782	63.3	10,762	60.2	7,675	94.0	4,263	97.5	7,871	81.82	6,471
DEER	95.7	1,481	70.0	8,097	59.7	6,728	94.6	3,316	95.0	5,782	83.00	5,081
JET	95.5	1,577	66.6	7,811	61.2	6,899	94.0	3,447	92.5	5,267	81.96	5,000
S-GRPO	95.3	1,631	66.6	7,631	60.3	6,727	94.0	3,271	95.0	5,072	82.24	4,866
QuantThink	95.5	1,465	70.0	7,265	60.1	6,839	93.8	3,301	95.0	5,011	82.88	4,776

The KIVI run uses K2V2 with group size 32, a 32-token residual window, BF16, and two GPUs. On the LiveCodeBench subset, KIVI generates 8,913 tokens at 5.5 tokens/s, compared with 6,435 tokens at 12.4 tokens/s for FP16;

A.3 Q-GRPO Training Algorithm

Algorithm 1 Quantization-Aware Group Relative Policy Optimization (Q-GRPO)

Require: Prompt x, quantized policy π_θ, verification dictionary $\mathcal{D}_V$, answer checker Acc, weights λ and η
Ensure: Updated LoRA parameters θ

Quantized rollout and segmentation:
Freeze the rollout policy $\pi_{\theta_{\mathrm{old}}} \leftarrow \pi_\theta$
Generate a complete quantized trajectory $y \sim \pi_{\theta_{\mathrm{old}}}(\cdot \mid x)$
Segment y with $\mathcal{D}_V$ into R, checkpoints $A_0, \ldots, A_m$, verification episodes $V_1, \ldots, V_m$, and conclusion C
Commitment-group construction:
for $j = 0, \ldots, m$ **do**
    Append </think> after $y_{\le A_j}$ and decode conclusion C_j with $\pi_{\theta_{\mathrm{old}}}$
    Set $c_j \leftarrow \mathrm{Acc}(C_j)$ and $\ell_j \leftarrow |y_{\le A_j}| + |C_j|$
end for
Set $\mathcal{C}(x) \leftarrow \{j : c_j = 1\}$ and $c \leftarrow \min \mathcal{C}(x)$; skip x if $\mathcal{C}(x) = \emptyset$ or $c = m$
Reward and advantage computation:
for $j = 0, \ldots, m$ **do**
    $r_j^{\mathrm{vc}} \leftarrow -1$ if $c_j = 0$; otherwise $r_j^{\mathrm{vc}} \leftarrow 1/(1 + j - c)$
    $A_j^{\mathrm{vc}} \leftarrow r_j^{\mathrm{vc}} - (m+1)^{-1} \sum_{k=0}^{m} r_k^{\mathrm{vc}}$
    Compute A_j^{len} from the end-to-end lengths of correct branches using Eq. equation 7
    $A_j^{\text{Q-GRPO}} \leftarrow A_j^{\mathrm{vc}} + \lambda A_j^{\mathrm{len}}$
end for
Policy update:
Mask the shared prefixes $y_{\le A_j}$ and retain </think> and all tokens in C_j in the policy loss
Update only the LoRA parameters with the clipped objective in Eq. equation 9 and clipping range η

A.4 Illustrative Reward Assignment

Figure 7 follows one quantized rollout through its candidate commitment points. An incorrect early conclusion receives -1; correct conclusions receive successively smaller verification-to-commit rewards, which are then centered by the group mean to form A^{vc}.

In this example, the problem is to compute 50% of one third of 36. An early conclusion can still be wrong even when the visible derivation has already mentioned 6; its reward depends on the decoded final answer, not on whether the number appears somewhere in the prefix. Once a branch outputs the correct answer, later correct branches retain positive but progressively smaller rewards as they include further verification episodes. The separate length advantage then compares their complete generated responses, including the conclusion after </think>.

