WHERE TO PAUSE AND HOW TO RESUME: TOWARDS HUMAN-INSPIRED ITERATIVE CHAIN-OF-THOUGHT

Anonymous authors
Paper under double-blind review

ABSTRACT

Iterative chain-of-thought (CoT) reasoning extends reasoning depth by periodically summarizing intermediate steps and resuming generation within a bounded context. However, existing iterative CoT approaches typically construct supervision through length-based segmentation, which can interrupt reasoning at arbitrary positions and provide limited guidance on how to resume from summaries. We propose PhaseCoT, a data-centric refinement that improves iterative reasoning along two dimensions: *where to pause* and *how to resume*. PhaseCoT segments reasoning trajectories at semantic milestones, so that each segment corresponds to a coherent reasoning phase, and further rewrites segment heads to introduce context-aware continuation from prior summaries. We construct supervised fine-tuning datasets for standard CoT, conventional length-driven iterative CoT, and PhaseCoT, and evaluate multiple models on MATH500, AIME22, and AMC23. Results show that PhaseCoT improves mathematical reasoning over both standard CoT and length-driven iterative CoT in most model-benchmark comparisons, which suggests that effective iterative reasoning requires not only longer reasoning horizons, but also semantically grounded pauses and context-aware resumptions.

1 INTRODUCTION

Recent advances in large reasoning models have shown that scaling chain-of-thought (CoT) at inference time can substantially improve performance on tasks requiring multi-step deliberation (Nye et al., 2021; Wei et al., 2022; Kojima et al., 2022; Brown et al., 2024; Snell et al., 2025; Muennighoff et al., 2025). Models such as OpenAI o1, DeepSeek-R1, and QwQ often generate long reasoning trajectories that explore alternatives, backtrack, and refine intermediate conclusions before producing a final answer (OpenAI et al., 2024; Guo et al., 2025; Qwen Team, 2025). However, this monolithic long-CoT paradigm has several limitations. Attention cost grows quadratically with sequence length in Transformer architectures (Vaswani et al., 2017); reasoning may terminate once the context window is exhausted; and even within the window, long-range dependencies can be weakened by "lost-in-the-middle" effects (Liu et al., 2024).

A growing line of work addresses these issues through iterative reasoning, where reasoning is extended by repeatedly compressing accumulated context and resuming from summaries (Yan et al., 2026b; Vajipey et al., 2025; Wu et al., 2025; Yan et al., 2026a). In this paradigm, generation is periodically interrupted, the accumulated reasoning is compressed into a summary, and the model then resumes reasoning within a refreshed bounded context conditioned on that summary. A representative example is *InftyThink* (Yan et al., 2026b), which operationalizes this paradigm by training the model to internalize three key capabilities, as illustrated in Figure 1. This makes it possible to extend reasoning depth while keeping the active context size bounded.

Despite the promise of this iterative paradigm, it remains largely driven by a simple length-based dataset construction strategy (Yan et al., 2026b). This creates two important discrepancies between the learned iterative behavior and how humans naturally organize extended reasoning, as studied in human and mathematical problem solving (Newell & Simon, 1972; Pólya, 1945; Schoenfeld, 1985). First, the model is forced to pause according to a predefined length threshold rather than a semantic milestone, while human reasoning tends to pause after finishing a subgoal, resolving a case or deriving an intermediate result. Second, the model is not explicitly trained to re-establish context

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.

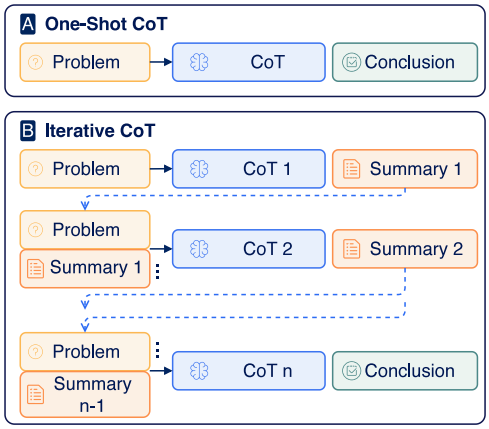


Figure 1: Vanilla versus iterative reasoning. Iterative reasoning extends reasoning depth within a bounded active context by training the model to (i) pause at a stopping point (a predefined length threshold in length-driven methods such as InftyThink), (ii) summarize the key intermediate steps and conclusions obtained so far, and (iii) resume conditioned on the original problem and the summary.

when resuming reasoning, but humans often begin the next reasoning phase with a brief recap that situates the problem and clarifies what has already been obtained.

These discrepancies motivate the central question of this paper: **Can iterative CoT be made more effective by aligning its structure with phase-based descriptions of human and mathematical problem solving?** To this end, we propose **PhaseCoT**, a data-centric refinement of conventional length-driven iterative CoT that improves the construction of iterative reasoning supervision along two dimensions: *where to pause* and *how to resume*.

Semantic milestone-based segmentation: Instead of imposing boundaries based on a target length, we partition the original CoT according to semantic milestones. By aligning segment boundaries with these milestones, each segment becomes a complete and meaningful reasoning phase, which enables higher-quality summaries and provides more useful context for later iterations.

Context-aware continuation via segment rewriting: Semantic segmentation alone does not guarantee smooth transitions between iterations. We therefore further refine the training data by rewriting the beginning of each segment so that it naturally follows from the previous summary. This introduces a concise contextual re-grounding step before new reasoning begins.

To evaluate PhaseCoT, we construct three types of datasets: standard CoT supervision, length-driven iterative CoT supervision, and our proposed human-inspired iterative CoT supervision that involves semantic segmentation and segment rewriting. We fine-tune multiple models on these datasets and evaluate them on MATH500 (Hendrycks et al., 2021; Lightman et al., 2024), AIME22 (Mathematical Association of America, 2022), and AMC23 (Mathematical Association of America, 2023). In most model-benchmark comparisons, PhaseCoT improves mathematical reasoning performance, showing that the effectiveness of iterative reasoning depends significantly on whether it pauses at meaningful reasoning states and resumes from the compressed summary in a contextually grounded manner.

2 Related Work

Long reasoning and inference-time scaling Chain-of-thought (CoT) prompting has become a standard way to elicit multi-step reasoning in large language models. Recent reasoning models and test-time scaling studies further show that allocating more inference-time computation can improve mathematical, coding, and scientific reasoning performance (OpenAI et al., 2024; Brown et al., 2024; Snell et al., 2025; Muennighoff et al., 2025; Guo et al., 2025; Kimi Team et al., 2025; Yang et al., 2025). However, long monolithic reasoning traces are expensive: Transformer attention and

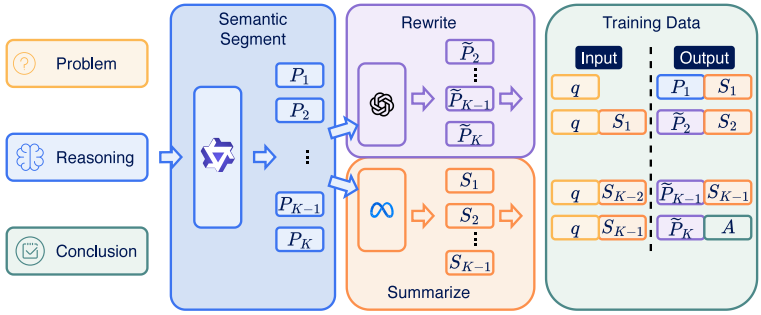


Figure 2: Overview of the PhaseCoT data construction pipeline. Given a problem and its long-form CoT, PhaseCoT (i) identifies semantic milestones and partitions the CoT into coherent phases $P_1, \ldots, P_K$; (ii) generates cumulative summaries S_i of $P_{\leq i}$ for $i < K$; and (iii) rewrites the head of each P_j ($j \geq 2$) into $\widetilde{P}_j$ so that it continues naturally from S_{j-1} while the rest of the segment is preserved verbatim. Problem, summary, rewritten segment, and conclusion are then assembled into the training instances of Equation (3).

KV-cache costs grow with sequence length (Vaswani et al., 2017), and long contexts may still suffer from degradation such as the "lost-in-the-middle" effect (Liu et al., 2024). This motivates methods that either compress reasoning traces or restructure reasoning into a sequence of shorter iterative phases.

