

Adaptive Latent Block Insertion via Online Reward Optimization

Yewon Cho¹ Hyeonsoon Hwang^{1,2} Yein Park^{1,2} Hyeon Hwang¹ Jaewoo Kang^{1,2*}

Korea University¹ AIGEN Sciences²

{yewon330, hhs8746, 522yein, hyeon-hwang, kangj}@korea.ac.kr

Abstract

Chain-of-thought prompting improves reasoning by generating intermediate steps as discrete text, but each step is forced to commit to a single token even when the model is uncertain. Continuous latent reasoning offers an alternative: a latent state can preserve multiple candidate interpretations without immediately selecting one token. Existing training-free approaches either place latent states only before decoding or switch between explicit and latent modes without optimizing the inserted state itself. We propose **MidThink**, a training-free framework that inserts and optimizes latent blocks both at the start of reasoning and adaptively during generation. When a confidence trigger detects sustained mid-stream uncertainty, MidThink inserts a new latent block and refines it with a rollout-based reward computed from short future continuations. We further show that initialization strongly affects whether a mid-stream block preserves a superposition over multiple tokens or collapses toward a single token. Across five mathematical reasoning benchmarks, MidThink improves over training-free latent-reasoning baselines, with the largest gains on the AIME benchmarks. Mechanistic analysis links these gains to three measurable properties of the latent block: distributional entropy across layers, attention centroid position, and stability under reinforcement learning updates.

1 Introduction

Chain-of-thought (CoT) prompting has become the standard way to elicit reasoning from large language models: the model writes out intermediate steps as text before producing a final answer (Wei et al., 2022). This representation is interpretable, but it requires the model to select one token at every step, even when several continuations have similar probability. Once selected, an early token becomes

part of the context for all later steps and can steer the subsequent trace away from a correct solution (Zhu et al., 2026).

An alternative is to let the model reason in *continuous vectors* rather than discrete tokens (Hao et al., 2024). Instead of committing to one token at each step, the model passes a continuous vector forward—one that need not correspond to any single vocabulary entry, and can therefore encode multiple candidate interpretations at once. This property is called *superposition*: the latent vector keeps multiple reasoning paths alive in a single forward pass rather than collapsing to one prematurely (Zhu et al., 2026). Such a state can represent ambiguity that discrete CoT resolves immediately through token selection.

In practice, the promise has been hard to realize. Latent tokens have no anchor in the discrete vocabulary, so their distribution shifts easily during optimization and the resulting state is not robust. Coconut (Hao et al., 2024) addresses this by training the model to generate latent tokens directly, embedding the latent computation into the weights. Recent analysis tempers the promise of this approach: off-the-shelf models—and even Coconut-style fine-tuned models—collapse superposed inputs to a single interpretation within the first few transformer layers, leaving little superposition for the network to exploit (Rizvi-Martel et al., 2026).

Training-free methods remove the need for fine-tuning, but existing designs still impose two limitations. Methods that optimize latent states either do so only before decoding, as in LTPO (Ye et al., 2026), or refine complete candidate trajectories rather than intervening within an ongoing generation, as in LatentSeek (Li et al., 2025). Methods that do adapt latent computation during decoding, such as SwiReasoning and LaTER (Shi et al., 2026; Li et al., 2026), switch modes without optimizing an inserted mid-stream latent block per instance.

We address these limitations with **MidThink**,

*Corresponding author

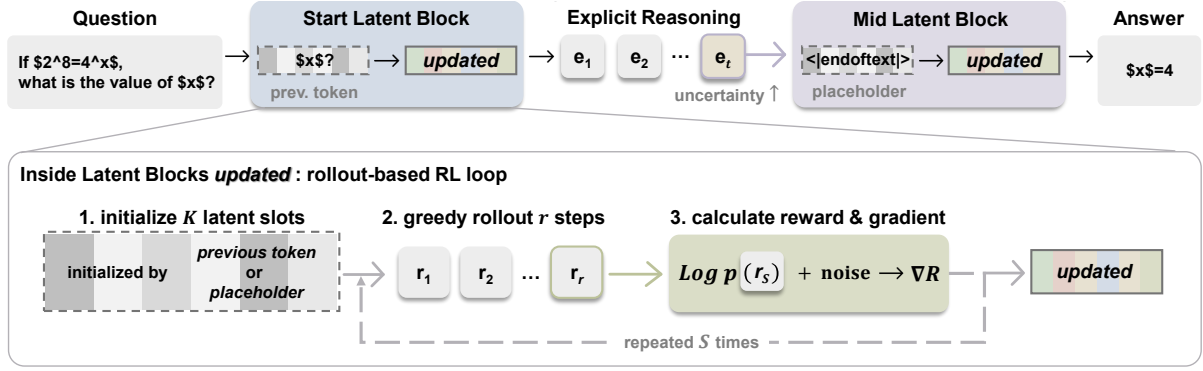


Figure 1: Overview of MidThink. A start latent block is inserted after the question and optimized before generation, and a confidence trigger inserts an additional mid latent block when uncertainty arises during decoding. Both blocks are refined by the rollout-based optimization loop shown below, which evaluates short continuations over S update steps, where S is the number of policy-gradient updates applied to each block. Rollout tokens are generated internally to score the reward and are never returned.

a training-free framework built on three design choices (Figure 1). First, we insert latent blocks at two points: a *start latent block* immediately after the question, and a *mid latent block* whenever a confidence trigger fires during generation, letting the model pause and refine its hidden state mid-stream without regenerating the full trace as LatentSeek does. Second, we optimize both blocks online with the same policy gradient procedure, replacing the next-token confidence reward of prior work with a *rollout-based reward*: we sample short continuations from each candidate latent state and aggregate their confidence, producing a signal that reflects future reasoning rather than only the next prediction. Third, we treat *initialization as a first-class design variable*. Our analysis shows that the initial values written into a latent block determine whether the block holds a distribution over multiple tokens or collapses immediately to one—and the optimization cannot recover what initialization throws away.

Our contributions are three.

- We propose MidThink, a training-free framework that combines a start latent block, a confidence-triggered mid latent block, and a rollout-based reward for online optimization.
- Across five reasoning benchmarks and four models, our method attains the highest mean Best@8 among training-free methods on all four, with the largest gains on the competition-level AIME benchmarks, where latent reasoning is hardest, improving over LTPO by up to 20 points on AIME2024.

- We give a mechanistic account of why initialization matters: good initializations keep the block superposed, a property that co-occurs with higher accuracy, and we distill this into four principles for selecting one (Section 5.4).

2 Related Work

Test-time latent reasoning. Training-free methods optimize or manipulate latent states at inference without updating model weights. LatentSeek (Li et al., 2025) optimizes latent states with policy gradients but evaluates each candidate by full autoregressive decoding, which is expensive. LTPO (Ye et al., 2026) avoids this cost by scoring top- k confidence at the latent positions in a single forward pass, but places latent tokens only at the start of generation and cannot respond to mid-stream uncertainty. Soft Thinking (Zhang et al., 2026) replaces each discrete token with a probability-weighted mixture of embeddings, an intervention applied at the token level rather than as an optimized block.

Other training-free approaches adapt latent computation during decoding. SwiReasoning (Shi et al., 2026) and LaTER (Li et al., 2026) switch between latent and explicit computation, whereas SeLaR (Fu and Luo, 2026) selectively activates soft representations at high-entropy steps. MidThink differs from these methods by inserting a dedicated mid-stream latent block and optimizing it per instance with a rollout-based reward.

Test-time scaling. A broad family of inference-time methods improves reasoning by allocating more compute without touching model weights, but operates on *explicit* traces rather than latent

states. Self-consistency (Wang et al., 2023) and Best-of- N (Cobbe et al., 2021) sample multiple full traces and aggregate or select among them; PoLR (Jindal et al., 2026) cuts this cost by expanding only the dominant cluster of short reasoning prefixes, and DiffAdapt (Liu et al., 2026b) predicts per-question difficulty to select an inference strategy. These methods are orthogonal to ours: they decide *how many* explicit traces to draw or *how long* to make them, whereas MidThink intervenes within a single trace. Combining its rollout reward and confidence trigger with prefix-level filtering or difficulty-adaptive budgeting is a natural direction for future work.

Training-required latent reasoning. A separate line of work trains models to use latent representations. Implicit CoT (Deng et al., 2023) distills explicit reasoning into hidden-state computation. Coconut (Hao et al., 2024) feeds hidden states back as continuous reasoning inputs after fine-tuning. SoftCoT (Xu et al., 2025), which we include as a baseline, produces soft thought tokens through a trained projection module. Soft Tokens, Hard Truths (Butt et al., 2025) studies the limitations of related training regimes. Unlike these approaches, MidThink requires no offline training.

Mechanistic understanding. Two recent studies inform our analysis. Rizvi-Martel et al. (2026) show that off-the-shelf LLMs collapse soft inputs to discrete interpretations within the first few layers, and that the next-token pretraining objective drives last-layer entropy collapse. Chen et al. (2026) show that answer tokens read reasoning traces with a forward-shifting attention centroid and persistent focus on semantic anchors when the answer is correct. We use both lenses—layerwise distributional analysis and attention pattern analysis—to ask whether our optimized latent blocks preserve superposition and whether they participate in benign self-reading.

3 Method

We introduce **MidThink**, a training-free framework that inserts and optimizes latent blocks before decoding and at uncertainty-triggered points during generation. MidThink first optimizes a start latent block after the question embedding. It then generates explicit reasoning tokens autoregressively. When the confidence trigger detects sustained uncertainty, it inserts and optimizes a mid latent block

at the current position before resuming generation.

Table 1 contrasts MidThink with prior methods: training-free approaches optimize latent states only before generation, while mid-stream methods switch computation mode without optimizing latent states. MidThink alone combines both.

3.1 Preliminaries

Let $\mathcal{M}_\theta$ be a frozen LLM with embedding layer E and hidden dimension d . Given a question q with embedded form $E(q) \in \mathbb{R}^{|q| \times d}$, we augment the input with two kinds of latent blocks. The *start latent block* $H^s = (h_1^s, \dots, h_{K_s}^s) \in \mathbb{R}^{K_s \times d}$ is inserted immediately after the question. During generation, whenever a confidence trigger fires at token position t_k , we insert a *mid latent block* $H_k^m \in \mathbb{R}^{K_m \times d}$ between the tokens already generated and the tokens still to come. The full input to the model at any point is therefore

$$E(q) \parallel H^s \parallel r_{1:t_1} \parallel H_1^m \parallel r_{t_1+1:t_2} \parallel \dots \parallel a, \quad (1)$$

where $r_{i:j}$ denotes explicit reasoning tokens from position i to j , and a is the final answer.