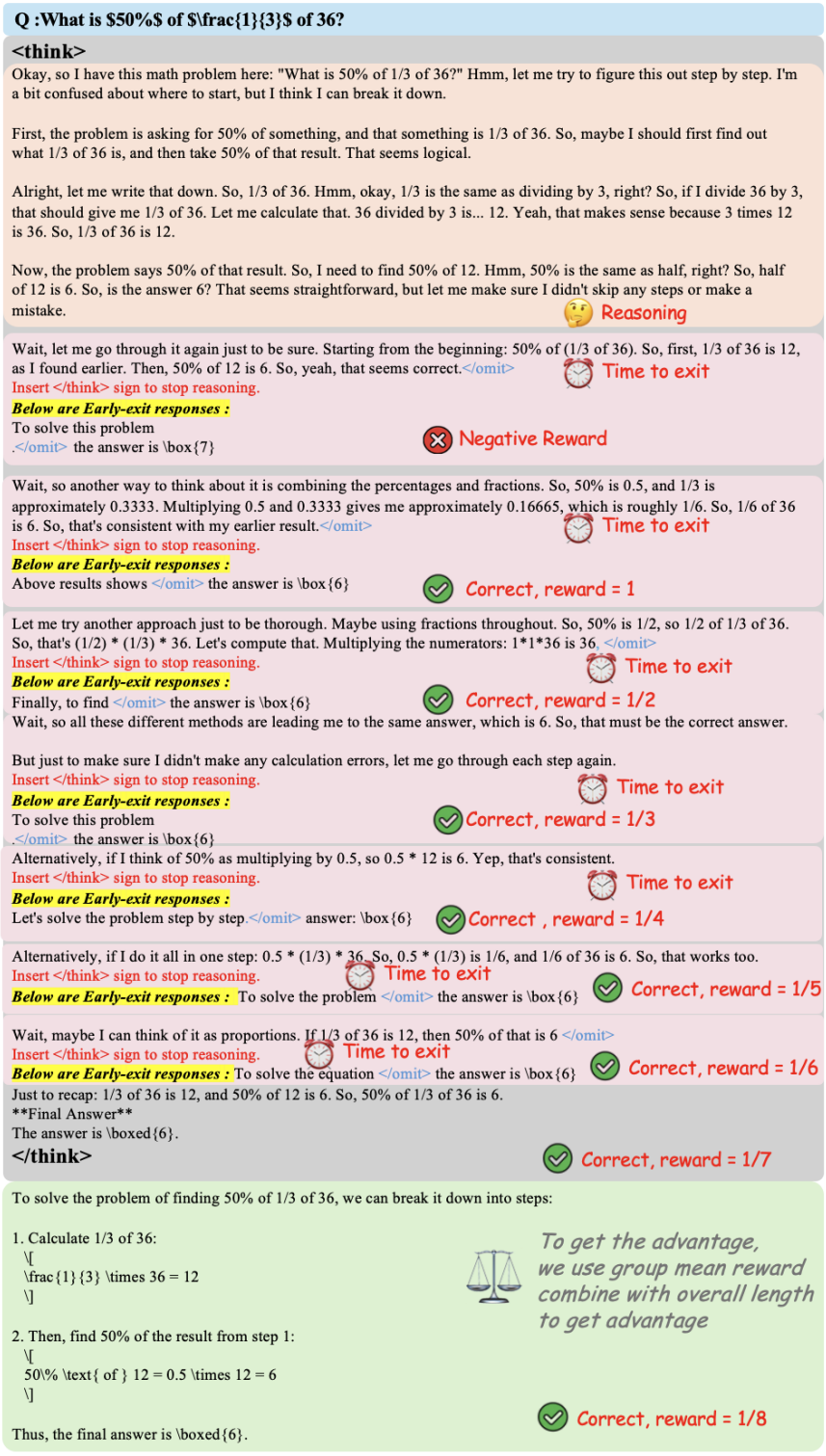

Figure 7: Illustrative Q-GRPO reward assignment across early-exit branches from a single quantized rollout.

A.5 Segmentation Ablation

We compare the offline dictionary-based segmentation with random and uniform boundaries, as well as boundaries selected by a BERT-based detector. Sentence-level (-S) and token-level (-T) variants use the same number of boundaries as QuantThink, isolating the effect of where the candidate commitment points are placed.