Compressing chain-of-thought One line of work reduces the token footprint of CoT while preserving accuracy. Prompting methods encourage concise drafts or adapt token budgets to instance difficulty (Renze & Guven, 2024; Xu et al., 2025a; Han et al., 2025); training-based approaches learn length-controllable or concise reasoning behaviors (Ma et al., 2025; Luo et al., 2025; Munkhbat et al., 2025; Qiao et al., 2025); and pruning methods remove redundant tokens, steps, or chunks from long traces (Xia et al., 2025; Cui et al., 2025; Choi et al., 2025; Wang et al., 2025). Another family compresses reasoning into dense, continuous, or soft representations rather than natural-language text (Cheng & Van Durme, 2024; Hao et al., 2025; Shen et al., 2025; Xu et al., 2025b; Zhang et al., 2025). These methods demonstrate substantial redundancy in long CoT, but they largely optimize a single monolithic trace. In contrast, PhaseCoT keeps reasoning in natural language and focuses on how to construct supervision for summary-conditioned continuation across multiple phases, making it compatible with standard supervised fine-tuning pipelines.

Structured and iterative reasoning A complementary line of work focuses on modifying the reasoning procedure itself. Decomposition and planning methods break complex problems into smaller subproblems or explicit plans (Press et al., 2023; Zhou et al., 2023; Khot et al., 2023; Wang et al., 2023), while search-based approaches explore multiple reasoning paths, such as trees, graphs, or algorithmic structures (Yao et al., 2023a; Besta et al., 2024; Sel et al., 2024). Iterative and agent-based methods further improve outputs through feedback, actions, or self-reflection (Yao et al., 2023b; Madaan et al., 2023; Shinn et al., 2023). While these approaches demonstrate the benefits of structured reasoning, they remain limited from the perspective of *internalization*: they are either implemented as inference-time frameworks without being learned by the model itself, or, when internalized through training, fail to capture semantically grounded segmentation and context-aware continuation. Consequently, they do not fully support extending a single coherent reasoning trajectory within a bounded context.

3 METHODS

Given a problem q, its original long-form chain-of-thought (CoT) reasoning process R, and the corresponding final answer A, PhaseCoT reconstructs iterative CoT training data by reorganizing the reasoning trajectory into a sequence of *semantic phases* (Figure 2). It then ensures that each subsequent phase begins with an explicit contextual bridge from the accumulated summary of prior reasoning.

3.1 SEMANTIC MILESTONE-BASED SEGMENTATION

A *semantic milestone* is a natural transition point in the reasoning process where a coherent subtask has been completed and the current reasoning state is ready to be summarized. We define a boundary position t in the reasoning process R as a valid semantic milestone if two conditions hold. First, the prefix $R_{1:t}$ should form a self-contained reasoning phase whose main conclusion can be summarized without relying on unfinished derivations. Second, the suffix $R_{t+1:}$ should begin a new reasoning operation rather than continue a syntactically or logically incomplete step.

In practice, we implement milestone identification using an LLM-based segmenter $\mathcal{G}_{\text{seg}}$ (Qwen3.5-397B-A17B). The segmenter receives the full original reasoning process R as input (the problem statement is not provided) and outputs a sequence of boundary strings $(b_1, b_2, \ldots, b_{K-1}) = \mathcal{G}_{\text{seg}}(R)$, each of which is required to be copied verbatim from the original CoT and to be the final sentence of a semantic phase (the segmenter also returns the final sentence of R as the last anchor, which defines no boundary). We use these boundary strings to partition the reasoning process into a sequence of semantically coherent segments $R = P_1 \oplus P_2 \oplus \cdots \oplus P_K$, where each segment P_i is the contiguous span of text between two consecutive boundary strings and, by construction, corresponds to a complete reasoning phase. See Appendix A for the prompt used for milestone identification and model output example. Appendix H.1 evaluates the boundaries.

3.2 SEGMENT SUMMARIZATION

After obtaining semantic segments, we generate a compact progress summary after each non-final segment. We use Llama3.3-70B-Instruct as the summarization model. The summary is intended to represent the accumulated reasoning state up to the current phase, rather than merely a local description of the current segment. Formally, for each $1 \leq i < K$, we generate

$$S_i = \mathcal{G}_{\text{sum}}\left(q,\, P_{\leq i}\right), \qquad P_{\leq i} = P_1 \oplus P_2 \oplus \cdots \oplus P_i. \tag{1}$$

The resulting summary S_i stores the key intermediate conclusions, definitions, constraints, and unresolved goals needed to continue the solution. See Appendix B for the summarization prompt and model output example. Appendix H.2 audits summary fidelity.

3.3 CONTEXT-AWARE CONTINUATION VIA SEGMENT REWRITING

Semantic segmentation ensures that the model pauses at meaningful points, but it does not by itself guarantee that the next segment will read naturally when conditioned only on the previous summary. The original segment P_i for $i > 1$ was written as a continuation of the full prior CoT $P_{<i}$, not as a continuation of the compressed summary S_{i-1}. Therefore, even with semantic boundaries, the beginning of P_i may still rely on implicit context that is no longer visible.

To address this issue, PhaseCoT rewrites only the beginning of each non-initial segment. For each $i > 1$, we split the original segment P_i at its first paragraph delimiter into a rewrite head $H_i = \mathrm{FirstPara}(P_i)$ and an unchanged tail T_i, use GPT-OSS-120B as the rewriter $\mathcal{G}_{\text{rew}}$ to generate a new head conditioned on the problem, the previous summary, and the original head, and concatenate the rewritten head with the original tail:

$$P_i = H_i \oplus T_i, \quad \widetilde{H}_i = \mathcal{G}_{\text{rew}}(q, S_{i-1}, H_i), \quad \widetilde{P}_i = \widetilde{H}_i \oplus T_i, \qquad i > 1. \tag{2}$$

This design restricts rewriting to the original opening paragraph of the segment; it does not regenerate the entire segment. The rewriter is explicitly instructed to preserve the mathematical intent of H_i, avoid introducing new claims or conclusions not supported by the provided text, and produce a continuation that is stylistically consistent with human-written reasoning. See Appendix C for the rewriting prompt and model output example. Appendix H.3 audits rewrite fidelity.

3.4 TRAINING INSTANCE CONSTRUCTION

Finally, we combine the semantic segments, cumulative summaries, and rewritten continuations into PhaseCoT training instances. Following InftyThink, a reasoning process shorter than the 4K-token segmentation threshold is kept as a single phase: the segmenter is not invoked and PhaseCoT introduces neither summaries nor rewriting, i.e., $\mathcal{D}^{\text{PhaseCoT}}(q, R, A) = \{(q, R, A)\}$ for $K = 1$; the

Table 1: Main results, obtained by sampling each model 16 times at temperature 0.7. ACC: average accuracy (%); TOK: average number of generated tokens (K); Average is over the three benchmarks. Standard deviations across the 16 samples (one training run per configuration) are below 0.9% for ACC and 0.15K for TOK in all cases.

Model	**Train Format**	**MATH500**		**AIME22**		**AMC23**		**Average**	
		ACC	TOK	ACC	TOK	ACC	TOK	ACC	TOK
Llama3.1-8B	Vanilla	50.80	1.74	18.54	6.05	54.53	4.52	41.29	4.10
	InftyThink	51.10	1.91	19.17	6.63	55.00	5.03	41.76	4.52
	PhaseCoT	55.18	2.04	20.63	7.08	60.16	4.96	45.32	4.69
Qwen3-8B-Base	Vanilla	67.00	2.23	24.38	8.68	67.03	6.27	52.80	5.73
	InftyThink	64.00	2.13	22.29	7.00	62.97	4.42	49.75	4.52
	PhaseCoT	68.80	2.17	23.33	11.71	67.66	5.91	53.26	6.60
Qwen3-14B-Base	Vanilla	74.20	3.01	32.50	10.26	73.91	7.44	60.20	6.90
	InftyThink	77.70	3.34	33.13	11.02	73.59	7.88	61.47	7.41
	PhaseCoT	80.40	3.59	39.58	12.37	80.00	8.31	66.66	8.09

same holds when the segmenter returns a single phase for a longer process. When $K > 1$, PhaseCoT produces one instance for each semantic phase, $\mathcal{D}^{\mathrm{PhaseCoT}}(q, R, A) = \{D_k^{\mathrm{phase}}\}_{k=1}^{K}$, where

$$D_i^{\mathrm{phase}} = \begin{cases} (q, P_1, S_1), & i = 1, \\ (q, S_{i-1}, \widetilde{P}_i, S_i), & 1 < i < K, \\ (q, S_{K-1}, \widetilde{P}_K, A), & i = K. \end{cases} \tag{3}$$

4 Experiments

4.1 Settings

Training data All training variants are constructed from the same source pool. We randomly sample 11,776 CoT examples (a multiple of the global batch size of 256) from OpenR1-Math-220K as the seed set $\mathcal{S}$ to build different CoT or iterative CoT training data.