3.2 Start Latent Block Optimization

We treat H^s as the parameters of a Gaussian policy and optimize it for S update steps. At each step t , we sample a perturbation $A^{(t)} = H^{s,(t)} + \epsilon^{(t)}$ with $\epsilon^{(t)} \sim \mathcal{N}(0, \sigma^2 I)$. We then compute a *rollout-based reward*: starting from $A^{(t)}$, the model generates r tokens internally against the KV cache of the preceding context, and we average the top- k probabilities at each rolled-out position. These rollout tokens are used only to score the reward and are never returned as part of the output. Formally, with $P_i^{(t)}$ the predicted distribution at rolled-out position i , the reward is

$$R_{\text{RL}}(A^{(t)}) = \frac{1}{r} \sum_{i=1}^r \frac{1}{k} \sum_{v \in \text{top}_k(P_i^{(t)})} \log P_i^{(t)}(v). \quad (2)$$

The update follows the REINFORCE estimator:

$$H^{s,(t+1)} = H^{s,(t)} + \eta \cdot (R_{\text{RL}}(A^{(t)}) - b^{(t)}) \cdot \frac{\epsilon^{(t)}}{\sigma^2}. \quad (3)$$

where $b^{(t)}$ is a running-mean baseline computed from the reward history $\{R_{\text{RL}}(A^{(\tau)})\}_{\tau < t}$, which reduces the variance of the REINFORCE estimator without changing the gradient in expectation (Williams, 1992). We additionally decay the

Method	Train. free	Start latent	Mid latent	Rollout reward
CoT (Wei et al., 2022)	✓	✗	✗	✗
Coconut (Hao et al., 2024)	✗	Sequence	Sequence	✗
SoftCoT (Xu et al., 2025)	✗	✓	✗	✗
LatentSeek (Li et al., 2025)	✓	✓	✗	✓
Soft Tokens (Butt et al., 2025)	✗	Sequence	Sequence	✗
LTPO (Ye et al., 2026)	✓	✓	✗	✗
Soft Thinking (Zhang et al., 2026)	✓	✗	Token level	✗
LaTER (Li et al., 2026)	✓	✓	✗	✗
MidThink (Ours)	✓	✓	✓	✓

Table 1: Feature-level comparison of MidThink and representative latent reasoning methods. MidThink is the only training-free method combining start and mid-stream latent insertion with a rollout-based reward.

exploration variance geometrically $\sigma_{t+1}^2 = \gamma\sigma_t^2$ with $\gamma < 1$, so that early steps explore broadly and later steps refine locally; since the Gaussian perturbation itself injects exploration noise, this stabilizes convergence without explicit entropy regularization. Since the optimization trajectory is not monotonic, we return the best-reward iterate of H^s rather than the final one, a standard practice in zeroth-order policy optimization (Ye et al., 2026) that reduces sensitivity to the choice of S .

We tune a small number of hyperparameters (the learning rate η , the exploration scale σ_0 , and the mid block size K_m) per model on 200 examples sampled from the training splits of GSM8K and ASDiv-Aug. All other hyperparameters are fixed globally. For the remaining benchmarks (MATH500, AIME2024, and AIME2025) we directly transfer the GSM8K configuration. The full configuration and a sensitivity analysis appear in Appendices B and C. The rollout window is the central change from LTPO. LTPO scores only the latent positions themselves; we score short continuations, so the reward reflects how the latent state shapes future explicit reasoning, not just how confident the model is at the latent positions.

3.3 Confidence Trigger

The confidence trigger determines when to insert a mid block. It fires only under sustained uncertainty, filtering out isolated entropy spikes caused by sentence boundaries or tokenizer artifacts.

Uncertainty metric. At each generated token, we compute the top- k uncertainty u_t from the predictive distribution.

$$u_t = - \sum_{v \in \text{top}_k(P_t)} \tilde{P}_t(v) \log \tilde{P}_t(v), \quad (4)$$

where $\tilde{P}_t$ is the distribution renormalized over the top- k tokens. We use top- k entropy rather than full-vocabulary entropy because the latter is dominated by the long tail of low-probability tokens and varies weakly across reasoning quality; top- k entropy measures how probability mass is distributed among the likely candidate continuations.

Smoothing. Raw u_t is noisy. We apply an exponential moving average with smoothing coefficient α : $\bar{u}_t = \alpha u_t + (1 - \alpha)\bar{u}_{t-1}$. The smoothing suppresses single-token spikes while preserving sustained increases.

Trigger condition. The trigger fires at position t when $\bar{u}_\tau > \theta$ for all τ in a dwell window $[t - d + 1, t]$, where θ is the threshold and d is the dwell length. After a trigger fires and the mid block is inserted, we impose a refractory period equal to the dwell length d , during which no further trigger can fire, preventing back-to-back insertions.

Math token protection. We additionally protect math tokens (digits, operators, and $\text{\LaTeX}$ macros) from interruption, since splitting a numeric expression with a latent block harms readability and does not address the underlying uncertainty.

3.4 Mid Latent Block Optimization

When the trigger fires at position t_k , we insert H_k^m and optimize it by the same procedure as H^s . Only the initialization differs. The start block has no preceding explicit reasoning, so it must be initialized from the question alone (we use the previous token’s embedding by default). The mid block has a full prefix of generated reasoning, which gives four initialization options—each with different consequences for superposition, as we will show.

3.5 Why Initialization Matters

We next study a design choice that prior latent-reasoning methods have not analyzed: latent-block initialization. The same optimization procedure can produce a useful latent block or a degenerate one, and the difference is set at initialization. Before any RL update, the initial values of H_k^m already place the block somewhere on a spectrum from “superposed” (the block’s hidden states maintain a broad distribution over vocabulary tokens)

to “collapsed” (the block’s hidden states concentrate on a single token). Our analysis in Section 5 suggests that RL updates refine the block within the region established at initialization rather than reversing an initially collapsed representation.

We compare four mid-block initializations: `prev_tok` (the embedding of the preceding generated token), `text` (a short fixed phrase such as “Next step is”), `randn` (random vectors drawn from a standard normal distribution) and `placeholder` (a placeholder embedding designed to be generic). These initializations differ in how concentrated their starting distribution is and, as our analysis shows, in whether the resulting block enters useful attention patterns.

4 Experiments

4.1 Experimental Setup

Models. We evaluate MidThink on four 3B–14B models: Llama-3.2-3B-Instruct (Grattafiori et al., 2024), Ministral-3-8B-Reasoning-2512 (Liu et al., 2026a), Qwen3-8B, and Qwen3-14B (Yang et al., 2025). Using three model families provides an initial check on family-specific trends.

Datasets. The five benchmarks cover a range of mathematical difficulty. GSM8K (Cobbe et al., 2021) and ASDiv-Aug (Miao et al., 2020) are grade-school arithmetic tasks where baselines already perform well. They serve as a ceiling test to confirm that mid-stream insertion does not hurt on easy problems. MATH500 (Hendrycks et al., 2021; Lightman et al., 2024) contains competition-level problems demanding multi-step algebraic manipulation. AIME2024 (Hugging Face H4, 2024) and AIME2025 (Zhang and Math-AI Team, 2025) come from the American Invitational Mathematics Examination and represent the hardest tier. Accuracy on these two sets is far below that on the grade-school benchmarks for every method we compare, with Avg@8 at or below 25% throughout, so they are the most discriminative of the five.

Baselines and Metric. We compare against three baselines. Zero-Shot CoT prompts the model with “Let’s think step by step” (Kojima et al., 2022) and generates a full explicit reasoning trace. Soft-CoT (Xu et al., 2025), which we reproduce with the authors’ official code, feeds convex combinations of vocabulary embeddings as soft thought tokens through a lightweight projection module but does not optimize per instance. LTPO, the

closest baseline to our work, optimizes a start latent block via zeroth-order policy gradient with a confidence-based reward but does not insert blocks during generation.

We report both Best@ k and Avg@ k with $k = 8$. Best@ k counts a problem as solved if at least one of eight independent generations produces the correct answer, whereas Avg@ k measures the average accuracy across the same eight generations. All methods use the same number of output samples and the same maximum generation length; Appendix E reports their internal test-time costs.

4.2 Main Results

Table 2 reports results across four models and five benchmarks. The ablation row “Ours (w/o mid)” retains the start latent block and the rollout-based reward but disables the mid block; Sections 4.3 and 4.4 analyze the rollout reward and the choice of initialization separately.

Averaged over the five benchmarks, the full method attains the highest mean B@8 among training-free methods on all four models (59.22, 62.47, 70.88, and 72.32), and the highest overall on three. The exception is Ministral-3-8B-Reasoning, where SoftCoT, which requires offline training, reaches 63.09. Mean A@8 is highest on Ministral-3-8B-Reasoning and Qwen3-8B and tied with “Ours (w/o mid)” on Qwen3-14B, but on Llama-3.2-3B it falls to 40.87, below both SoftCoT (42.74) and “Ours (w/o mid)” (41.42).

The gains over the training-free baselines concentrate on AIME. On AIME2024 Best@8 the full method reaches 46.67% with both Qwen models, against 26.67% and 33.33% for LTPO, and 23.33% with Ministral-3-8B-Reasoning against 13.33%; with Llama-3.2-3B it matches LTPO at 26.67%. On AIME2025 it scores 10.00% with Llama-3.2-3B, where LTPO scores zero.

On MATH500 we see no comparable advantage: Best@8 differs from the strongest training-free baseline by at most 1.20 points in either direction, and Avg@8 is below it on all four models. MATH500 and both AIME sets have no training split and use each model’s GSM8K configuration rather than a tuned one. On the grade-school sets, Best@8 stays within about three points across methods apart from LTPO on Llama-3.2-3B, and the full method is best or tied-best on GSM8K across all four models.

Zero-Shot CoT scores 0.00 on AIME2024 with Ministral-3-8B-Reasoning, the only zero among

Model	Method	Benchmark					Mean B@8 / A@8
		GSM8K B@8 / A@8	MATH500 B@8 / A@8	AIME2024 B@8 / A@8	AIME2025 B@8 / A@8	ASDivAug B@8 / A@8	
Llama-3.2-3B-Instruct	Zero-shot CoT	90.98 / 65.09	<u>72.60</u> / 45.42	16.67 / 5.83	3.33 / <u>0.42</u>	94.32 / 83.38	55.58 / 40.03
	SoftCoT [†]	<u>91.66</u> / 74.65	69.60 / 44.02	26.67 / 5.42	<u>6.67</u> / 1.67	93.64 / 87.93	<u>57.65</u> / 42.74
	LTPO	82.56 / 59.86	62.20 / 30.18	26.67 / <u>5.42</u>	0.00 / 0.00	90.46 / 82.11	52.38 / 35.51
	Ours (w/o mid)	91.44 / <u>72.42</u>	<u>72.52</u> / <u>44.04</u>	<u>20.00</u> / <u>5.42</u>	0.00 / 0.00	<u>94.61</u> / <u>85.22</u>	<u>55.71</u> / <u>41.42</u>
	Ours	91.81 / 69.31	72.84 / 42.43	26.67 / 5.83	10.00 / 1.67	94.80 / 85.12	59.22 / 40.87
Ministral-3-8B-Reasoning-2512	Zero-shot CoT	94.84 / 83.85	75.80 / 62.60	0.00 / 0.00	3.33 / 0.42	93.55 / 89.55	53.50 / 47.28
	SoftCoT [†]	94.31 / 87.23	83.20 / <u>72.55</u>	23.33 / 8.33	20.00 / 8.33	<u>94.61</u> / 90.59	63.09 / 53.41
	LTPO	94.92 / 86.84	80.20 / 72.85	13.33 / <u>9.58</u>	13.33 / <u>7.50</u>	94.12 / 91.51	59.18 / <u>53.66</u>
	Ours (w/o mid)	94.68 / <u>89.69</u>	81.40 / 71.65	<u>20.00</u> / 7.50	<u>16.67</u> / <u>7.50</u>	94.52 / 90.41	61.45 / 53.35
	Ours	94.92 / 90.99	<u>82.60</u> / <u>72.47</u>	23.33 / 10.00	<u>16.67</u> / <u>7.50</u>	94.83 / <u>90.85</u>	<u>62.47</u> / 54.36
Qwen3-8B	Zero-shot CoT	94.84 / 89.80	83.00 / 65.88	20.00 / 7.50	13.33 / 3.75	93.26 / 91.38	60.89 / 51.66
	SoftCoT [†]	95.07 / 89.56	91.40 / <u>79.53</u>	<u>40.00</u> / 21.25	30.00 / <u>15.42</u>	96.05 / 93.50	<u>70.50</u> / 59.85
	LTPO	96.06 / 92.98	<u>88.80</u> / 79.57	26.67 / 17.08	20.00 / 15.83	95.47 / 91.03	65.40 / 59.30
	Ours (w/o mid)	<u>96.72</u> / <u>94.06</u>	85.93 / 76.15	30.00 / <u>22.00</u>	20.00 / 15.33	95.47 / 92.73	65.62 / <u>60.05</u>
	Ours	97.22 / 94.14	88.12 / 76.64	46.67 / 22.08	<u>26.67</u> / 15.41	<u>95.70</u> / <u>92.81</u>	70.88 / 60.22
Qwen3-14B	Zero-shot CoT	<u>95.91</u> / 91.76	85.60 / 71.47	16.67 / 10.00	16.67 / 8.75	95.09 / 93.12	61.99 / 55.02
	SoftCoT [†]	95.15 / 91.04	91.60 / <u>82.17</u>	40.00 / 22.08	<u>26.67</u> / <u>18.75</u>	<u>96.92</u> / 93.99	<u>70.07</u> / 61.61
	LTPO	95.07 / <u>92.29</u>	91.00 / 83.25	33.33 / 21.67	<u>26.67</u> / 19.17	96.63 / 83.77	68.54 / 60.03
	Ours (w/o mid)	95.83 / 93.26	89.68 / 81.09	36.67 / <u>22.92</u>	30.00 / <u>18.75</u>	96.54 / <u>93.70</u>	69.74 / 61.94
	Ours	96.91 / 91.51	<u>91.02</u> / 81.28	46.67 / 25.00	30.00 / 19.17	96.98 / 92.72	72.32 / 61.94