Figure 8 visualizes the recurring cues in the offline verification dictionary.



Figure 8: Word cloud of the offline verification-cue dictionary constructed from DeepSeek-R1-Distill-Qwen-7B training rollouts. Word size reflects the observed cue frequency.

To assess how reliably the dictionary recovers semantic boundaries, we compare it with prompted LLM segmentation and a BERT-based detector on manually annotated trajectories from three reasoning models. Figure 9 shows the prompt used by the LLM segmenter. For each model, the three methods are evaluated against the same sentence-boundary annotations.

Template of Segmentation Prompt

Prompt

You are given a complete trajectory from a reasoning language model. Segment it into semantic components R, A, V, and C
Definitions:
•R (Reasoning): The initial forward reasoning process toward a candidate answer.
•A (Answer-ready checkpoint): A semantic boundary where the reasoning so far is sufficient to produce a final answer.
•V (Verification): A complete episode that checks, recomputes, validates, revises, or challenges an answer-ready result.
•C (Conclusion): The final response span from immediately after </think> to <EOS>.
The structure should follow:
$R \to A_0 \to V_0 \to A_1 \to V_1 \to \ldots \to A_m \to C$
A_0 is the first answer-ready point. Each later A_i marks the completion of a verification episode and does not imply correctness.
Requirements:
1.Use semantic interpretation rather than surface cues.
2.Preserve the original text exactly.
3.Place boundaries only between complete semantic units.
4.Do not treat ordinary forward reasoning as verification.
5.If no answer-ready point appears before </think>, return only R and C.
6.If the trajectory becomes answer-ready but is not subsequently verified, use A_0 without V.
7.Segments must be ordered, non-overlapping, and cover the entire trajectory.

Figure 9: Prompt used for LLM-based semantic segmentation of reasoning trajectories into reasoning (R), answer-ready checkpoints (A), verification episodes (V), and the conclusion (C).

As a supervised alternative, we train a BERT token classifier to detect hesitation cues that initiate another verification episode. For token t_i, let $z_i \in \{0, 1\}$ indicate whether it is a hesitation cue. Its contextual representation h_i produces a cue probability p_i, trained with binary cross-entropy:

$$h_i = \mathrm{BERT}(t_{1:n})_i, \quad p_i = \sigma(w^\top h_i + b), \quad \mathcal{L}_{\mathrm{BERT}} = -\sum_{i=1}^{n} \big[z_i \log p_i + (1 - z_i) \log(1 - p_i) \big]. \tag{12}$$

Detected cues provide candidate verification boundaries for the comparisons in Tables 9 and 10.

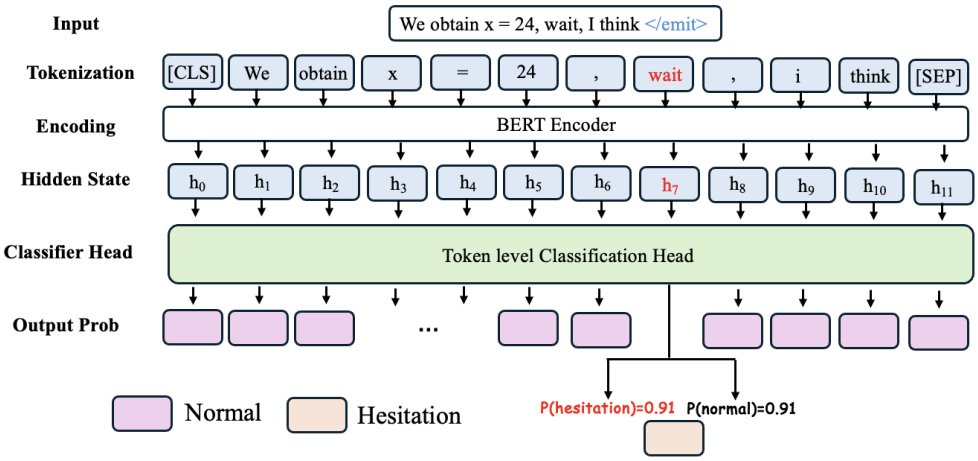

Figure 10: BERT-based token classification of hesitation cues used to propose verification boundaries.