Evaluation datasets We evaluate on MATH500, AIME22, and AMC23, which cover both standard mathematical problem solving and competition-style reasoning across different difficulty levels.

Backbones We evaluate both Base and Instruct models from the Llama and Qwen families. The Base models include Llama3.1-8B, Qwen3-8B-Base, and Qwen3-14B-Base. The Instruct models include Llama3.1-8B-Instruct, Qwen3-8B, and Qwen3-14B.

Compared methods We evaluate three training formats. (1) *Vanilla*: the original CoT dataset $\mathcal{S}$ without any iterative reconstruction. (2) *InftyThink*: CoT instances in $\mathcal{S}$ are segmented using the conventional length-driven strategy proposed by InftyThink (Yan et al., 2026b), with a target segment length of 4K tokens, the default setting used in their main experiments. (3) *PhaseCoT*: CoT instances in $\mathcal{S}$ are segmented using the semantic milestone-based strategy and augmented with context-aware segment-head rewriting, as introduced in Section 3. See Appendix D for additional details on the experimental setup, training hyperparameters, and evaluation protocols.

4.2 Main results

Table 1 reports the main results across three base backbones and three mathematical reasoning benchmarks; results for their instruct-model counterparts are provided in Appendix G.

First, PhaseCoT achieves the best average accuracy for every backbone and improves over the baselines in most model-benchmark comparisons. At the model-benchmark level, PhaseCoT outperforms Vanilla in 8 out of 9 comparisons and InftyThink in 9 out of 9 comparisons. The improvement

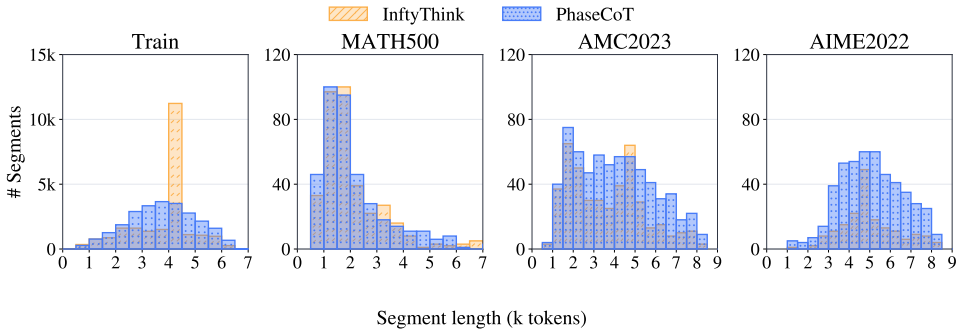


Figure 3: Segment length distributions of InftyThink and PhaseCoT. The four panels show the segment length distributions on the training set and on model generations from MATH500, AMC23, and AIME22.

reaches up to $+7.08$ points over Vanilla and $+6.45$ points over InftyThink, indicating that the gains are substantial and not tied to a particular model family or benchmark.

Second, PhaseCoT usually generates more tokens than the two baselines, particularly on more challenging benchmarks. Part of this increase is the structural overhead of iterative reasoning: each additional round adds a progress summary, and the next round opens with a context-aware transition that resumes from it. Another part is that PhaseCoT leads the model to reason through more rounds, especially on harder problems (Section 5.2). Whether the accuracy gains could instead be obtained by simply spending more tokens is addressed by the budget-matched comparisons in Section 6.2.

5 Analysis

To understand why PhaseCoT is effective, we compare the generation behavior of InftyThink, a representative length-driven method, and PhaseCoT on Qwen3-14B-Base across MATH500, AMC23, and AIME22 (in increasing order of difficulty), using the generations in which every round terminates normally, focusing on two aspects: the segment length distribution and the distribution of iterative generation rounds. Ablations are deferred to Section 6.

5.1 Segment length distribution

Figure 3 compares the segment length distributions of InftyThink and PhaseCoT. On the training set, InftyThink shows a sharp peak around its target length. This is because its segmentation is mainly controlled by a predefined token budget, which makes segment lengths regular but may split a reasoning phase in the middle.

PhaseCoT instead produces a broader and more balanced distribution: since it segments reasoning by semantic milestones, the length of each segment depends on the complexity of the current reasoning phase, so simple steps form short segments while complete derivations or harder subproblems require longer ones.

The same trend appears in test-time generations: across MATH500, AMC23, and AIME22, PhaseCoT consistently yields a more balanced segment length distribution than InftyThink, and the difference grows as the benchmark becomes harder. On MATH500, many problems can be solved with compact reasoning, so the two distributions are closer, whereas on AMC23 and especially AIME22, PhaseCoT's semantic pause policy leads to more variable segment lengths while InftyThink remains strongly influenced by length-driven stopping. This indicates that PhaseCoT changes the learned iterative behavior: the model does not simply stop after a roughly fixed number of tokens, but pauses after completing a coherent reasoning phase and then resumes from a summarized state. Appendix H.1 tests this directly on the pauses of the trained models.

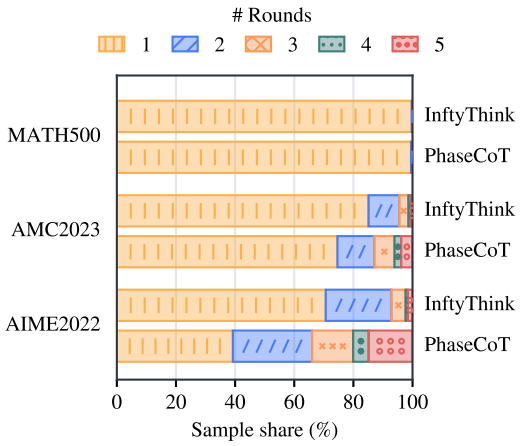


Figure 4: Distribution of iterative generation rounds. We compare the proportions of samples completed with different numbers of generation rounds by InftyThink and PhaseCoT on MATH500, AMC23, and AIME22 using Qwen3-14B-Base.

5.2 Distribution of iterative generation rounds

Figure 4 shows the distribution of the number of iterative generation rounds. For both methods, harder datasets lead to a larger proportion of high-round generations, consistent with the intuition that more difficult problems require more intermediate reasoning phases.

However, PhaseCoT produces a noticeably larger proportion of high-round samples than InftyThink, especially on the harder AMC23 and AIME22, i.e., it is more willing to continue reasoning across multiple phases. Since PhaseCoT is trained to pause at semantic milestones, each round tends to correspond to a meaningful reasoning unit, so a larger number of rounds reflects a finer-grained decomposition of the trajectory into coherent phases rather than fragmented generation.

Using more rounds is only beneficial if the model reliably reconnects each new round with the previous summary; the round distribution alone cannot show this, since it does not separate the effect of semantic segmentation from that of context-aware continuation. Section 6.3 isolates the two components and shows that both contribute to the accuracy gains.

6 Ablation Study

The comparisons in Table 1 hold the source pool fixed, but not the amount of optimization each training format receives or the test-time computation each trained model spends. Before attributing the gains to PhaseCoT itself, we therefore control the training supervision budget (Section 6.1) and the inference budget (Section 6.2), and then isolate the contribution of each component (Section 6.3).

6.1 Training supervision budget

Where the budgets differ All three formats are built from the same 11,776 trajectories, and every reasoning token of every trajectory appears exactly once as a supervision target in each format. The formats nevertheless differ in optimization budget: Equation (3) expands a K-phase trajectory into K shorter instances, whose targets additionally include the summaries S_i (and, for PhaseCoT, the rewritten heads $\widetilde{H}_i$) and whose prompts repeat the problem and carry S_{i-1}; and since the global batch is a fixed number of instances, the number of optimizer updates scales with the instance count rather than the token count. Relative to Vanilla, PhaseCoT has $1.94\times$ the instances and $1.96\times$ the optimizer updates but only $1.07\times$ the loss tokens and $1.14\times$ the total processed tokens, and InftyThink $1.78\times$ (instances and updates alike), $1.04\times$, and $1.09\times$, respectively (Table 4 in Appendix D). The main experiments match the three formats by epochs (three passes over their respective instances), not by updates or tokens; the extra loss tokens of the iterative formats consist entirely of summaries and rewritten heads, while the number of updates is the quantity that differs most.