Table 2: Main results across mathematical reasoning benchmarks. Each cell reports B@8 / A@8: B@8 counts a problem correct if any of 8 samples is correct, and A@8 averages accuracy over those samples. For each model, benchmark, and metric, **bold** marks the best score and underline the second-best. “Ours (w/o mid)” disables mid-stream insertion to isolate the rollout reward. [†]SoftCoT requires offline training; all other methods are training-free.

the Zero-Shot CoT rows, from truncation rather than reasoning failure. The model is reasoning-tuned and its default trace exceeds the shared 4096-token budget on all 30 problems. Latent insertion shortens the trace enough to reach an answer, the mechanism we examine in Appendix D.2.

Comparing “Ours (w/o mid)” with the full method shows where the mid block contributes, and its effect is concentrated rather than uniform. Best@8 on AIME2024 rises from 20.00 to 26.67 (Llama-3.2-3B), 20.00 to 23.33 (Ministral-3-8B-Reasoning), 30.00 to 46.67 (Qwen3-8B), and 36.67 to 46.67 (Qwen3-14B). On AIME2025 it adds 10.00 points on Llama-3.2-3B and 6.67 on Qwen3-8B and leaves the other two unchanged, which for Ministral-3-8B-Reasoning we attribute to the model already producing extended reasoning traces without insertion. MATH500 rises on all four models by 0.32 to 2.19 points and the grade-school sets by at most 1.08, so mid-stream insertion does not hurt where explicit reasoning already suffices. Avg@8 changes are not uniformly positive, the largest decrease being 3.11 points on GSM8K with Llama-3.2-3B.

4.3 Ablation: Rollout Reward

Setup. We isolate the effect of the rollout-based reward by varying two parameters: β , which con-

trols the weight of the rollout component relative to the latent-position confidence used in LTPO, and the number of rollout steps. Setting $\beta = 0$ recovers the LTPO reward; increasing β toward 1.0 shifts the signal entirely toward future-token confidence. We run this ablation on Llama-3.2-3B-Instruct using all 30 AIME2024 problems and 200 MATH500 problems. Because the ablation sweeps a grid, we report Avg@5 to keep compute manageable.¹

Results. The main-table ablation already isolates this reward: comparing LTPO with “Ours (w/o mid)”, which differ only in reward design, the rollout reward alone improves the mean B@8 on all four models (+3.33, +2.27, +0.22, and +1.20), and the mean A@8 on three of the four (+5.91 on Llama-3.2-3B, +0.75 on Qwen3-8B, and +1.91 on Qwen3-14B); on Ministral-3-8B-Reasoning the mean A@8 is essentially unchanged (−0.31). The β and rollout-depth sweep (Figure 2) confirms the same direction: accuracy rises with both, reaching 12.0% on AIME at $\beta = 1.0$ with 32 rollout steps—more than double the 6.0% of the LTPO reward ($\beta = 0$). MATH500 shows the same trend at smaller magnitude, since the baseline is already strong there. The gain from looking beyond the

¹Ministral-3-8B-Reasoning-2512 was added after the main experiments; the ablations in Sections 4.3 and 4.4 and the analysis in Section 5 are run on Llama-3.2-3B-Instruct only.

latent positions is therefore largest where downstream reasoning matters most.

4.4 Ablation: Initialization Method

Setup. We compare four initialization strategies for both the start block and the mid block: `prev_tok` (embedding of the preceding token), `text` (a fixed phrase such as “First/Next step is”), `randn` (random vectors drawn from a standard normal distribution), and `placeholder` (a generic placeholder embedding, e.g., `|endoftext|` or `|reserved_special_token_{i}|`). Model and datasets match Section 4.3.

Start block. For the start block, initialization has little effect: all four strategies fall within ~ 2 points on MATH500, with `prev_tok` marginally best. Full per-dataset results appear in Appendix F.

Mid block. Mid-block initialization is substantially more consequential. With the start block fixed to `prev_tok`, `placeholder` performs best on both datasets, `randn` ranks second, and `prev_tok` and `text` fall well below—on AIME2024 to roughly a third of placeholder. Full per-dataset numbers appear in Appendix F. The two strategies that anchored the latent block to a concrete token—which worked well or acceptably for the start block—now perform worst in the mid block. This sensitivity is the central puzzle of our analysis: the ranking inverts between the two positions. `prev_tok` is the best start-block initialization on both datasets yet the weakest mid-block one on AIME2024, while `placeholder` is the reverse. On MATH500 the spread also widens sharply, from about two points at the start to nearly fifteen mid-stream. Section 5 explains why.

4.5 Computational Cost

Overhead. The additional cost over LTPO comes from two sources: the rollout steps during reward computation and the optimization loop for each mid block. We measure estimated FLOPs on Qwen3-8B; a detailed breakdown appears in Appendix D. Although mid-stream insertion shortens the explicit reasoning trace—to roughly 68% of the Zero-Shot CoT length on Qwen3-8B/AIME—the rollout-based optimization dominates in a pure-FLOPs accounting. MidThink exchanges decoded sequence length for test-time compute. The overhead scales linearly with the trigger count C and is tunable through the rollout hyperparameters; Appendix E reports measured costs under an accounting that

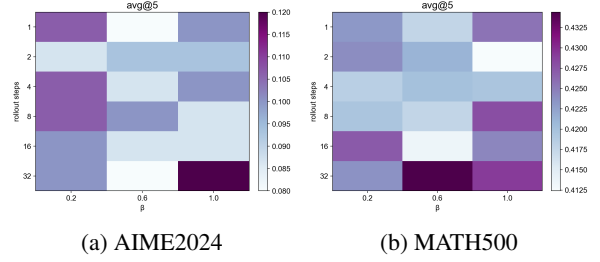


Figure 2: Ablation results on rollout settings. Performance tends to improve toward the bottom right, indicated by darker purple.

registers input, cached, and output tokens across all calls, where a lighter setting improves accuracy while keeping cost at or below CoT.

5 Analysis

We examine how initialization affects the latent block’s layerwise distribution, attention pattern, and response to RL updates. We support this analysis with three observations and one case study. The three observations below use Llama-3.2-3B-Instruct on all 30 AIME2024 and 200 GSM8k and MATH500 problems.² The case study instead examines a representative Qwen3-14B trace, where the longer reasoning chains make the effect of mid-stream insertion more visible.

5.1 Observation 1: Initialization Sets the Per-Layer Distribution

We probe the latent block at each transformer layer using Logit Lens (nostalgebraist, 2020), computing two quantities: the Shannon entropy of the top- k predicted tokens and cosine similarity to the nearest vocabulary embedding.

Top- k entropy. Figure 3 shows the layerwise top- k entropy of the mid latent block. The anchored initializations (`prev_tok`, `text`), starting from a deterministic token, stay low-entropy across the network and never recover a broad distribution. By contrast, `placeholder` and `randn` sustain high entropy from the early through the mid layers—the depth where reasoning is organized—and concentrate onto a single token only in the final layers, the profile expected of a block performing continuous computation.

Cosine similarity. For each layer, we compute the maximum cosine similarity between the block’s

²Ministral-3-8B-Reasoning-2512 was added after the main experiments; the ablations of Section 4.3 and 4.4 and the analysis in this section are run on Llama-3.2-3B-Instruct only.

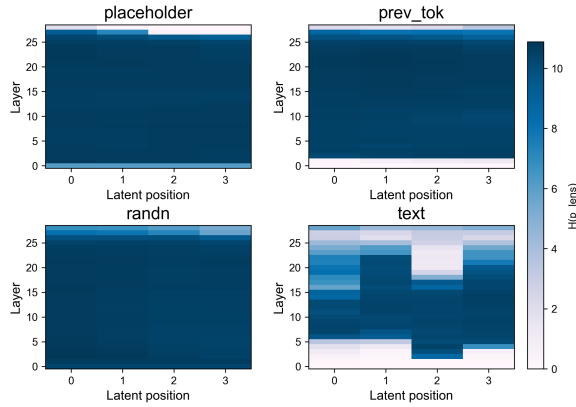


Figure 3: Layerwise top- k entropy of the mid latent block under the four initialization strategies. Blue indicates high entropy, purple(white) low entropy.

hidden state and any vocabulary embedding. High similarity to a vocabulary embedding is a geometric proxy for collapse toward a discrete token. The pattern matches the entropy result: mid-block `prev_tok` approaches the nearest token at most layers, while `placeholder` stays further away. We defer the full layerwise heatmap to Appendix G.

5.2 Observation 2: Distribution Shapes Attention Pattern

Attention centroid. If the block holds a distribution rather than a single token, what does the network do with it? We adapt the attention-centroid notion of Chen et al. (2026) to measure position within the latent block.

The centroid measures, on a $[0, 1]$ scale, where within the latent block attention concentrates (Figure 4). In mid-layers (L8–L16) `randn` and `placeholder` place the centroid near 0.35, meaning attention focuses on the front of the block while `text` and `prev_tok` place it near 0.50, meaning attention spreads uniformly. A front-loaded centroid is consistent with the block being read as an ordered intermediate representation, whereas uniform attention indicates the model treats each position as an independent token without internal structure. The separation between initializations is more pronounced for the mid block than for the start block, and the initializations with a front-loaded mid-block centroid are also the more accurate ones in Section 4.4.