Table 9: Segmentation accuracy and cost on manually annotated MATH-500 trajectories. Precision, recall, and F1 evaluate detected A–V transition boundaries; Exact Match is the fraction of trajectories with a fully correct segmentation. Latency is mean segmentation time per trajectory in seconds.

Segmentation method	Precision ↑	Recall ↑	F1 ↑	Exact Match ↑	Latency (s) ↓
DeepSeek-R1-Distill-Qwen-1.5B					
Dictionary (ours)	0.92	0.90	0.91	0.64	0.11
LLM segmentation	0.86	0.82	0.84	0.61	10.6
BERT-based detector	0.85	0.84	0.84	0.60	0.69
DeepSeek-R1-Distill-Qwen-7B					
Dictionary (ours)	0.94	0.91	0.92	0.67	0.12
LLM segmentation	0.89	0.80	0.84	0.63	17.7
BERT-based detector	0.87	0.84	0.85	0.62	0.73
Qwen3-8B					
Dictionary (ours)	0.93	0.92	0.92	0.66	0.10
LLM segmentation	0.90	0.84	0.87	0.65	20.3
BERT-based detector	0.88	0.86	0.87	0.60	0.70

The boundary metrics in Table 9 answer different questions. Precision penalizes spurious verification boundaries, whereas recall penalizes missed ones; F1 summarizes both. Exact Match is stricter, requiring all boundaries in a trajectory to agree with the manual annotation. Latency measures the cost of finding candidate boundaries before branch evaluation. Detecting a hesitation cue alone does not establish that the following verification is unnecessary: branch correctness supplies that separate signal during Q-GRPO group construction.

Table 10: Segmentation ablation on DeepSeek-R1-Distill-Qwen-7B with MATH-500 under KIVI K2V2. -S/-T denotes sentence/token-level splitting, and Uniform denotes uniformly sampled boundaries. All variants use equal numbers of boundaries.

Method	Pass@1 ↑	Tokens ↓	RVC ↓	Latency (s) ↓
Random-S	87.0	3,531	0.91	11.4
Random-T	83.2	5,529	1.07	16.8
Uniform-S	85.8	4,821	0.98	14.6
Uniform-T	86.0	4,531	0.95	13.9
BERT-based detector	88.6	3,483	0.81	10.8
QuantThink	88.8	3,226	0.78	10.2

Table 10 tests whether boundary placement matters after matching the number of candidate checkpoints. On MATH-500, the dictionary-based QuantThink setting attains 88.8 pass@1 with 3,226 tokens, compared with 88.6 and 3,483 for the BERT-based detector. The random and uniform placements generally use more tokens or lose accuracy, especially random token-level splitting. Unlike Table 9, which measures boundary detection itself, this ablation measures the downstream answer-quality and decoding-cost consequences of those boundaries.

A.6 S-GRPO Component Ablation

Table 11 isolates whether the gain over S-GRPO comes from semantic checkpoint placement, the verification-to-commit advantage, or the end-to-end length advantage. The reward-only variants retain the semantic checkpoints: A^{vc} replaces S-GRPO's decay reward, whereas A^{len} is added to that original reward.

Table 11: S-GRPO component ablation on DeepSeek-R1-Distill-Qwen-7B with MATH-500 under KIVI K2V2. Checkmarks denote enabled components.Length advantage coefficient stays 0.1.

Method	**Semantic boundaries**	A^{vc}	A^{len}	Pass@1 ↑	Tokens ↓
S-GRPO	–	–	–	86.2	3,531
	✓	–	–	87.2	3,804
	✓	✓	–	87.6	3,721
	✓	–	✓	88.0	3,351
QuantThink	✓	✓	✓	88.8	3,226

A.7 Six-Run Average Results

Table 12: Mean pass@1 and generated tokens over six independent runs for DeepSeek-R1-Distill-Qwen-7B and Phi-4-reasoning-14B. FP16 Base uses full precision; all other methods use KIVI K2V2. "Overall" is the macro-average across the five benchmarks. Values are reported as mean ± standard deviation.