Table 2: Accuracy (%) of the baselines trained with a dominated budget versus PhaseCoT trained for three epochs, on Qwen3-14B-Base. Updates and loss tokens are given relative to PhaseCoT (3 epochs), following Table 4.

Train Format	Updates	Loss tok.	MATH500	AIME22	AMC23	Average
Vanilla (3 ep.)	0.51	0.93	74.20	32.50	73.91	60.20
Vanilla (4.5 ep.)	0.77	1.40	73.33	32.08	72.97	59.46
Vanilla (6 ep.)	1.02	1.87	74.46	32.50	74.38	60.45
InftyThink (3 ep.)	0.91	0.97	77.70	33.13	73.59	61.47
InftyThink (4 ep.)	1.21	1.29	76.33	34.79	74.22	61.78
PhaseCoT (3 ep.)	1.00	1.00	80.40	39.58	80.00	66.66

Budget-dominated baselines Rather than matching the budgets exactly along a single axis, we train each baseline until it receives strictly more of everything: Vanilla for up to $2\times$ the number of epochs (4.5 and 6 epochs) and InftyThink for 4 epochs, with the same learning-rate schedule stretched over the longer run and all other hyperparameters unchanged. Vanilla at six epochs and InftyThink at four epochs both receive more optimizer updates, more loss tokens, and more training compute than PhaseCoT at three epochs, so any remaining gap can no longer be attributed to extra training. Table 2 reports the results. Doubling the Vanilla budget moves its average from 60.20 to 60.45, and the intermediate 4.5-epoch run is slightly worse (59.46), so additional passes over the same trajectories neither help nor overfit noticeably; both remain six to seven points below PhaseCoT (66.66), whose advantage holds on every benchmark. InftyThink is already close to PhaseCoT in budget at three epochs (within 9% in updates and 3% in loss tokens); a fourth epoch, which gives it 21% more updates and 29% more loss tokens than PhaseCoT, raises its average by only 0.3 points ($61.47 \rightarrow 61.78$), leaving a gap of 4.9 points. Extra optimization therefore does not account for the gains of PhaseCoT over either baseline.

6.2 Inference budget

Table 1 compares single-sample accuracy at each model's natural generation length, and PhaseCoT often generates more tokens than the baselines. Since additional test-time computation generally improves reasoning accuracy, we ask whether the baselines would catch up if allowed to spend the same number of tokens. We give every method the standard way of converting extra tokens into accuracy, majority voting over $k \in \{1, 2, 4, 8, 16\}$ samples per problem: the cost of a method is k times its mean generation length per sample, and its accuracy is that of the majority vote, estimated over random subsets of the 16 samples already used in Table 1. Figure 5 plots accuracy against tokens spent per problem for Qwen3-14B-Base; the leftmost point of each curve is the corresponding entry of Table 1, and a curve that lies above another delivers higher accuracy at equal cost. The comparison of interest is whether PhaseCoT's single-sample point lies above the Vanilla curve, in which case Vanilla cannot reach the same accuracy by spending the same tokens on additional samples.

This holds on all three benchmarks: Vanilla needs 2.4–$3.6\times$ as many tokens to match the single-sample accuracy of PhaseCoT, and its curve lies below that of PhaseCoT at every budget where the two overlap, with the gap widening as the budget grows on AIME22. InftyThink also trails PhaseCoT at small budgets, but the two iterative methods converge as k grows: on MATH500 and AMC23 they are within 0.6 points for $k \geq 8$, where majority voting approaches saturation, and on AIME22 they are tied at $k = 8$ before PhaseCoT leads by 3.3 points at $k = 16$. The accuracy gains of PhaseCoT are therefore not a by-product of its longer generations: spending the same tokens on additional Vanilla samples does not close the gap.

6.3 Component ablation

Having controlled the two budgets, we finally ask whether both components of PhaseCoT contribute, and whether they do so independently. PhaseCoT differs from InftyThink along two axes, *where to pause* (semantic milestones versus a length threshold) and *how to resume* (rewritten versus original segment heads), so we cross them into a 2×2 design. Each variant is named after the component it keeps: *PhaseCoT-Semantic* keeps semantic segmentation but removes segment-head rewriting, and *PhaseCoT-Rewrite* keeps rewriting but replaces semantic segmentation with the 4K length-based

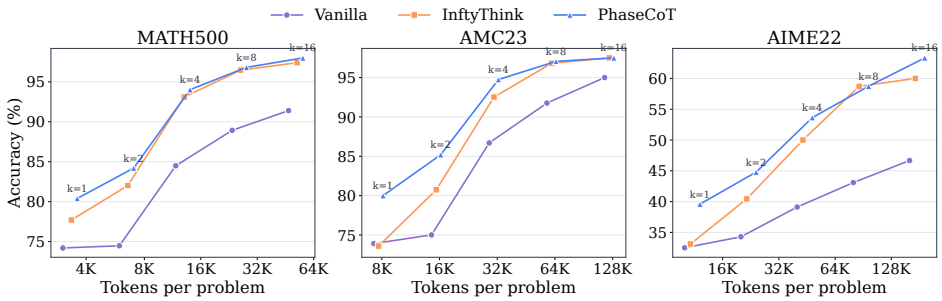


Figure 5: Accuracy versus test-time computation for Qwen3-14B-Base. Each point uses $k \in \{1, 2, 4, 8, 16\}$ samples per problem with majority voting; the horizontal axis is the number of generated tokens per problem (k times the mean sample length, log scale). The leftmost point of each curve corresponds to Table 1.

Table 3: Component ablation (Accuracy, %) on Qwen3-14B-Base, crossing the segmentation strategy with segment-head rewriting.

Train Format	Semantic seg.	Rewriting	MATH500	AIME22	AMC23	Average
InftyThink	–	–	77.70	33.13	73.59	61.47
PhaseCoT-Semantic	✓	–	79.10	35.63	77.50	64.08
PhaseCoT-Rewrite	–	✓	77.70	36.46	75.63	63.26
PhaseCoT	✓	✓	80.40	39.58	80.00	66.66

segmentation of InftyThink; removing both recovers InftyThink. All variants share the summarizer, the rewriter, the source pool, and the training recipe. Table 3 reports the results on Qwen3-14B-Base.

Both components help, and they help independently. Starting from InftyThink, semantic segmentation alone (PhaseCoT-Semantic) improves the average by 2.6 points with gains on all three benchmarks, and rewriting alone (PhaseCoT-Rewrite) by 1.8 points, leaving MATH500 unchanged and helping mainly on AIME22 and AMC23. The two effects are slightly super-additive on this backbone: rewriting adds 2.6 points on top of semantic segmentation versus 1.8 on top of length-based segmentation, and semantic segmentation adds 3.4 points on top of rewriting versus 2.6 without it, consistent with the motivation of Section 3 that a rewritten head is most useful when the pause it resumes from is a completed reasoning phase. Appendix E reports the 2×2 for all six backbones: each component improves the average accuracy of every backbone, their combination is the best variant for every backbone, and the larger rewriting gain under semantic segmentation holds for four of the six. Appendix F shows that the gain does not depend on the specific rewriting model.

7 Conclusion

We propose PhaseCoT, a human-inspired, data-centric refinement of length-driven iterative chain-of-thought along two dimensions: *where to pause* and *how to resume*. Semantic milestone-based segmentation aligns iteration boundaries with meaningful reasoning phases, and context-aware continuation lets the model re-establish context when resuming from summaries. On MATH500, AIME22, and AMC23, PhaseCoT achieves the highest average accuracy on every tested backbone and outperforms both baselines in most model-benchmark comparisons, suggesting that effective iterative reasoning depends not only on extending reasoning depth but also on structuring it around semantically coherent phases and grounded continuation.

Limitations Our study is confined to supervised fine-tuning; how much further gain reinforcement learning can unlock on top of PhaseCoT remains open. PhaseCoT also relies on external LLMs to identify milestones, summarize, and rewrite segment heads. Appendix H audits their outputs with rule-based checks, cross-model LLM judges, and a human audit; the audit is post hoc and was not used to filter the training data.