Attention magnitude. The magnitude profiles of the start and mid blocks diverge. The start block is consumed in the early layers regardless of initialization and then ignored. The mid block is not mono-

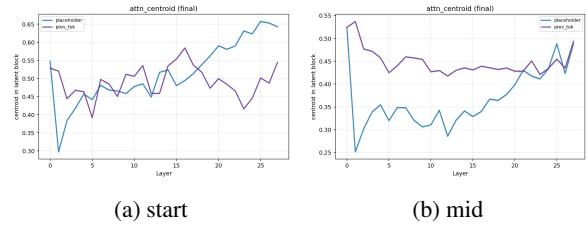


Figure 4: Attention centroid of start/mid block. 0=front side of the block, 1=back side of the block

tone: under `placeholder` and `randn` the magnitude rebounds through the mid layers (L8–L16), so later positions keep reading the block at the depth where reasoning is organized, whereas `prev_tok` and `text` are read only near read-out. This mid-layer rebound co-occurs with higher accuracy, indicating the block is consulted while reasoning is still being shaped rather than only at the answer stage. Full curves appear in Appendix H.

5.3 Observation 3: RL Cannot Escape What Initialization Sets

The previous two observations describe the block’s state at initialization. RL applies updates on top of this state. Does it move the block much?

We track whether the top-1 token at each layer changes between the initial state and the final RL state. Figure 5 shows top-1 token stability across layers for each initialization, with blue indicating preservation and purple indicating flips under RL updates. Two things stand out. First, the change is not uniform across depth, and its profile differs by initialization. Under `placeholder` the top-1 token is increasingly preserved toward the deeper layers, so RL leaves the early layers freer to move while the read-out end stabilizes. Under `prev_tok` the identity is largely unchanged at every depth: the block starts on a single token and RL does not move it off. `text` behaves like `prev_tok`, and `randn` shows neither the stability of the anchored initializations nor the depthwise gradient of `placeholder`. This is the distinction that the cosine geometry of Appendix G and the magnitude profile of Appendix H cannot resolve on their own, since both group `placeholder` with `randn`.

Second, and more consequentially, the change is small in magnitude for every initialization—the layerwise distribution at the final RL state closely tracks the initial one. RL refines the answer probabilities at the margin; it does not redesign the block. A good initialization is therefore not a starting point

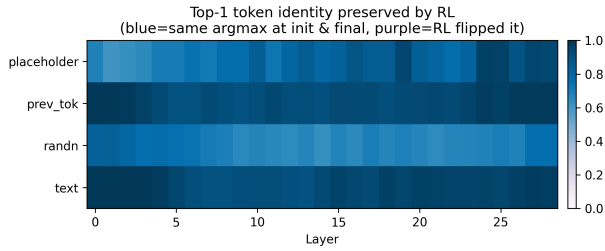


Figure 5: Top-1 token identity across layers before and after RL updates, per initialization. Darker blue indicates the argmax token is preserved; lighter shades indicate flips.

that RL improves upon but a starting region that RL fine-tunes within, and a bad initialization places the block in a region where RL’s refinements cannot recover the lost capacity.

5.4 Initialization Principles

The three observations converge on four rules for choosing an initialization, state here so that the analysis transfers to settings we did not test.

P1. Do not anchor a mid block to a committed token. An initialization copied from preceding text starts on a single embedding direction and stays there (Section 5.1, Appendix G), and RL does not move it off (Section 5.3). `prev_tok` and `text` fall to at most roughly a third of `placeholder`’s accuracy on AIME2024 for this reason.

P2. The same rule does not apply at the start. With no preceding reasoning to copy, `prev_tok` reduces to the question’s final token and does not collapse (Appendix G). The start block is also consumed in the early layers regardless of initialization (Appendix H), so P1’s mechanism has less room to operate. We default to `prev_tok`, which is best on both datasets and the simplest to construct.

P3. Prefer a generic embedding over a random vector. Both preserve the distributional spread that P1 asks for, but they differ under RL: `placeholder` shows a depthwise gradient in top-1 stability while `randn` shows neither that gradient nor the stability of the anchored initializations (Section 5.3). `placeholder` is best on both datasets and `randn` second (Appendix F).

P4. Screen candidates before running the full benchmark. Layerwise top- k entropy and maximum cosine similarity can be read from a handful of forward passes and separate the anchored from the unanchored initializations without any accuracy measurement. They are necessary but not sufficient conditions, so they narrow the candidate set rather than rank it (Section 5.1 and Appendix G).

5.5 Case Study: How Mid-Stream Insertion Refines Reasoning

To make the preceding observations concrete, we examine a Qwen3-14B AIME2024 generation in which the confidence trigger fires, comparing our method against two failing Zero-Shot CoT samples on the same problem (traces in Appendix I). The two Zero-Shot CoT failures diverge at different points. In the first (Appendix Figure 11), the trace *drifts*: it never commits at the reciprocal identity, introducing an extra variable and pursuing a harder equation rather than reaching the cross-multiplication. In the second (Appendix Figure 12), it reaches the same relation as MidThink, $xy = 25$, then *loops*, opening a verification branch (“Wait, so is that it?”) and re-substituting the solution without ever re-emitting the derived value. Both share a mechanism: whether before or after a correct derivation, uncertainty is not resolved but *prolonged*, inflating the trace until the token budget is exhausted and the answer is never re-emitted.

With the mid latent block inserted, the refined hidden state lets explicit decoding commit to the derived value and proceed to the answer. In this trace the mid block keeps the reasoning from straying onto an unproductive path, curbing both the drift and the redundant verification that uncertainty would otherwise trigger. This is consistent with Section 5.3 and with the shortened explicit traces measured in Appendix D.2: mid-stream insertion lets the trace terminate without redundant deliberation, one plausible mechanism for its gains on the hardest benchmarks.

6 Conclusion

Latent reasoning has appealed to researchers for its expressive potential, but training-free methods have struggled to realize it: start-only blocks miss mid-stream uncertainty, and mode-switching schemes lack optimization. We show that a confidence-triggered mid block, a rollout-based reward refining both blocks, and initializations that preserve superposition together deliver consistent gains over strong baselines, largest on the hardest benchmarks. The mechanistic story is simple: initialization sets the distribution, the distribution shapes attention, and RL acts within the region initialization has chosen. Section 5.4 turns this into four rules for selecting an initialization in settings we did not test. Latent reasoning works when the latent state is given room to be continuous.

Limitations

Confidence as a proxy for correctness. Our rollout-based reward and confidence trigger both rely on the model’s own predictive distribution as a signal of reasoning quality. This is a useful proxy but not a perfect one. A model can be confidently wrong: when its prior is biased toward an incorrect answer, the rollout reward will reinforce that bias, and the confidence trigger will fail to fire even when the reasoning is drifting. Reward signals that combine confidence with external verification or self-consistency could address this, at the cost of additional compute.

Generality across reasoning domains. We evaluate exclusively on mathematical reasoning benchmarks. The mechanism by which mid-stream insertion helps is plausibly relevant to other domains that require multi-step deduction (code, scientific reasoning, formal logic), but we have not tested this. Domains in which uncertainty signals are dominated by genuine ambiguity rather than reasoning errors (open-ended generation, dialogue) may not benefit from the same trigger design.

Acknowledgements

This work was supported in part by the National Research Foundation of Korea [NRF-2023R1A2C3004176], the Ministry of Health & Welfare, Republic of Korea [HR20C002103], the Ministry of Science and ICT (MSIT) [RS-2023-00262002], the Institute of Information & Communications Technology Planning & Evaluation(IITP)-ICT Creative Consilience Program grant funded by the Korea government(MSIT)(IITP-2026-RS-2020-II201819), the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT and MOE) (No. RS-2025-16652968), and Development of storytelling AI technology for cultural heritage tailored to the various interests of users [RS-2023-00220195].

References

Natasha Butt, Ariel Kwiatkowski, Ismail Labiad, Julia Kempe, and Yann Ollivier. 2025. Soft tokens, hard truths. *arXiv preprint arXiv:2509.19170*.

Haoyang Chen, Yi Liu, Jianzhi Shao, Tao Zhang, Chengfu Huo, and Wei Hu. 2026. How do answer tokens read reasoning traces? self-reading patterns in thinking llms for quantitative reasoning. *arXiv preprint arXiv:2604.19149*.

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

Yuntian Deng, Kiran Prasad, Roland Fernandez, Paul Smolensky, Vishrav Chaudhary, and Stuart Shieber. 2023. Implicit chain of thought reasoning via knowledge distillation. *arXiv preprint arXiv:2311.01460*.

Renyu Fu and Guibo Luo. 2026. Selar: Selective latent reasoning in large language models. *arXiv preprint arXiv:2604.08299*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

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

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Hugging Face H4. 2024. AIME 2024. https://huggingface.co/datasets/HuggingFaceH4/aime_2024. Hugging Face dataset, accessed: 2026-05-24.

Ishan Jindal, Sai Prashanth Akuthota, Jayant Taneja, and Sachin Dev Sharma. 2026. The path of least resistance: Guiding llm reasoning trajectories with prefix consensus. *International Conference on Learning Representations*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

Hengli Li, Chenxi Li, Tong Wu, Xuekai Zhu, Yuxuan Wang, Zhaoxin Yu, Eric Hanchen Jiang, Song-Chun Zhu, Zixia Jia, Ying Nian Wu, et al. 2025. Seek in the dark: Reasoning via test-time instance-level policy gradient in latent space. *arXiv preprint arXiv:2505.13308*.

Xuan Li, Yining Wang, Yuchen Liu, GuanJun Liu, Delai Qiu, Shengping Liu, Jiaen Liang, Wei Huang, Jun Yu, and Junnan Zhu. 2026. Later: Efficient test-time reasoning via latent exploration and explicit verification. *arXiv preprint arXiv:2605.07315*.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe.

2024. Let’s verify step by step. In *International Conference on Learning Representations*, volume 2024, pages 39578–39601.
- Alexander H Liu, Kartik Khandelwal, Sandeep Subramanian, Victor Jouault, Abhinav Rastogi, Adrien Sadé, Alan Jeffares, Albert Jiang, Alexandre Cahill, Alexandre Gavaudan, et al. 2026a. Ministral 3. *arXiv preprint arXiv:2601.08584*.
- Xiang Liu, Xuming Hu, Xiaowen Chu, and Eunsol Choi. 2026b. Diffadapt: Difficulty-adaptive reasoning for token-efficient llm inference. *International Conference on Learning Representations*.
- Shen-Yun Miao, Chao-Chun Liang, and Keh-Yih Su. 2020. A diverse corpus for evaluating and developing english math word problem solvers. In *Proceedings of the 58th annual meeting of the Association for Computational Linguistics*, pages 975–984.
- nostalgebraist. 2020. [Interpreting gpt: the logit lens](#).
- Michael Rizvi-Martel, Guillaume Rabusseau, and Marius Mosbach. 2026. The illusion of superposition? a principled analysis of latent thinking in language models. *arXiv e-prints*, pages arXiv–2604.
- Dachuan Shi, Abedelkadir Asi, Keying Li, Xiangchi Yuan, Leyan Pan, Wenke Lee, and Wen Xiao. 2026. Swireasoning: Switch-thinking in latent and explicit for pareto-superior reasoning llms. *International Conference on Learning Representations*.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. *International Conference on Learning Representations*.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. [Chain-of-thought prompting elicits reasoning in large language models](#). In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc.
- Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256.
- Yige Xu, Xu Guo, Zhiwei Zeng, and Chunyan Miao. 2025. Softcot: Soft chain-of-thought for efficient reasoning with llms. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23336–23351.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.
- Wengao Ye, Yan Liang, and Lianlei Shan. 2026. Thinking on the fly: Test-time reasoning enhancement via latent thought policy optimization. *International Conference on Learning Representations*.
- Yifan Zhang and Math-AI Team. 2025. American Invitational Mathematics Examination (AIME) 2025. <https://huggingface.co/datasets/math-ai/aime25>. Hugging Face dataset, accessed: 2026-05-24.
- Zhen Zhang, Xuehai He, Weixiang Yan, Ao Shen, Chenyang Zhao, and Xin Wang. 2026. Soft thinking: Unlocking the reasoning potential of llms in continuous concept space. *Advances in Neural Information Processing Systems*, 38:168990–169012.
- Hanlin Zhu, Shibo Hao, Zhiting Hu, Jiantao Jiao, Stuart J Russell, and Yuandong Tian. 2026. Reasoning by superposition: A theoretical perspective on chain of continuous thought. *Advances in Neural Information Processing Systems*, 38:79931–79963.