	GSM8K		**AIME 2024**		**LiveCodeBench**		**MATH-500**		**AMC 2023**		**Overall**	
Method	Pass@1	Tokens	Pass@1	Tokens	Pass@1	Tokens	Pass@1	Tokens	Pass@1	Tokens	Pass@1	Tokens
DeepSeek-R1-Distill-Qwen-7B												
FP16 Base	92.5±0.4	1,849±43	54.4±4.8	13,118±487	37.0±1.1	15,371±521	89.6±0.7	5,608±132	79.2±2.8	9,381±314	70.5±1.1	9,065±257
KIVI K2V2	88.6±0.6	2,426±71	44.4±5.1	18,294±684	34.6±1.3	22,318±796	84.5±0.9	7,925±204	70.0±3.2	12,874±461	64.4±1.3	12,767±382
GRPO	89.9±0.5	2,367±79	53.3±4.6	20,219±771	35.1±1.2	24,006±851	86.0±0.8	7,221±218	73.8±3.0	13,781±518	67.6±1.2	13,519±426
DEER	89.0±0.6	1,257±38	45.6±5.4	11,746±449	36.9±1.2	12,154±402	85.7±0.8	4,481±119	72.9±3.1	8,462±287	66.0±1.3	7,620±221
JET	90.3±0.5	1,381±44	38.3±5.7	12,742±486	34.5±1.4	14,716±503	83.3±0.9	5,194±154	75.0±3.0	7,018±251	64.3±1.4	8,210±236
S-GRPO	91.1±0.4	1,219±31	50.6±4.5	10,208±361	36.0±1.0	13,696±422	86.2±0.7	3,487±91	78.3±2.7	6,092±192	68.5±1.0	6,940±181
QuantThink	92.1±0.3	1,131±27	52.2±4.2	8,937±286	37.8±0.9	10,571±318	88.7±0.6	3,264±82	80.8±2.5	4,746±143	70.3±0.9	5,730±154
Phi-4-reasoning-14B												
FP16 Base	95.5±0.3	2,151±52	67.8±4.1	10,982±398	53.0±1.0	10,986±344	93.0±0.5	3,802±96	91.7±2.1	5,581±171	80.2±0.9	6,700±198
KIVI K2V2	93.2±0.4	2,514±69	52.8±4.9	14,636±536	52.8±1.1	12,672±421	90.4±0.7	5,136±138	89.2±2.3	6,742±211	75.7±1.0	8,340±264
GRPO	93.9±0.4	2,416±73	61.7±4.4	16,154±602	53.5±1.0	13,731±447	90.3±0.7	4,581±126	87.9±2.5	7,194±238	77.5±1.0	8,815±281
DEER	93.8±0.4	971±28	61.1±4.7	10,148±387	53.5±1.0	10,037±318	89.5±0.7	2,479±71	92.1±2.0	4,002±122	78.0±1.0	5,527±167
JET	94.1±0.4	1,211±34	60.0±4.6	10,266±404	52.7±1.1	8,724±291	90.7±0.6	2,727±78	91.7±2.1	4,231±137	77.8±1.0	5,432±173
S-GRPO	94.3±0.3	1,013±25	62.2±4.3	8,741±301	53.3±0.9	7,808±247	91.4±0.6	2,094±56	89.2±2.4	3,243±99	78.1±0.9	4,580±137
QuantThink	95.9±0.3	1,089±24	63.9±4.0	8,563±274	54.4±0.9	6,895±201	92.9±0.5	2,143±51	89.6±2.3	3,465±104	79.3±0.8	4,431±128

Table 12 averages six separate single-response evaluations; it does not report pass@6, which would score whether any of six attempts succeeds. This distinction matters because averaging pass@1 retains the accuracy–cost trade-off of a typical generation rather than granting a model six attempts per question. Under KIVI K2V2, QuantThink raises the 7B macro-average from 64.4 to 70.3 while reducing mean tokens from 12,767 to 5,730. For Phi-4-reasoning-14B, the corresponding values move from 75.7 to 79.3 and from 8,340 to 4,431 tokens. These repeated-run means complement the single-run main results without replacing them.