AI USE STATEMENT

In this work, we used generative AI tools to generate synthetic data sets and to clean and reformat a dataset: the PhaseCoT training data are constructed by LLMs as an integral part of the proposed method. Specifically, starting from 11,776 CoT examples sampled from OpenR1-Math-220K, Qwen3.5-397B-A17B identifies semantic milestones and segments each reasoning trajectory, Llama3.3-70B-Instruct generates the progress summary after each non-final segment, and GPT-OSS-120B (Llama3.1-70B-Instruct in the ablation of Appendix F) rewrites the head of each non-initial segment; the exact prompts are given in Appendices A–C. The same LLM-constructed data are used for the baselines and ablations to keep the comparison fair. We also used LLMs as judges to support the qualitative analysis of these data: the audit in Appendix H labels the constructed boundaries, summaries, and rewrites with LLM judges, each stage being judged by models that did not produce its outputs, and calibrates the judges against a human audit; the judge prompts are given in Appendix H.5. We also used LLM-based assistants to discuss and obtain feedback on the research methodology and experimental design; all design decisions were made by the authors. We have not used generative AI tools to develop theoretical models or conceptual frameworks, formulate mathematical claims, prove mathematical claims or write proofs, propose or refine hypotheses, implement methods, or interpret results. Assisting with translation is not applicable to this work.

Additionally, we used LLM-based assistants for tasks with recommended disclosure: to draft and edit parts of the paper to improve readability, including reformatting the manuscript to the ICLR template; to create or edit non-core software code, such as scripts for result aggregation, plotting, and LaTeX tables; and to create or modify scientific figures and tables.

We have reviewed all AI-assisted work. All AI-drafted or AI-edited text was reviewed sentence by sentence by the authors, who checked every claim and number against the experimental records and revised the wording. AI-assisted code was inspected and tested by the authors, and the figures and tables it produced were checked against the underlying results. The LLM-constructed training data are subject to the construction-time checks described in Appendix H (exact matching of milestone boundaries in the original CoT and normal termination of every LLM call) and to the post-hoc audit of Appendix H, which combines LLM judges with a human audit, and the model outputs in Appendices A and C are unedited examples of these data, and Appendix B shows the format of the summaries. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We have made efforts to ensure that the results in this paper can be reproduced. The complete data-construction procedure of PhaseCoT is specified in Section 3, and the exact prompts used for semantic segmentation, summarization, and segment-head rewriting, together with an unedited model output for segmentation and rewriting and the output format for summarization, are given in Appendices A–C. The data-quality audit, including the judge prompts and the human-audit protocol, is described in Appendix H. The source pool (11,776 examples sampled from the public OpenR1-Math-220K), the evaluation benchmarks (MATH500, AIME22, and AMC23), the six open-weight backbones, and the three compared training formats are described in Section 4.1; the training environment and all training and evaluation hyperparameters, as well as the supervision-budget accounting of the three formats, are listed in Appendix D (Tables 4–7). Full per-backbone results for the main experiments, the component ablation, and the rewriter ablation are reported in Appendices E–G. The code for the data-construction pipeline and a sample of the constructed training data are included in the supplementary material, and the complete constructed training data will be released publicly upon acceptance.

REFERENCES

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michał Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 17682–17690, 2024. doi: 10.1609/aaai.v38i16.29720.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

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

Daewon Choi, Jimin Lee, Jihoon Tack, Woomin Song, Saket Dingliwal, Sai Muralidhar Jayanthi, Bhavana Ganesh, Jinwoo Shin, Aram Galstyan, and Sravan Babu Bodapati. Think clearly: Improving reasoning via redundant token pruning. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 21437–21451, Suzhou, China, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-emnlp.1169.

Yingqian Cui, Pengfei He, Jingying Zeng, Hui Liu, Xianfeng Tang, Zhenwei Dai, Yan Han, Chen Luo, Jing Huang, Zhen Li, Suhang Wang, Yue Xing, Jiliang Tang, and Qi He. Stepwise perplexity-guided refinement for efficient chain-of-thought reasoning in large language models. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 18581–18597, Vienna, Austria, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-acl.956.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, 2025. doi: 10.1038/s41586-025-09422-z.

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. Token-budget-aware LLM reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 24842–24855, Vienna, Austria, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-acl.1274.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=tG4SgayTtk.

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. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/be83ab3ecd0db773eb2dc1b0a17836a1-Abstract-round2.html.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed prompting: A modular approach for solving complex tasks. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=_nGgzQjzaRy.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, et al. Kimi k1.5: Scaling reinforcement learning with LLMs. *arXiv preprint arXiv:2501.12599*, 2025.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems*, volume 35, pp. 22199–22213, 2022.

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. In *The Twelfth International Conference on Learning Representations*, 2024.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024. doi: 10.1162/tacl_a_00638.

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

Xinyin Ma, Guangnian Wan, Runpeng Yu, Gongfan Fang, and Xinchao Wang. CoT-Valve: Length-compressible chain-of-thought tuning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6025–6035, Vienna, Austria, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.300.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Mathematical Association of America. 2022 American Invitational Mathematics Examination I and II (AIME I/II). `https://artofproblemsolving.com/wiki/index.php/2022_AIME_I` and `https://artofproblemsolving.com/wiki/index.php/2022_AIME_II`, 2022. Accessed: 2026-05-02.

Mathematical Association of America. American Mathematics Competitions 2023. `https://maa.org/student-programs/amc/`, 2023. Accessed: 2026-05-02.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20275–20321, Suzhou, China, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.emnlp-main.1025.

Tergel Munkhbat, Namgyu Ho, Seo Hyun Kim, Yongjin Yang, Yujin Kim, and Se-Young Yun. Self-training elicits concise reasoning in large language models. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 25127–25152, Vienna, Austria, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-acl.1289.

Allen Newell and Herbert A. Simon. *Human Problem Solving*. Prentice-Hall, Englewood Cliffs, NJ, 1972.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021.

OpenAI, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, et al. OpenAI o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

George Pólya. *How to Solve It: A New Aspect of Mathematical Method*. Princeton University Press, Princeton, NJ, 1945.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 5687–5711, Singapore, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.378.

Ziqing Qiao, Yongheng Deng, Jiali Zeng, Dong Wang, Lai Wei, Guanbo Wang, Fandong Meng, Jie Zhou, Ju Ren, and Yaoxue Zhang. ConCISE: Confidence-guided compression in step-by-step efficient reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 8010–8029, Suzhou, China, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.emnlp-main.405.

Qwen Team. QwQ-32B: Embracing the power of reinforcement learning. `https://qwenlm.github.io/blog/qwq-32b/`, 2025. Accessed: 2026-09-26.

Matthew Renze and Erhan Guven. The benefits of a concise chain of thought on problem-solving in large language models. In *2024 2nd International Conference on Foundation and Large Language Models (FLLM)*, pp. 476–483. IEEE, 2024. doi: 10.1109/FLLM63129.2024.10852493.

Alan H. Schoenfeld. *Mathematical Problem Solving*. Academic Press, Orlando, FL, 1985. doi: 10.1016/C2013-0-05012-8.

Bilgehan Sel, Ahmad Tawaha, Vanshaj Khattar, Ruoxi Jia, and Ming Jin. Algorithm of thoughts: Enhancing exploration of ideas in large language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 44136–44189. PMLR, 2024.

Zhenyi Shen, Hanqi Yan, Linhai Zhang, Zhanghao Hu, Yali Du, and Yulan He. CODI: Compressing chain-of-thought into continuous space via self-distillation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 677–693, Suzhou, China, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.emnlp-main.36.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations*, 2025.

Vivek Vajipey, Aditya Tadimeti, Justin Shen, Ben Prystawski, Michael Y. Li, and Noah Goodman. Simple, scalable reasoning via iterated summarization. In *ICML 2025 Workshop on Long-Context Foundation Models*, 2025.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2609–2634, Toronto, Canada, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.147.

Yibo Wang, Haotian Luo, Huanjin Yao, Tiansheng Huang, Haiying He, Rui Liu, Naiqiang Tan, Jiaxing Huang, Xiaochun Cao, Dacheng Tao, and Li Shen. R1-Compress: Long chain-of-thought compression via chunk compression and search. *arXiv preprint arXiv:2505.16838*, 2025.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837, 2022.

Xixi Wu, Kuan Li, Yida Zhao, Liwen Zhang, Litu Ou, Huifeng Yin, Zhongwang Zhang, Xinmiao Yu, Dingchu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, Minhao Cheng, Shuai Wang, Hong Cheng, and Jingren Zhou. ReSum: Unlocking long-horizon search intelligence via context summarization. *arXiv preprint arXiv:2509.13313*, 2025.