A Baseline Reproduction Details

This appendix documents how each baseline in Table 2 was implemented, so that our comparisons can be reproduced. All baselines were evaluated using the same protocol as MidThink, with identical benchmark problems, the same number of output samples per problem ($k = 8$), and a maximum explicit generation length of 4096 tokens. Differences in internal test-time compute across methods are reported separately in Appendices D and E.

A.1 Zero-Shot CoT

Zero-Shot CoT prompts the model with “Let’s think step by step” (Kojima et al., 2022) and decodes a full explicit reasoning trace followed by an answer. We use the following decoding settings: sampling with temperature 0.7 and top-p 0.95.

A.2 SoftCoT

We reproduce SoftCoT (Xu et al., 2025) using the authors’ official implementation.³ SoftCoT produces soft thought tokens as convex combinations of vocabulary embeddings through a lightweight projection module; the module is trained offline and applied without per-instance optimization at test time. Following SoftCoT’s pairing scheme, we pair each base model with a smaller assistant from the same family: Llama-3.2-3B with Llama-3.2-1B, Ministral-3-8B-Reasoning-2512 with Ministral-3-3B-Instruct-2512, and Qwen3-8B and Qwen3-14B both with Qwen3-1.7B.⁴ For GSM8K and ASDiv-Aug, which provide a dedicated training split, we train the projection on the corresponding split. Since MATH500, AIME2024, and AIME2025 have no training split, we reuse the GSM8K-trained projection on these benchmarks without further tuning.

A.3 LTPO

We reproduce LTPO (Ye et al., 2026) using the authors’ official implementation.⁵ LTPO is the closest baseline to MidThink: it is training-free

³<https://github.com/xuyige/SoftCoT>

⁴For Ministral we pair with the instruction-tuned rather than the reasoning-tuned assistant. SoftCoT’s assistant is not used for generation: it runs a single forward pass and its last-layer hidden states are fed to the projection module. Reasoning tuning affects the model’s behavior over a long decoded trace, so it confers no advantage in a single forward pass. This also matches the other pairings, where Qwen3-1.7B is run with thinking disabled and Llama-3.2-1B-Instruct is not reasoning-tuned.

⁵<https://github.com/ltpo2025/LTPO>

and optimizes a latent block at test time, but inserts that block only at the start of generation, using a confidence-based reward evaluated at the latent positions in a single forward pass, with no rollout. LTPO depends on several test-time hyperparameters, including the number of optimization steps, top- k , the exploration scale, and the learning rate. For Llama-3.2-3B-Instruct and Qwen3-14B, both of which are evaluated in the original LTPO paper (Ye et al., 2026), we use the per-benchmark hyperparameters released by the authors for all five benchmarks. For Qwen3-8B and Ministral-3-8B-Reasoning-2512, which the original paper does not cover, we determine the hyperparameters ourselves via grid search: for GSM8K and ASDiv-Aug we search over a set of 200 examples sampled from the training data and select the configuration with the highest accuracy, and for MATH500, AIME2024, and AIME2025, which have no training split, we reuse this model’s GSM8K configuration.

B Hyperparameter Configuration

Table A lists the full hyperparameter configuration used in all main experiments.

C Sensitivity Analysis

We analyze the sensitivity of **MidThink** to its main hyperparameters on the training splits of GSM8K and ASDiv-Aug—the same splits used for hyperparameter selection. For each hyperparameter we hold all others fixed and sweep the parameter of interest, aggregating accuracy over n runs that vary the random seed and the remaining configuration. Table B reports the mean and standard deviation for each value. Because this sweep is performed on the training splits, it reflects the selection procedure itself rather than test performance; test results are reported in the main paper.

Robust hyperparameters. Accuracy is essentially unchanged across the swept values of the top- k truncation, the start and mid block sizes, and the rollout mixing coefficient β . On both splits the mean accuracy varies by less than one point between settings—for example, top- $k \in \{5, 10\}$ differs by 0.5 points on GSM8K and 0.1 on ASDiv-Aug. We therefore fix these globally.

Hyperparameters with a consistent trend. Two hyperparameters show a consistent, interpretable trend on both splits. (i) The dwell length d : a longer window ($d=128$) outperforms a shorter one ($d=64$) by 6.0 points on GSM8K and 4.6 on

Shared across all models and benchmarks				
Confidence trigger	top- k	10		
	dwel length d	128		
	EMA coefficient α	0.05		
Latent blocks	start block size K_s	4		
Optimization	variance decay γ	0.95		
	optimization steps S	20		
	start rollout length r_s	4		
	mid rollout length r_m	32		
	rollout mixing β	1.0		
Tuned per model on the training split				
Model	Set	η	σ_0	K_m
Llama-3.2-3B Instruct	GSM8K [†]	0.0005	20	4
	ASDiv-Aug	0.05	5	4
Ministral-3-8B Reasoning-2512	GSM8K [†]	0.05	5	4
	ASDiv-Aug	0.05	5	4
Qwen3-8B	GSM8K [†]	0.05	20	4
	ASDiv-Aug	0.05	20	4
Qwen3-14B	GSM8K [†]	0.005	20	8
	ASDiv-Aug	0.005	20	4

[†]Applied to MATH500, AIME2024, and AIME2025 without further tuning.

Table A: Hyperparameter configuration. Shared values are fixed across all models and benchmarks. Tuned configurations are selected on GSM8K and ASDiv-Aug training splits; each model’s GSM8K configuration is reused for MATH500, AIME2024, and AIME2025.

ASDiv-Aug, and lowers the standard deviation on both ($0.057 \rightarrow 0.034$ on GSM8K, $0.026 \rightarrow 0.013$ on ASDiv-Aug). A short dwell admits transient entropy spikes as triggers; a longer dwell enforces the *sustained*-uncertainty criterion the trigger is designed for. (ii) Accuracy declines monotonically as η grows on both splits; the effect is pronounced on GSM8K ($0.871 \rightarrow 0.793$) and milder but directionally identical on ASDiv-Aug ($0.829 \rightarrow 0.811$). Large steps overshoot the local region that zeroth-order optimization can reliably exploit.

Benchmark-dependent hyperparameters.

The exploration scale σ_0 has a noticeable effect on GSM8K (+3.8 points for $\sigma_0=20$ over 5) but is essentially flat on ASDiv-Aug (82.0 at both values). This benchmark-dependence is precisely why σ_0 is among the hyperparameters we select per model and per tuning set.

Across all swept hyperparameters, accuracy varies smoothly rather than exhibiting sharp peaks. This indicates that hyperparameter selection on the training split does not require fine-grained

Hyperparameter	Value	n	GSM8K	ASDiv-Aug
top- k	5	96	83.2 ± 5.7	82.1 ± 3.0
	10	96	83.7 ± 5.3	82.0 ± 3.2
dwel length d	64	96	80.4 ± 5.7	79.7 ± 2.6
	128	96	86.4 ± 3.4	84.3 ± 1.3
K_s	4	96	83.7 ± 5.8	81.1 ± 2.8
	8	96	83.1 ± 5.3	82.9 ± 3.1
K_m	4	96	84.0 ± 4.7	82.4 ± 2.9
	8	96	82.9 ± 6.3	81.7 ± 3.3
learning rate η	5×10^{-4}	64	87.1 ± 2.2	82.9 ± 2.3
	5×10^{-3}	64	84.0 ± 5.1	82.1 ± 2.7
	5×10^{-2}	64	79.3 ± 5.5	81.1 ± 3.9
σ_0	5	96	81.5 ± 6.0	82.0 ± 3.3
	20	96	85.4 ± 4.2	82.0 ± 2.9
rollout mixing β	0.8	96	83.5 ± 5.5	82.1 ± 3.0
	1.0	96	83.4 ± 5.6	82.0 ± 3.2

Table B: Sensitivity analysis on GSM8K and ASDiv-Aug. Each block sweeps one hyperparameter while fixing all others at the default configuration in Table A. Results report accuracy as mean $\pm$ standard deviation over n runs; default values are in **bold**.

search, and supports our transfer setting, in which each model’s GSM8K configuration is applied unchanged to MATH500, AIME2024, and AIME2025.

D Computational Cost Analysis

We provide a theoretical FLOPs estimate for **Mid-Think** and compare it against Zero-Shot CoT, Soft-CoT, and LTPO. Following standard practice, we approximate the forward-pass cost of a transformer with N non-embedding parameters as $2N$ FLOPs per token, and omit the attention term, which is subdominant in the regime where the sequence length is small relative to the hidden width. All costs are reported per problem instance. The number of answer tokens T_a is common to every method and negligible relative to the explicit reasoning length ($T_a \ll T_r$); we therefore absorb it into the reasoning term and drop it from the expressions below.⁶

Notation. Let N be the number of model parameters and d the hidden dimension. We write T_r for the number of explicit reasoning tokens in a Zero-Shot CoT trace and T_a for the number of answer tokens. For methods that insert latent blocks, K_s denotes the start-block size and K_m the mid-block

⁶On AIME with Qwen3-8B the answer span is a few tokens against a multi-thousand token reasoning trace, so $(T_r + T_a) \approx T_r$. Because T_a enters every method identically, retaining it would not change the relative overhead.

size. The online optimization runs for S steps, and each rollout advances r_s tokens internally for the start block and r_m for each mid block. Finally, C is the number of mid latent blocks inserted in a generation, i.e., the number of times the confidence trigger fires.

D.1 Per-Method Cost

Zero-Shot CoT. The model decodes T_r reasoning tokens (answer tokens absorbed as above) with no latent computation:

$$\text{FLOPs}_{\text{CoT}} = 2NT_r = O(T_r N). \quad (5)$$

Since $T_r \gg T_a$ on multi-step problems, the explicit reasoning trace dominates the cost.

SoftCoT. SoftCoT prepends K_s soft thought tokens produced by a trained projection module. Excluding the one-time *offline* training of that module, the per-instance cost is

$$\text{FLOPs}_{\text{SoftCoT}} = \underbrace{2d^2 K_s}_{\text{projection}} + 2N(K_s + T_r). \quad (6)$$

The projection term is a small constant; the per-instance cost is therefore essentially that of CoT plus a fixed overhead.