A.8 Raw Training Logs

Figures 11–13 reproduce the unaltered Weights & Biases screenshots from Q-GRPO training of DeepSeek-R1-Distill-Qwen-7B under KIVI K2V2. They track the policy-gradient loss, commitment accuracy on training batches, and mean reasoning length by optimization step. The reasoning length falls sharply early in training and then fluctuates at a lower level, while batch commitment accuracy remains mostly high. These are training diagnostics, not held-out benchmark results.

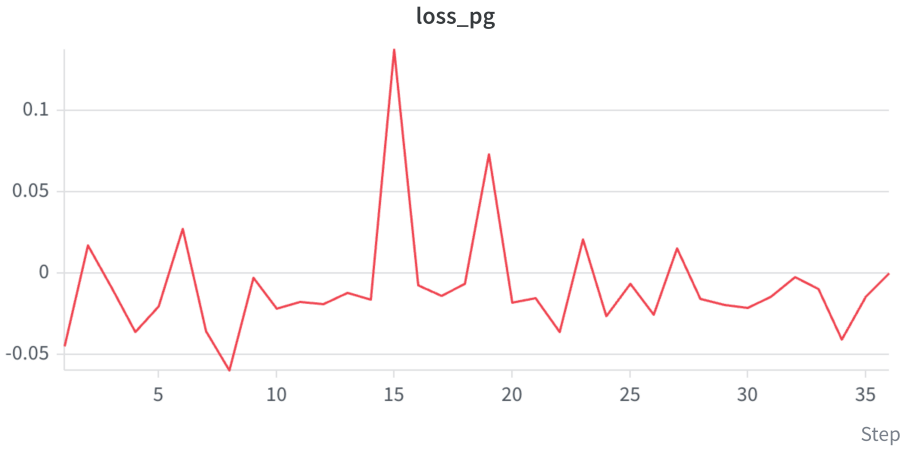

Figure 11: Direct Weights & Biases screenshot of the 7B policy-gradient loss (`loss_pg`) during Q-GRPO training.



Figure 12: Direct Weights & Biases screenshot of 7B training-batch commitment accuracy (`commit_acc`).

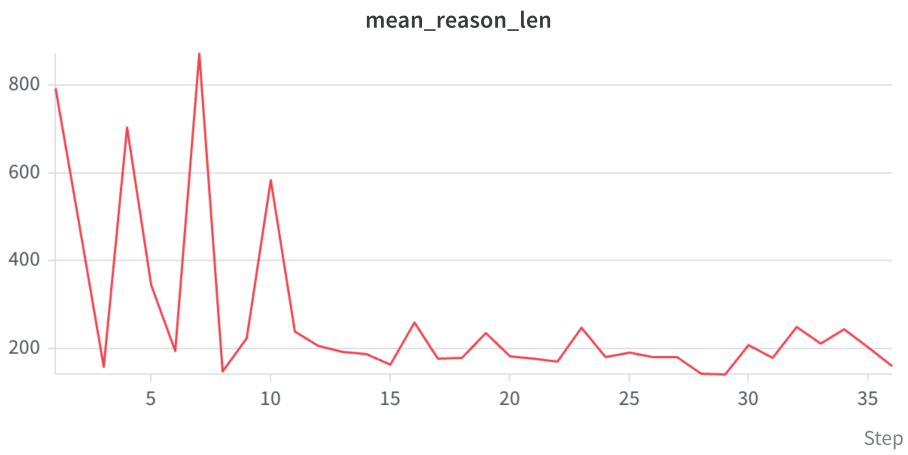

Figure 13: Direct Weights & Biases screenshot of the 7B mean reasoning length (`mean_reason_len`) during training; this measures CoT tokens only, excluding the conclusion.