Heming Xia, Chak Tou Leong, Wenjie Wang, Yongqi Li, and Wenjie Li. TokenSkip: Controllable chain-of-thought compression in LLMs. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 3351–3363, Suzhou, China, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.emnlp-main.165.

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

Yige Xu, Xu Guo, Zhiwei Zeng, and Chunyan Miao. 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)*, pp. 23336–23351, Vienna, Austria, 2025b. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.1137.

Yuchen Yan, Liang Jiang, Jin Jiang, Shuaicheng Li, Zujie Wen, Zhiqiang Zhang, Jun Zhou, Jian Shao, Yueting Zhuang, and Yongliang Shen. InftyThink+: Effective and efficient infinite-horizon reasoning via reinforcement learning. *arXiv preprint arXiv:2602.06960*, 2026a.

Yuchen Yan, Yongliang Shen, Yang Liu, Jin Jiang, Mengdi Zhang, Jian Shao, and Yueting Zhuang. InftyThink: Breaking the length limits of long-context reasoning in large language models. In *The Fourteenth International Conference on Learning Representations*, 2026b. URL https://openreview.net/forum?id=T1h5em349L.

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

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023b. URL https://openreview.net/forum?id=WE_vluYUL-X.

Jintian Zhang, Yuqi Zhu, Mengshu Sun, Yujie Luo, Shuofei Qiao, Lun Du, Da Zheng, Huajun Chen, and Ningyu Zhang. LightThinker: Thinking step-by-step compression. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 13307–13328, Suzhou, China, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.emnlp-main.673.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V. Le, and Ed H. Chi. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WZH7099tgfM.

A SEMANTIC SEGMENTATION PROMPT AND MODEL OUTPUT EXAMPLE

Prompt The following is the user prompt used for semantic segmentation, where `[ORIGINAL_COT]` is replaced by the full source chain-of-thought.

```
Original Chain-of-Thought:
[ORIGINAL_COT]

You are given a long chain-of-thought (CoT) and must divide it into
sequential milestones.

Requirements:
1) Each item must contain exactly these fields: milestone_idx,
end_anchor, summary.
2) milestone_idx starts at 1 and increases consecutively.
3) end_anchor must be the final sentence of that milestone, copied
verbatim as a contiguous span from the original CoT. Do not
paraphrase, rewrite, compress, normalize punctuation, or clean up
awkward wording; if the original sentence is repetitive or messy, copy
it exactly as written.
4) Milestones must be non-overlapping, contiguous, and together cover
the entire CoT. They must follow the original CoT strictly in reading
order: the first milestone starts at the beginning of the CoT, and
each subsequent milestone starts immediately after the previous
milestone's end_anchor.
5) summary: one sentence describing the reasoning accomplished in this
milestone.
6) Each milestone should correspond to a self-contained reasoning
phase (e.g., problem decomposition, case analysis, computation,
verification). Aim for 3-8 milestones for a typical CoT.
7) Do NOT create a milestone solely for opening remarks where the
model restates or paraphrases the problem conditions without
contributing new reasoning; instead, absorb that text into the first
substantive milestone.
8) Do NOT stop the final milestone early. The final milestone must end
at the actual final sentence of the CoT, including any closing
verification, restatement, or final-answer formatting if it is part of
the original text.
9) Before outputting the JSON, check that every end_anchor appears
verbatim in the original CoT and that the final end_anchor reaches the
end of the CoT.
10) Output the result as a JSON array wrapped in a ```json code fence.
```

Model output example The segmentation model returns a JSON array. Each item contains a milestone index, an ending anchor copied verbatim from the original CoT, and a short description of the reasoning completed in that milestone. The following is an example output.

````
```json
[
  {
    "milestone_idx": 1,
    "end_anchor": "Therefore, the GCD is 17, not higher powers.",
    "summary": "The model tests specific cases and small primes to
identify 17 as a common divisor and verifies that 17^2 does not divide
the terms."
  },
  {
    "milestone_idx": 2,
    "end_anchor": "Hence, 17 divides all such numbers.",
    "summary": "The model proves generally that 17 divides the
expression for all integers a, b, c by analyzing modular arithmetic
and sum of squares properties."
````
````

```
  },
  {
    "milestone_idx": 3,
    "end_anchor": "\\boxed{17}",
    "summary": "The model verifies no other primes divide all terms
and confirms 17 is the greatest common divisor."
  }
]
```

B SUMMARIZATION PROMPT AND MODEL OUTPUT EXAMPLE

Prompt The summarization prompt is represented as a three-message chat. Here `[PROBLEM]` denotes the original problem statement, and `[COT_PREFIX]` denotes the concatenation of all reasoning segments up to the current prefix boundary.

```
[
  {
    "role": "user",
    "content": "[PROBLEM]"
  },
  {
    "role": "assistant",
    "content": "[COT_PREFIX]"
  },
  {
    "role": "user",
    "content": "Please list what you have achieved in your last
response. Output only the summary. Include all key steps and important
intermediate conclusions. Prefix each bullet point with '*'."
  }
]
```

Model output example The summarization model is instructed to output only the summary, formatted as bullet points beginning with $\star$. The following shows the expected output format.

```
* Identified the main constraints from the problem statement.
* Derived the key intermediate equations needed for the next stage.
* Eliminated impossible cases using the divisibility and digit-sum
conditions.
* Found the remaining candidate structure to continue the solution.
```

C SEGMENT REWRITING PROMPT AND MODEL OUTPUT EXAMPLE

Prompt The following is the user prompt used for segment rewriting. Here `[PROBLEM]` is the original math problem, `[HISTORY]` is the compressed summary of previous reasoning, and `[CONTINUATION]` is the extracted segment head.

```
Given a math problem as follows:
<problem>
[PROBLEM]
</problem>

The problem was fed to a model, which produced a complete reasoning
process. This reasoning process was then manually split into two
parts: history and continuation. Since these two parts were obtained
by cutting a single continuous reasoning process, the transition
between them was originally natural and smooth.
```

```
Now, history has been summarized by a summarization model into a
sequence of summaries as follows:
<history>
[HISTORY]
</history>

The continuation is included as follows:
<continuation>
[CONTINUATION]
</continuation>

Because history has been turned into a compressed summary that retains
only core reasoning steps and key intermediate conclusions, the
transition from history to continuation is no longer natural or
smooth. Our goal is to use SFT to train a model to continue solving
the problem based on the problem statement and the compressed history.
Ideally, after being given the problem and the compressed history, the
model should generate a natural and coherent reasoning process, rather
than abruptly producing the original continuation.

Therefore, please reformulate the continuation so that it connects
naturally and smoothly with the compressed history. Please follow
these requirements:

1) The reformulated continuation must begin with a brief recap of the
problem and the compressed history.
2) The reformulated continuation should then slightly rewrite the
beginning of the original continuation while keeping the rest
unchanged. The rewrite should be as minimal as possible, just enough
to make the transition from the recap to the rest of the segment head
natural and smooth.
3) Do not introduce any new reasoning content that does not appear in
the continuation.
4) Enclose the reformulated continuation within <reformulated> and
</reformulated> tags.
```

Model output example The following is an example output of the rewriting model.

```
<reformulated>
Brief recap: We need the smallest number of distinct weight
denominations to let the squirrel prove that his 15 nuts weigh 50 g,
51 g, ..., 64 g, given that the friends initially know nothing about
the nuts. The earlier reasoning noted that the squirrel can use
already-proven nuts as known references, considered ternary
representations, and finally observed that a single 1 g weight
suffices, leading to the claim that the minimal number of distinct
weights is 1.

But wait, that seems contradictory to my previous reasoning. Let me
check again.
</reformulated>
```

D Training and Evaluation Environment and Parameters

Table 4 gives the supervision-budget accounting of the three training formats used in Section 6.1; Tables 5–7 list the training and evaluation configuration.

Table 4: Supervision budget of the three training formats (same 11,776 source trajectories). Loss tokens: assistant-side tokens on which the SFT loss is computed; total tokens additionally count the prompt side (problem, instruction, and history summary); counts use the Qwen3 tokenizer and exclude the chat template. Steps follow from a global batch of 256 instances and three epochs.