LTPO. LTPO optimizes a single start latent block of size K_s for S steps before generation begins, then decodes explicitly. Because its reward scores only the latent positions in a single forward pass, each optimization step costs $K_s \cdot 2N$:

$$\text{FLOPs}_{\text{LTPO}} = 2N(SK_s + K_s + T'_r). \quad (7)$$

Here T'_r is the length of the explicit trace produced *after* latent optimization. The optimized start block empirically shortens the trace; we measure $T'_r \approx 0.91 T_r$ on Qwen3-8B (Appendix D.2).

MidThink (Ours). Our method differs from LTPO in two ways. First, the reward is rollout-based, so each of the S optimization steps additionally advances r_s (start block) or r_m (mid block) tokens internally rather than scoring only the latent positions. Second, a mid latent block of size K_m is inserted and optimized whenever the confidence trigger fires, which occurs C times per instance. Writing this out,

$$\text{FLOPs}_{\text{Ours}} = 2N \left(S r_s K_s + \boxed{C S r_m K_m} + K_s + \boxed{C K_m} + T''_r \right), \quad (8)$$

The two boxed terms are the cost **MidThink** adds over LTPO: the optimization of the C inserted mid blocks, and the forward-pass cost of those blocks during final decoding. The quantity T''_r is the explicit-trace length under **MidThink**; because mid-stream insertion changes where and how the trace continues, T''_r cannot be inferred from T_r or T'_r and must be measured directly.

Remark. Comparing $\text{FLOPs}_{\text{LTPO}}$ and $\text{FLOPs}_{\text{Ours}}$, the start-block optimization term grows by a factor of r_s (the rollout length) because the rollout reward looks beyond the latent positions. This is the price of the more reliable optimization signal documented in Section 4.3. The mid-block term $C S r_m K_m \cdot 2N$ scales linearly with the number of triggers C , which is therefore the main driver of the overhead.

D.2 Token-Count Measurement: Qwen3-8B on AIME

To turn Equation (8) into a concrete number we measure the token-count quantities on Qwen3-8B over the AIME2024 and AIME2025 problems. For each method we record the mean explicit-reasoning length and, for **MidThink**, the mean number of triggered mid blocks C . All values are averaged over the 8 samples per problem drawn under the shared 4096-token budget.

Method	Reasoning tokens	C	Ratio to T_r
Zero-Shot CoT	$T_r = 4094$	–	1.00
LTPO	$T'_r = 3728$	–	≈ 0.91
Ours	$T''_r = 2768$	4	≈ 0.68

Table C: Measured average explicit-reasoning token counts on AIME (2024 + 2025) for Qwen3-8B. Answer tokens are omitted: they are common to all methods and negligible relative to T_r .

Mid-stream insertion shortens the explicit trace to $T''_r \approx 0.68 T_r$, so the decoding term falls from $T_r \cdot 2N \approx 65.5$ TFLOPs for CoT to about 44.3 TFLOPs. The rollout optimization, however, adds a term scaling as $C S r_m K_m$, which under the configuration of Appendix B outweighs that saving. MidThink therefore exchanges decoded sequence length for test-time compute: the shorter trace helps wall-clock latency under memory-bandwidth-bound decoding, but in a pure-FLOPs accounting the rollout dominates. The overhead scales linearly in C , S , and r_m , all of which are hyperparameters; Appendix E reports measured costs under a

lighter setting, together with an accounting that registers input, cached, and output tokens across all calls. The shortened trace also reduces truncation. Under the shared 4096-token budget, 53.96% of Zero-Shot CoT generations on AIME exceed the budget and are cut off before an answer is emitted, against 22.71% for SoftCoT, which decodes a shorter trace by construction.

E Measured Accuracy and Token Cost

Appendix D reports a theoretical FLOPs estimate that fixes the trigger count at its configured maximum and is therefore an upper bound. This appendix reports measured costs instead, under an accounting that registers every token of every call rather than only the decoded output.

Aggregate rule. Following commercial per-token pricing, which charges input, cached input, and output at different rates, we score each generation as

$$c = p_{\text{in}}(n_{\text{in}} - n_{\text{cache}}) + p_{\text{cache}}n_{\text{cache}} + p_{\text{out}}n_{\text{out}}, \quad (9)$$

with $(p_{\text{in}}, p_{\text{cache}}, p_{\text{out}}) = (1.25, 0.1, 5.0)$ USD per million tokens, the published rates for a representative small serving model.⁷ The rate structure, not the absolute prices, is what matters: output tokens dominate, so a method that shortens the explicit trace is rewarded even when it performs additional internal computation.

Rollout tokens are generated internally to score the reward and are never returned to the caller. Each rollout step advances against the KV cache of the preceding context and the resulting tokens are discarded, so rollout cost enters Equation (9) through n_{cache} rather than n_{out} .

A lighter setting. The rollout overhead is set by S , r_m , and the number of permitted mid-block insertions, all of which are hyperparameters. We evaluate a light setting v1 ($S=5$, $r_m=8$, at most two insertions) on the two larger benchmarks with Qwen3-8B.

Results. Table D reports the measurements. Two points follow.

First, the accuracy gains do not come from spending more tokens. On MATH500, v1 exceeds LTPO on both metrics (B@8 89.58 vs 88.80, A@8 80.16 vs 79.57) while decoding fewer tokens

Benchmark	Method	B@8	A@8	Out. tok.	TFLOPs	×CoT
GSM8K	Zero-Shot CoT	94.84	89.80	2,017	32.3	1.00
	LTPO	96.06	92.98	575	10.5	0.29
	Ours (v1)	96.86	93.53	1,007	18.2	0.51
MATH500	Zero-Shot CoT	83.00	65.88	3,097	49.5	1.00
	LTPO	88.80	79.57	1,824	30.5	0.59
	Ours (v1)	89.58	80.16	1,777	29.4	0.58

Table D: Measured accuracy and cost on Qwen3-8B under the light setting v1 ($S=5$, $r_m=8$, at most two mid-block insertions). Output tokens and TFLOPs are per generation, averaged over the 8 samples per problem. ×CoT is the token cost of Equation (9) relative to Zero-Shot CoT on the same benchmark.

(1,777 vs 1,824) and using less compute (29.4 vs 30.5 TFLOPs). Rollouts add internal computation, but mid-stream insertion shortens the explicit trace and offsets it, so the total does not increase.

Second, the light setting stays at or below the CoT cost on both accountings. Measured FLOPs are 18.2 against 32.3 on GSM8K and 29.4 against 49.5 on MATH500, while accuracy improves (B@8 94.84 → 96.86 and 83.00 → 89.58). Under Equation (9) the same settings cost 0.51× and 0.58× the CoT baseline. The rollout overhead is real but tunable, and at this setting it does not translate into a higher cost than explicit decoding.

F Initialization Ablation Details

This appendix reports the full per-dataset results for the four initialization strategies summarized in Section 4.4, for both the start latent block and the mid latent block.

Setup. We use Llama-3.2-3B-Instruct on 200 problems sampled from MATH500 and all 30 AIME2024 problems, the same split as the rollout-reward ablation (Section 4.3). Each configuration is run five times under independent sampling, and we report avg@5 accuracy (the mean over the five runs). For the start-block sweep the mid block is disabled; for the mid-block sweep the start block is held fixed at prev_tok. All other hyperparameters follow the global configuration of Appendix B.

Start block. Table E gives the start-block numbers. prev_tok attains the highest avg@5 on both datasets, but the margin is small: on MATH500 all four strategies lie within 42.3–44.3, a spread of two points, and placeholder, randn, and text are difficult to separate. The one strategy that behaves inconsistently is randn: it sits in the middle of the pack on MATH500 (44.1) but falls to the bottom on AIME2024 (2.0, against 8.0 for prev_tok). The

⁷<https://platform.claude.com/docs/en/about-claude/pricing>

Dataset	prev_tok	placeholder	randn	text
MATH500	44.3	43.5	<u>44.1</u>	42.3
AIME2024	8.0	3.3	2.0	<u>5.3</u>
Avg.	26.2	23.4	23.1	<u>23.8</u>

Table E: Start-block initialization, avg@5 accuracy (%). prev_tok is best on both datasets. The four strategies fall within about two points on MATH500 but differ by a factor of four on AIME2024, where randn is lowest.

spread is narrow on MATH500—all four strategies lie within two points, consistent with the cosine-similarity analysis of Appendix G, where the four start-block initializations stay close together across the network—but wide on AIME2024, where the best and worst differ by a factor of four. What distinguishes the start block from the mid block is therefore not the size of the gap but its ordering, which we take up below. We adopt prev_tok as the default start-block initialization: it is best on both datasets and the simplest to construct.

Mid block. Table F gives the mid-block numbers, with the start block held fixed at prev_tok. The spread is far wider than for the start block: placeholder attains the highest avg@5 on both datasets (44.0 on MATH500, 9.3 on AIME2024), randn ranks second, and prev_tok and text fall to 34.6 and 29.3 on MATH500 and to 2.0 and 3.3 on AIME2024. The ranking also inverts relative to the start block. prev_tok is the best start-block initialization on both datasets yet the weakest mid-block one on AIME2024, while placeholder moves from third at the start to first mid-stream. Anchoring the block to a concrete preceding token is therefore harmless before generation but costly mid-stream, and Sections 5.1–5.3 take up why.

G Layerwise Cosine Similarity

This appendix provides the full layerwise cosine-similarity analysis summarized in Observation 1 (Section 5). The main text reports only the MATH500 result; here we show all three datasets (AIME2024, GSM8K, MATH500) and both the start and the mid latent block.

Setup. We use Llama-3.2-3B-Instruct on all 30 AIME2024 problems and 200 problems each sampled from GSM8K and MATH500. At every transformer layer we read out the latent block’s hidden state with the Logit Lens (nostalgebraist, 2020) and compute, for each block position, the maximum co-

Dataset	prev_tok	placeholder	randn	text
MATH500	34.6	44.0	<u>36.8</u>	29.3
AIME2024	2.00	9.3	<u>4.7</u>	3.3
Avg.	18.3	26.7	<u>20.7</u>	16.3

Table F: Mid-block initialization, avg@5 accuracy (%), with the start block fixed to prev_tok. In contrast to the start block, the spread between strategies is large: placeholder is best, while the token-anchored prev_tok and text fall well below it—on AIME2024 to roughly a third.

sine similarity between that hidden state and any vocabulary embedding,

$$\cos_\ell = \max_{v \in V} \frac{\langle h_\ell, E(v) \rangle}{\|h_\ell\| \|E(v)\|}, \quad (10)$$

where h_ℓ is the hidden state at layer ℓ and $E(v)$ the embedding of vocabulary token v . A value close to 1 means the block has aligned with a single discrete token—the geometric signature of superposition collapse—while a lower value means the block remains spread over many directions in embedding space. We report the per-layer mean over block positions and over the problems in each dataset.

Start block. Figure 6 shows the start-block result. At the start position there is no preceding explicit reasoning to anchor the block to, and the previous-token embedding that prev_tok copies is the question’s final token rather than a committed reasoning step. As a result prev_tok does *not* collapse onto a single token: it maintains a comparatively low cosine similarity across the network, on par with or below placeholder. The four initializations therefore stay close together throughout the network, the geometric counterpart of the narrow $\sim 2\%$ start-block accuracy gap reported in Section 4.4. This is why we select prev_tok as the start-block initialization: it preserves superposition just as well as placeholder at the start, while being the simplest to construct.