Normal Reasoning

To evaluate \(\log_2 64\), I need to determine the exponent that raises 2 to get 64.

I know that \(2^6 = 64\), so the logarithm base 2 of 64 is 6.
</think>

To evaluate \(\log_2 64\), we need to determine the exponent that raises 2 to get 64.

Step 1: Understand the logarithm expression.
\[
\log_2 64 = x \quad \text{means} \quad 2^x = 64
\]

Step 2: Find the value of \(x\) such that \(2^x = 64\).
\[
2^6 = 64
\]

Step 3: Therefore, \(x = 6\).

Final Answer:
\[
\boxed{6}
\]

CoT-Completeness(CoT-C) = 1

Reward Cheating By Insert </think> at Start

</think>

To evaluate \(\log_2 64\), we need to determine the exponent that, when applied to the base 2, results in 64.

Let's break it down step by step:

1. **Understand the logarithm definition**:
   \[
   \log_2 64 = x \quad \text{means} \quad 2^x = 64
   \]

2. **Find the exponent \(x\)**:
   \[
   2^6 = 64
   \]
   Therefore, \(x = 6\).

Final Answer:
\[
\boxed{6}
\]

CoT-Completeness(CoT-C) = 0

Figure 14: Illustration of CoT-completeness (CoT-C). Some early-stop methods can obtain an artificially short CoT by emitting `</think>` immediately and moving the derivation into the conclusion. This bypasses explicit reasoning rather than learning when to stop, and can reduce answer accuracy; both responses in this illustrative example nevertheless answer correctly.

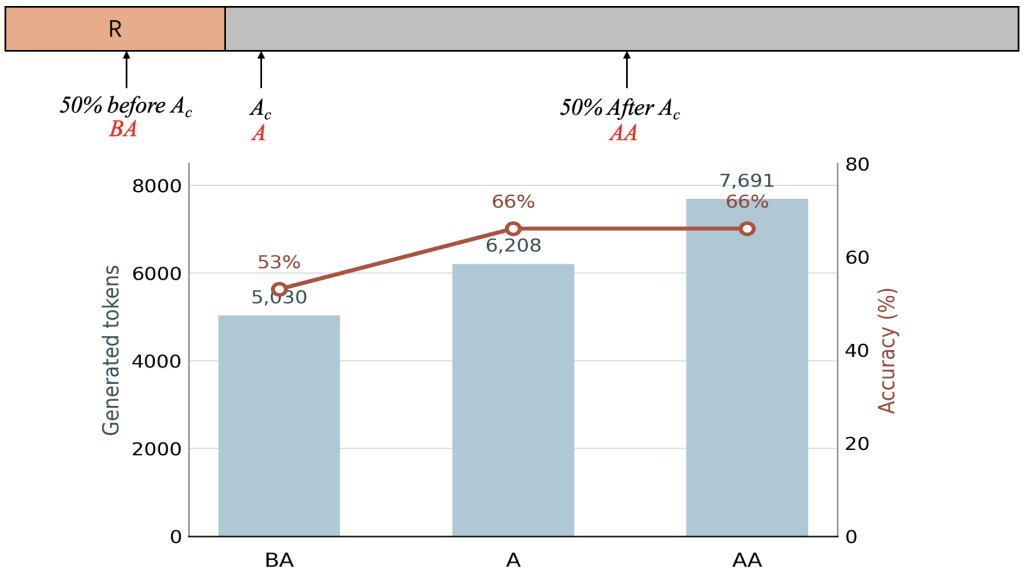

Figure 15: Answer stability around A_c on AIME 2024 trajectories collected under KIVI K2V2. We probe checkpoints 50% before A_c (BA), at A_c (A), and 50% after A_c (AA), averaging generated tokens and answer accuracy over ten continuations at each checkpoint. Accuracy remains at 66% from A to AA while mean length increases from 6,208 to 7,691 tokens, suggesting that the answer is already stable at A_c in aggregate.