Train Format	**Instances**	**Loss tok. (M)**	**Total tok. (M)**	**Steps**	**Epochs**
Vanilla	11,776	90.8	92.4	138	3
InftyThink	20,977	94.0	101.0	246	3
PhaseCoT	22,816	97.1	105.0	270	3

Table 5: Training environment and parameters. Vanilla CoT values are shown first; iterative CoT (InftyThink and PhaseCoT) values are shown in parentheses when they differ.

Parameter / Environment Information	Value
GPU type and memory	A100, 80 GB
Number of nodes	2
GPUs per node	4
Global train batch size	256
Micro batch size per GPU	4
Total epochs	3
Maximum sequence length	20480 (10240)
Maximum token length per GPU	10240 (5120)
Dynamic token batch sizing	True
Optimizer	AdamW
Learning rate	`1e-5`
Learning-rate warmup ratio	0.03
Learning-rate scheduler	`cosine`
Minimum learning-rate ratio	0.1
Weight decay	0.01
Adam betas	$\beta_1 = 0.9$, $\beta_2 = 0.999$
Gradient clipping	1.0
Engine	FSDP
FSDP dtype	`bfloat16`
Model dtype in engine configuration	`fp32`
Ulysses sequence parallel size	2
Gradient checkpointing	True
LoRA rank	0

Table 6: Training time (minutes) of the base models under a 2×4 A100 (80 GB) configuration; the instruct models exhibit similar patterns.

Model	Vanilla	InftyThink	PhaseCoT
Llama3.1-8B	433	488	514
Qwen3-8B-Base	511	572	585
Qwen3-14B-Base	621	703	729

Table 7: Evaluation generation parameters. Vanilla CoT values are shown first; iterative CoT (InftyThink and PhaseCoT) values are shown in parentheses when they differ.

Parameter	Value
Top-p	1.0
Temperature	0.7
Number of samples	16
Prompt length	1024 (4096)
Response length	16384 (8192)
Maximum model length	17408 (12288)
Maximum batched tokens	17408 (12288)
Rollout backend	`vllm`
GPU memory utilization	0.8

E Full Component Ablation Results

Table 8 extends the 2×2 component ablation of Section 6.3 to all six backbones. Each component helps on every backbone: relative to InftyThink, PhaseCoT-Semantic raises the average accuracy of all six backbones (by 0.7–2.6 points) and so does PhaseCoT-Rewrite (by 0.8–3.0 points); at the

model–benchmark level, PhaseCoT-Semantic and PhaseCoT-Rewrite win 15 and 14 of the 18 comparisons, respectively, each with 2 ties. Combining both is the best variant for every backbone: PhaseCoT improves over PhaseCoT-Semantic in 17 of the 18 pairs (with 1 tie) and over PhaseCoT-Rewrite in 16. The relative size of the two effects depends on the backbone—semantic segmentation contributes more on Llama3.1-8B, Qwen3-14B-Base, Llama3.1-8B-Instruct, and Qwen3-8B, whereas rewriting contributes more on Qwen3-8B-Base and Qwen3-14B—and the gain from adding rewriting is larger under semantic segmentation than under length-based segmentation on four of the six backbones (all but Qwen3-8B-Base and Qwen3-8B).

Table 8: Full component ablation results (Accuracy, %) for all backbones. PhaseCoT-Semantic keeps semantic segmentation without rewriting; PhaseCoT-Rewrite keeps rewriting with length-based segmentation; InftyThink has neither.

Model	Train Format	MATH500	AIME22	AMC23	Average
Base Models					
Llama3.1-8B	InftyThink	51.10	19.17	55.00	41.76
	PhaseCoT-Semantic	53.43	20.00	58.13	43.85
	PhaseCoT-Rewrite	51.83	19.38	57.97	43.06
	PhaseCoT	55.18	20.63	60.16	45.32
Qwen3-8B-Base	InftyThink	64.00	22.29	62.97	49.75
	PhaseCoT-Semantic	65.33	23.13	66.09	51.52
	PhaseCoT-Rewrite	66.94	24.58	66.72	52.75
	PhaseCoT	68.80	23.33	67.66	53.26
Qwen3-14B-Base	InftyThink	77.70	33.13	73.59	61.47
	PhaseCoT-Semantic	79.10	35.63	77.50	64.08
	PhaseCoT-Rewrite	77.70	36.46	75.63	63.26
	PhaseCoT	80.40	39.58	80.00	66.66
Instruct Models					
Llama3.1-8B-Instruct	InftyThink	60.10	17.92	56.88	44.97
	PhaseCoT-Semantic	59.83	17.92	60.47	46.07
	PhaseCoT-Rewrite	58.93	18.33	60.00	45.75
	PhaseCoT	60.28	17.92	62.97	47.06
Qwen3-8B	InftyThink	76.80	25.83	76.25	59.63
	PhaseCoT-Semantic	78.10	29.79	78.13	62.01
	PhaseCoT-Rewrite	76.80	28.33	77.97	61.03
	PhaseCoT	79.00	31.67	79.22	63.30
Qwen3-14B	InftyThink	83.60	40.00	82.97	68.86
	PhaseCoT-Semantic	84.10	40.00	84.69	69.60
	PhaseCoT-Rewrite	83.25	43.33	85.94	70.84
	PhaseCoT	85.20	44.17	86.09	71.82

F Ablation Results with Llama3.1-70B-Instruct as the Rewriting Model

Table 9 repeats the main comparison with Llama3.1-70B-Instruct in place of GPT-OSS-120B as the segment-head rewriting model.

Table 9: Ablation results using Llama3.1-70B-Instruct as the segment-head rewriting model. ACC denotes average accuracy (%), TOK denotes the average number of generated tokens (K), and Average is computed over the three benchmarks.

Model	Train Format	MATH500		AIME22		AMC23		Average	
		ACC	TOK	ACC	TOK	ACC	TOK	ACC	TOK
Base Models									
Llama3.1-8B	Vanilla	50.80	1.74	18.54	6.05	54.53	4.52	41.29	4.10
	InftyThink	51.10	1.91	19.17	6.63	55.00	5.03	41.76	4.52
	PhaseCoT	54.76	2.01	20.21	6.96	59.53	4.91	44.83	4.63
Qwen3-8B-Base	Vanilla	67.00	2.23	24.38	8.68	67.03	6.27	52.80	5.73
	InftyThink	64.00	2.13	22.29	7.00	62.97	4.42	49.75	4.52
	PhaseCoT	68.21	2.13	23.13	11.42	67.19	5.86	52.84	6.47
Qwen3-14B-Base	Vanilla	74.20	3.01	32.50	10.26	73.91	7.44	60.20	6.90
	InftyThink	77.70	3.34	33.13	11.02	73.59	7.88	61.47	7.41
	PhaseCoT	79.84	3.55	38.96	12.21	79.22	8.24	66.01	8.00
Instruct Models									
Llama3.1-8B-Instruct	Vanilla	59.60	2.02	18.33	7.08	58.59	5.32	45.51	4.81
	InftyThink	60.10	2.26	17.92	7.62	56.88	5.81	44.97	5.23
	PhaseCoT	60.23	2.31	17.92	7.98	62.19	5.79	46.78	5.36
Qwen3-8B	Vanilla	77.40	2.88	28.13	10.06	76.56	7.41	60.70	6.78
	InftyThink	76.80	3.16	25.83	10.93	76.25	7.86	59.63	7.32
	PhaseCoT	78.63	3.05	31.04	12.20	78.91	8.54	62.86	7.93
Qwen3-14B	Vanilla	83.00	3.51	40.63	12.44	82.34	9.22	68.66	8.39
	InftyThink	83.60	3.79	40.00	13.10	82.97	9.84	68.86	8.91
	PhaseCoT	84.88	4.01	43.54	14.60	85.47	10.33	71.30	9.65

G Full Main Experimental Results

Table 10 reports the main experimental results for all six backbones, including the instruct models.

Table 10: Main experimental results. The results are obtained by sampling each model 16 times with a temperature of 0.7. ACC denotes average accuracy (%), TOK denotes the average number of generated tokens (K), and Average is computed over the three benchmarks. Standard deviations across the 16 decoding samples (one training run per configuration) are below 0.9% for accuracy and below 0.15K for token length in all reported cases.