Mid block. Figure 7 shows the mid-block result. The pattern is consistent across all three datasets and matches the top- k entropy analysis in the main text. The prev_tok initialization sits close to the nearest vocabulary embedding at most layers: anchoring the block to a concrete preceding reasoning token places it on a single embedding direction from the outset, and the forward pass keeps it there. The text initialization behaves

similarly, since it too starts from deterministic token embeddings. By contrast, placeholder and randn stay markedly further from any single token through the mid layers, approaching a token only in the final layers where last-layer entropy collapse is expected (Rizvi-Martel et al., 2026). The separation between the anchored (prev_tok, text) and the unanchored (placeholder, randn) initializations is widest on AIME2024, the hardest benchmark, and narrowest on GSM8K, where shallow reasoning leaves less room for the latent state to matter. This high cosine similarity is the geometric counterpart of the mid-block accuracy gap reported in Section 4.4: because mid-block prev_tok and text collapse onto a single token, they forfeit the superposition that the mid block is meant to carry, and their accuracy falls well below that of placeholder—on AIME2024 to roughly a third. The contrast between the two blocks is the central point: the *same* prev_tok initialization preserves superposition at the start but collapses it mid-stream, which is why initialization choice is far more consequential mid-stream than at the start.

Takeaway. Cosine similarity is a geometric proxy for superposition: an initialization that keeps the mid block away from any single embedding direction preserves a distribution over multiple tokens, whereas one that snaps it onto a token has already collapsed before any RL update. The full layerwise curves confirm that this distinction is set by initialization and is stable across datasets. Note, however, that staying away from a single token is necessary but not sufficient: both placeholder and randn preserve this distributional spread, yet placeholder is consistently the stronger of the two. The difference is not visible in the cosine geometry itself but in how the block behaves under RL updates, which we analyze in Observation 3 (Section 5.3) and revisit at the end of Appendix H.

H Layerwise Attention Magnitude

This appendix provides the full attention-magnitude curves referenced in Observation 2 (Section 5).

Setup. On the same data as Appendix G—all 30 AIME2024 problems and 200 problems each sampled from GSM8K and MATH500—we measure the total attention mass that later token positions direct back to the latent block. For a block occupying positions $\mathcal{B}$, the attention magnitude at layer ℓ

is

$$\text{AttnMag}_\ell = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} \sum_{b \in \mathcal{B}} \alpha_{q \rightarrow b}^{(\ell)}, \quad (11)$$

where $\mathcal{Q}$ is the set of query positions after the block and $\alpha_{q \rightarrow b}^{(\ell)}$ is the attention weight from query q to block position b , averaged over heads. We report the per-layer mean over the sampled problems.

Start block. Figure 8 shows the start-block result. The profile is consistent across all four initializations and all three datasets: attention magnitude peaks in the first few layers and then decays steadily, reaching near zero by mid-depth. The early peak is most pronounced for placeholder and randn and milder for prev_tok and text, but in every case later positions effectively stop attending to the start block once the early layers have read it. This monotone decay—together with the narrow $\sim 2\%$ start-block accuracy gap reported in Section 4.4—indicates that at the start position the choice of initialization has little leverage: the block is consumed early regardless of how it is initialized.

Mid block. Figure 9 shows the mid-block result, and the contrast with the start block is the central point. The mid-block magnitude is *not* monotone. Under placeholder and randn the curve falls after the early layers but then rebounds through the mid layers (L8–L16), forming a pronounced U-shape that is clearest on AIME2024 and still visible on GSM8K and MATH500. Later positions therefore continue to read the mid block precisely at the depth where reasoning is organized. Under prev_tok and text the mid-layer region is comparatively flat; the magnitude instead drifts upward toward the final layers, where it is consumed near read-out rather than during the formation of the reasoning. The mid-layer rebound co-occurs with higher accuracy: the two initializations that preserve superposition (placeholder, randn, Appendix G) are also the two whose mid block is actively attended while reasoning is still being shaped. The two collapsed initializations forfeit both—they hold a single token and are read only late.

Takeaway and connection to RL. Attention magnitude alone does not separate the two superposition-preserving initializations: placeholder and randn have similar mid-layer rebounds, yet placeholder is the better mid-block initialization (Section 4.4). As with the cosine geometry of Appendix G, the magnitude profile is necessary but not sufficient to identify a good

initialization; what distinguishes placeholder from randn is their behavior under RL updates, analyzed in Section 5.3.

I Case Study

In this section, we examine the effect of mid-block insertion through an AIME2024 trace from Qwen3-14B. Starting from similar reasoning steps, Zero-Shot CoT fails to reach a correct answer; its failures broadly take two forms: reasoning-path misdirection (Figure 11) and verification loops (Figure 12). MidThink, in contrast, commits to the derived relation and closes on the correct answer (Figure 10).

J Elaboration

Artifact use. We use the publicly available implementations of LTPO⁸, released under the MIT License, and of SoftCoT⁹, released under the NTU-ITIV Dual License Agreement, which permits non-commercial academic research use. Our use of both is limited to non-commercial research and is consistent with their stated terms; we do not redistribute either codebase or derivatives thereof. The five mathematical reasoning benchmarks (GSM8K, MATH500, AIME2024, AIME2025, ASDiv-Aug) and the pretrained models (Llama-3.2, Ministral-3, Qwen3) are publicly released artifacts, used here solely for research evaluation consistent with their intended use. We do not deploy any of these artifacts in real-world applications.

AI assistant use. We used an AI assistant for language editing of the manuscript. All experiments, implementations, analyses, and numerical results are our own, and no text was included without author verification.

K Experimental details

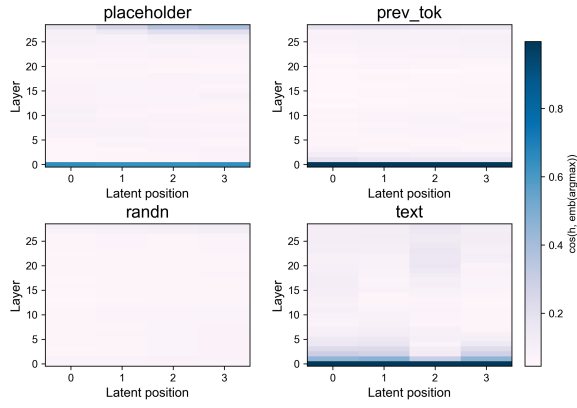
Data statistics. We evaluate on English-language mathematical reasoning benchmarks: GSM8K, MATH500, AIME2024, AIME2025, and ASDivAug. We use 1,319 GSM8K, 500 MATH500, 30 AIME2024, 30 AIME2025, and 1,038 ASDivAug problems.

All five benchmarks consist of mathematical word problems and competition questions; they contain no personally identifying information or offensive content by construction.

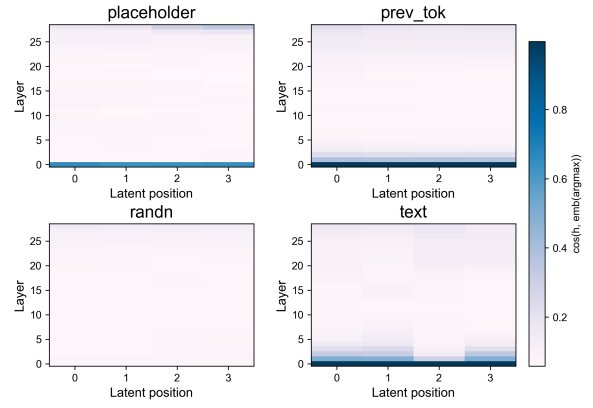
Compute. Experiments were run on NVIDIA RTX 3090 and NVIDIA A100 GPUs. Model sizes are 3.2B (Llama-3.2-3B-Instruct), 8B (Ministral-3-8B-Reasoning-2512), 8B (Qwen3-8B), and 14B (Qwen3-14B) parameters.

⁸<https://github.com/ltpo2025/LTPO>

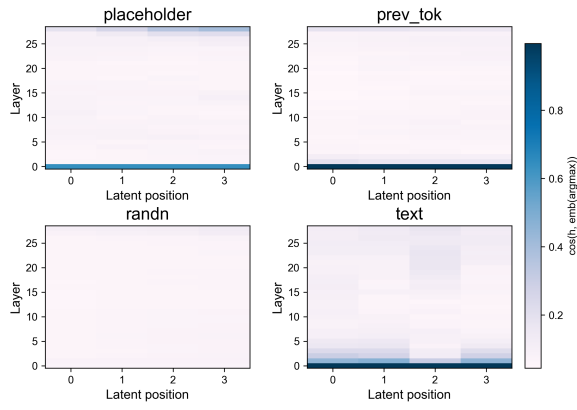
⁹<https://github.com/xuyige/SoftCoT>



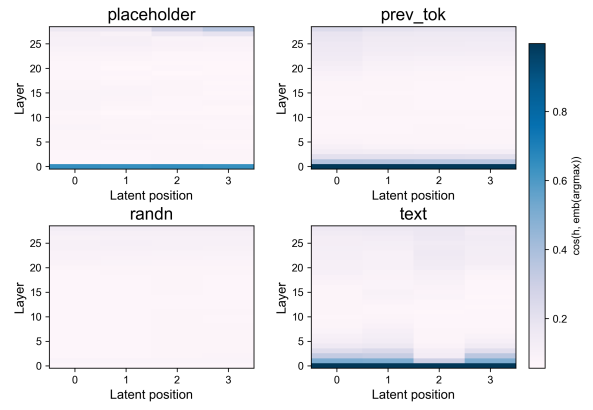
(a) AIME2024



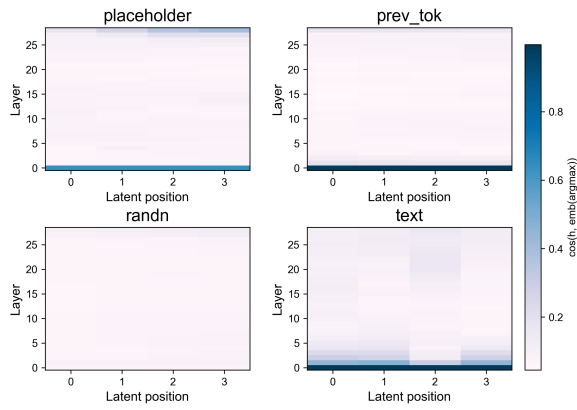
(a) AIME2024



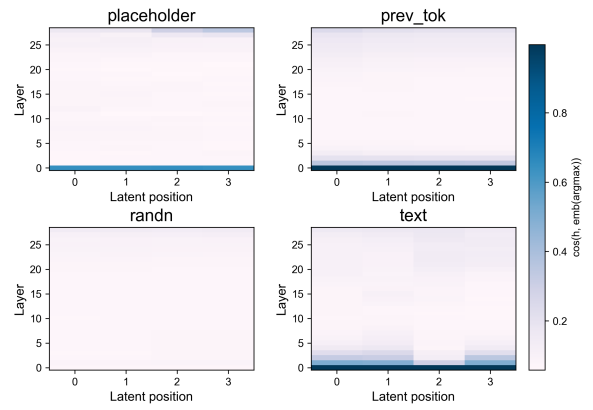
(b) GSM8K



(b) GSM8K



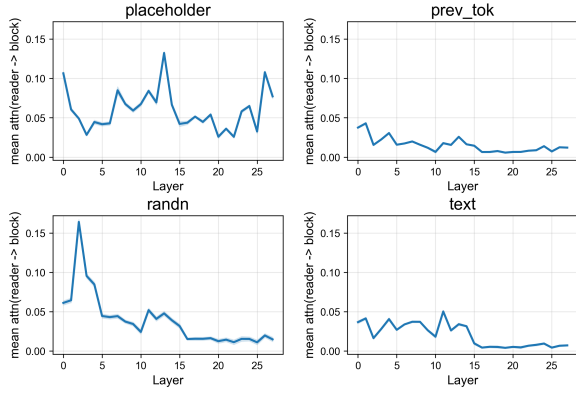
(c) MATH500



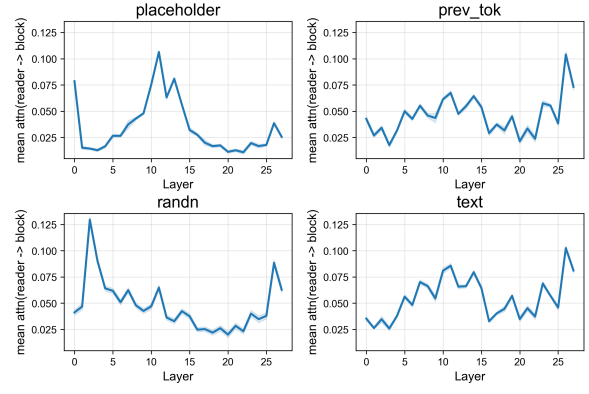
(c) MATH500

Figure 6: Layerwise maximum cosine similarity for the *start* latent block, under the four initialization strategies, across three datasets. Higher values indicate collapse onto a single token.

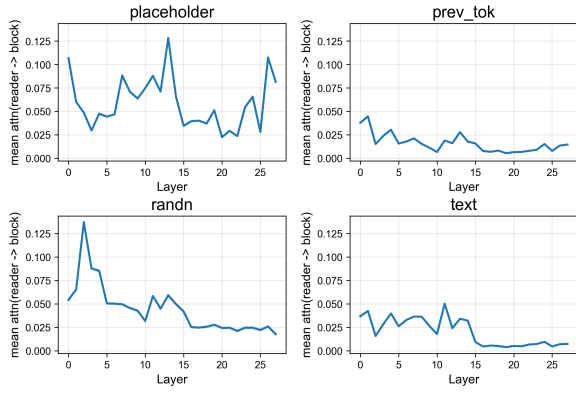
Figure 7: Layerwise maximum cosine similarity for the *mid* latent block, under the four initialization strategies, across three datasets. Higher values indicate collapse onto a single token.



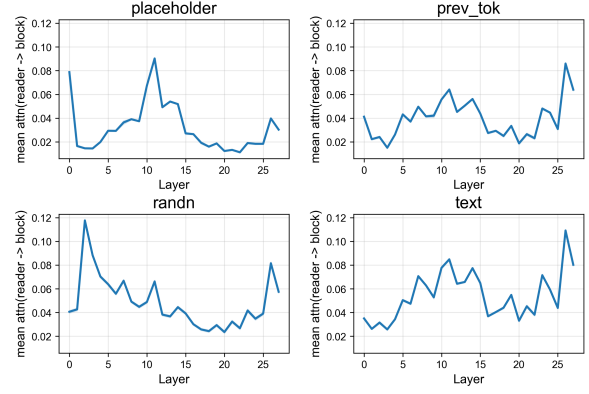
(a) AIME2024



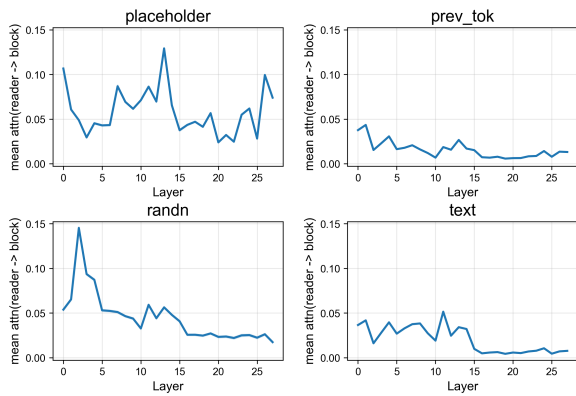
(a) AIME2024



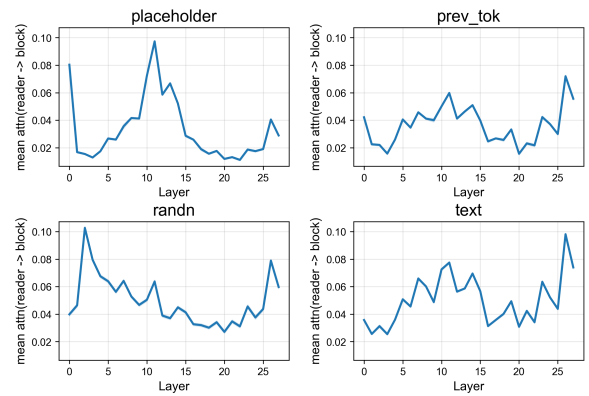
(b) GSM8K



(b) GSM8K



(c) MATH500



(c) MATH500

Figure 8: Layerwise attention magnitude directed at the *start* latent block, under the four initialization strategies, across three datasets.

Figure 9: Layerwise attention magnitude directed at the *mid* latent block, under the four initialization strategies, across three datasets.

Problem. There exist real numbers x and y , both greater than 1, such that $\log_x(y^x) = \log_y(x^{4y}) = 10$. Find xy .
Gold: 25

MidThink — correct

Okay, let's try to solve this problem. So, we have two equations here involving logarithms with bases x and y , and both x and y are greater than 1. The equations are $\log_x(y^x) = 10$, and $\log_y(x^{4y}) = 10$. We need to find the product xy . First, I need to recall the logarithm properties. The logarithm of a number raised to an exponent can be simplified by bringing the exponent in front. So, $\log_b(a^c) = c \cdot \log_b(a)$. Maybe I can apply that here. Starting

[MIDBLOCK#1]

Step 1: Apply logarithm power rule

$$\log_x(y^x) = x \log_x(y) = 10 \implies \log_x(y) = \frac{10}{x}$$

$$\log_y(x^{4y}) = 4y \log_y(x) = 10 \implies \log_y(x) = \frac{10}{4y} = \frac{5}{2y}$$

Step 2: Use change of base formula Recall that $\log_b(a) = \frac{1}{\log_a(b)}$. So [MIDBLOCK#2]

$$\log_x(y) = \frac{1}{\log_y(x)} \implies \frac{10}{x} = \frac{1}{\frac{5}{2y}} = \frac{2y}{5}$$

Step 3: Solve for relationship between x and y

$$\frac{10}{x} = \frac{2y}{5} \implies 50 = 2xy \implies xy = 25$$

Thus, the product xy is 25.

Figure 10: **MidThink (correct)**. After the latent pause at [MIDBLOCK#2], the trace commits to the derived relation and closes on the answer, rather than reopening the derivation. The pause softens the post-derivation uncertainty that would otherwise drive further deliberation in explicit decoding, reducing reasoning-path misdirection, unnecessary verification loops, and similar responses (Figs. 11, 12).

Zero-Shot CoT — wrong case 1

2.5Okay, so I need to find real numbers x and y , both greater than 1, such that the logarithms log base x of y^x and log base y of x^{4y} are both equal to 10. Then I need to find the product xy . Alright, let's start by writing down the equations given. First, the problem says that $\log_x(y^x) = 10$ and $\log_y(x^{4y}) = 10$. Let me write these equations out using logarithm properties. Starting with the first equation: log base x of y^x equals 10. Remember that $\log_b(a^c) = c * \log_b(a)$. So applying that here, $\log_x(y^x)$ is equal to $x * \log_x(y)$. And that equals 10. So: $x * \log_x(y) = 10$. Similarly, for the second equation: log base y of x^{4y} equals 10. Applying the same logarithm property, that would be $4y * \log_y(x) = 10$. So now I have two equations: 1) $x * \log_x(y) = 10$ 2) $4y * \log_y(x) = 10$. Hmm. Now, I need to relate these two equations. Let me recall that $\log_b(a)$ is equal to $1 / \log_a(b)$. So $\log_x(y)$ is $1 / \log_y(x)$. Let me denote $\log_y(x)$ as some variable to make it easier. Let me call $\log_y(x) = k$. Then $\log_x(y) = 1/k$. So substituting into equation 1: $x * (1/k) = 10$, which gives $x/k = 10$. Equation 2: $4y * k = 10$. So $4yk = 10$. So now I have: $x/k = 10 \Rightarrow x = 10k$ and $4yk = 10 \Rightarrow yk = 10/4 = 5/2 \Rightarrow y = (5/2)/k = 5/(2k)$. So now I have expressions for x and y in terms of k . Let me write them again: $x = 10k$ $y = 5/(2k)$. Now, since $\log_y(x) = k$, let's write that equation. log base y of x is k . Which means that $y^k = x$. So substituting the expressions for x and y : $y^k = x \Rightarrow [5/(2k)]^k = 10k$. So now I have an equation in terms of k : $[5/(2k)]^k = 10k$. This seems a bit complicated. [...]

Figure 11: **Failure mode 1: Reasoning-path misdirection.** Starting from the similar algebra as MidThink, the trace does not commit at the reciprocal identity; the uncertainty after the step is acted on by introducing an additional variable and pursuing a harder intermediate equation, expanding the search rather than closing it. The detour pulls the reasoning away from the relation it had in hand, and the trace ends without ever reaching an answer.

Zero-Shot CoT — wrong case 2

2.5Okay, so I need to solve this problem where there are real numbers x and y , both greater than 1, such that log base x of y^x equals log base y of x^{4y} equals 10. And I need to find the product xy . Alright, let's start by writing down what is given. First, the problem says: $\log_x(y^x) = 10$ and $\log_y(x^{4y}) = 10$. So, there are two equations here. ... Let me recall logarithm properties. Remember that $\log_b(a^c) = c * \log_b(a)$. Starting with the first equation: $\log_x(y^x) = 10$. Applying the logarithm power rule: this becomes $x * \log_x(y) = 10$ since $x * \log_x(y) = 10$, then $\log_x(y) = 10 / x$. Similarly, let's work on the second equation: $\log_y(x^{4y}) = 10$. Again, applying the power rule: $4y * \log_y(x) = 10$. Therefore, $\log_y(x) = 10 / (4y) = 5 / (2y)$. Now, note that $\log_x(y)$ and $\log_y(x)$ are reciprocals of each other. Because $\log_b(a) = 1 / \log_a(b)$. So, if I have $\log_x(y) = 10/x$, and $\log_y(x) = 5/(2y)$, then according to the reciprocal relationship, $\log_x(y)$ should be $1 / \log_y(x)$. Therefore: $10/x = 1 / [5/(2y)] = 2y / 5$. So, setting them equal: $10/x = (2y)/5$. Cross-multiplying: $10 * 5 = 2y * x$ which is $50 = 2xy \Rightarrow xy = 50 / 2 = 25$. Wait, so is that it? If that's the case, then $xy = 25$? But wait, let me check if that's consistent with the equations. Because maybe I need to check if there are solutions for x and y that satisfy both equations and also that x and y are greater than 1. [...]

Figure 12: **Failure mode 2: verification loops.** The trace reaches the same intermediate result MidThink uses, then post-derivation uncertainty opens a verification branch ("Wait, so is that it?"). Subsequent re-substitutions and trial searches keep the reasoning open without re-emitting the derived value, and the trace closes without ever returning to the answer.