Model	Train Format	MATH500		AIME22		AMC23		Average	
		ACC	TOK	ACC	TOK	ACC	TOK	ACC	TOK
Base Models									
Llama3.1-8B	Vanilla	50.80	1.74	18.54	6.05	54.53	4.52	41.29	4.10
	InftyThink	51.10	1.91	19.17	6.63	55.00	5.03	41.76	4.52
	PhaseCoT	55.18	2.04	20.63	7.08	60.16	4.96	45.32	4.69
Qwen3-8B-Base	Vanilla	67.00	2.23	24.38	8.68	67.03	6.27	52.80	5.73
	InftyThink	64.00	2.13	22.29	7.00	62.97	4.42	49.75	4.52
	PhaseCoT	68.80	2.17	23.33	11.71	67.66	5.91	53.26	6.60
Qwen3-14B-Base	Vanilla	74.20	3.01	32.50	10.26	73.91	7.44	60.20	6.90
	InftyThink	77.70	3.34	33.13	11.02	73.59	7.88	61.47	7.41
	PhaseCoT	80.40	3.59	39.58	12.37	80.00	8.31	66.66	8.09
Instruct Models									
Llama3.1-8B-Instruct	Vanilla	59.60	2.02	18.33	7.08	58.59	5.32	45.51	4.81
	InftyThink	60.10	2.26	17.92	7.62	56.88	5.81	44.97	5.23
	PhaseCoT	60.28	2.34	17.92	8.05	62.97	5.88	47.06	5.42
Qwen3-8B	Vanilla	77.40	2.88	28.13	10.06	76.56	7.41	60.70	6.78
	InftyThink	76.80	3.16	25.83	10.93	76.25	7.86	59.63	7.32
	PhaseCoT	79.00	3.08	31.67	12.39	79.22	8.65	63.30	8.04
Qwen3-14B	Vanilla	83.00	3.51	40.63	12.44	82.34	9.22	68.66	8.39
	InftyThink	83.60	3.79	40.00	13.10	82.97	9.84	68.86	8.91
	PhaseCoT	85.20	4.07	44.17	14.72	86.09	10.40	71.82	9.73

H Quality Control of the LLM-Constructed Supervision

PhaseCoT builds its supervision with three LLM calls: the segmenter places the phase boundaries (Appendix A), the summarizer writes the cumulative summaries S_i (Appendix B), and the rewriter produces the rewritten segment heads $\widetilde{H}_i$ (Appendix C). Over the 11,776 source trajectories, these calls produce 11,040 phase boundaries (the 22,816 PhaseCoT instances of Table 4 minus one instance per trajectory). Each boundary is followed by one summary and, in all but 11 cases, by one rewritten head; in those 11 cases the rewriter output lacked the required tags and the original head is kept. We audit the outputs of each call at three levels:

1. *Rule-based checks* on all 11,040 summaries and all 11,029 rewritten heads, and on the boundaries of the trajectories sampled in Appendix H.1.
2. *LLM judges* on random samples. Each stage is judged by the two pipeline models that did not produce its outputs, so that no model grades its own outputs: boundaries by GPT-OSS-120B and Llama3.3-70B-Instruct, summaries by GPT-OSS-120B and Qwen3.5-397B-A17B, and rewrites by Qwen3.5-397B-A17B and Llama3.3-70B-Instruct.
3. A *human audit* of a subset, which calibrates the judges and yields population-level error estimates (Appendix H.4).

The audit is post hoc: the training data of all experiments in this paper passed only two rule-based checks during construction, besides the tag fallback of the rewriter above—each milestone anchor

must occur verbatim in the original CoT, and each LLM call must terminate normally rather than at the output-length limit—and no outcome of the audit was used to filter or modify them.

H.1 BOUNDARY QUALITY

Section 5.1 shows that semantic segmentation changes the distribution of segment lengths, but a different length distribution does not by itself show that the boundaries are semantic milestones. We therefore evaluate the boundaries directly against the two conditions that define a milestone in Section 3: the prefix closes a self-contained reasoning phase, and the suffix begins a new reasoning operation.

Segmentations compared On the same 200 randomly sampled multi-phase trajectories, we compare three segmentations: *Semantic*, the PhaseCoT boundaries; *Length*, the length-based boundaries of InftyThink, each placed at the first sentence end after 4K tokens; and *Random-matched*, a control that keeps the number of phases K of each trajectory, draws its segment lengths from the empirical distribution of PhaseCoT segment lengths, rescales them to sum to the length of the trajectory, and moves each cut to the nearest paragraph end. Random-matched thus controls for the number of boundaries and their length distribution and, like the other two segmentations, never cuts inside a sentence, so that a remaining difference from Semantic reflects where the boundaries are placed rather than how many there are or how long the segments are. Summaries for the Random-matched boundaries are generated with the same summarizer and prompt as for the other two segmentations (Appendix B).

Metrics We use three complementary measures. *(i) Structural checks*, computed on all boundaries of the sampled trajectories: the share of cuts that leave an open computation—inside an unclosed math environment, or right after a line that ends with "=" or a binary operator—and the share of next segments whose first sentence begins with a continuation connective (e.g., "So", "Thus", "Then", "Therefore"). Semantic cuts follow a complete sentence by design, since each anchor is the final sentence of a milestone; Length cuts follow the first sentence end after 4K tokens and Random-matched cuts a paragraph end, so none of the three segmentations cuts inside a sentence. *(ii) Judged milestone validity*: the two defining conditions, asked as binary questions. *Closure*: does the reasoning before the cut leave no derivation, computation, or case unfinished? *Fresh start*: does the text after the cut begin a new reasoning operation rather than continue the current one? The judge sees the problem, about 400 tokens before the cut, and about 200 tokens after it, formatted identically for all segmentations and without any indication of which segmentation produced the cut; we judge 300 boundaries per segmentation. *(iii) Resumption gap*, a judge-free measure of how much the next phase still depends on the discarded history once the summary is given:

$$\Delta_i \;=\; \mathrm{NLL}\big(P_{i+1}^{[1:256]} \,\big|\, q, S_i\big) \;-\; \mathrm{NLL}\big(P_{i+1}^{[1:256]} \,\big|\, q, P_{\le i}\big), \tag{4}$$

where $P_{i+1}^{[1:256]}$ denotes the first 256 tokens of the original (not rewritten) next segment and NLL is the mean per-token negative log-likelihood under Qwen3-14B-Base without fine-tuning. Δ_i is small when the summary retains what the next phase needs, and scoring the original segment keeps rewriting out of the comparison. Because a formulaic opening sentence is easy to predict under either context, we also report Δ_i computed only on the scored tokens after the first sentence of P_{i+1}.

Table 11: Boundary quality of the three segmentations on the same trajectories. Open comp.: the cut leaves an open computation; Cont. opener: the next segment opens with a continuation connective. Structural checks and Δ (Equation (4); nats per token) use all boundaries of the sampled trajectories; Closure and Fresh start are judged on 300 boundaries per segmentation by GPT-OSS-120B / Llama3.3-70B-Instruct. ↓: lower is better; ↑: higher is better.

	Structural (%) ↓		Judged (%) ↑		Δ ↓	
Segmentation	Open comp.	Cont. opener	Closure	Fresh start	All tokens	After 1st sent.
Random-matched	6.1	22.4	58.3 / 61.0	48.7 / 46.0	0.12	0.14
Length (InftyThink)	8.2	25.3	40.7 / 43.3	37.7 / 35.0	0.18	0.21
Semantic (PhaseCoT)	1.8	7.9	90.3 / 88.7	85.0 / 82.3	0.07	0.08

Test-time pauses The same checks apply to the pauses that the trained models make on their own. For the Qwen3-14B-Base models trained with InftyThink and with PhaseCoT, we take the round boundaries of their generations on AIME22 and AMC23, i.e., the points where a non-final round stops to emit its summary, in the same generations as Figures 3 and 4 (those in which every round terminates normally: 126 and 209 of the 480 AIME22 generations and 356 and 441 of the 640 AMC23 generations for InftyThink and PhaseCoT, respectively), and apply the Closure judgment and the open-computation check. Fresh start is not judged here: PhaseCoT rounds open with a recap by design, which would favor PhaseCoT regardless of where the previous round stopped.

Table 12: Quality of the pauses made at test time by Qwen3-14B-Base trained with InftyThink and with PhaseCoT. Closure is judged by GPT-OSS-120B / Llama3.3-70B-Instruct.

Benchmark	Train format	Round boundaries	Closure (%) ↑	Open comp. (%) ↓
AIME22	InftyThink	51	42.3 / 44.8	22.1
	PhaseCoT	271	85.2 / 83.0	3.1
AMC23	InftyThink	76	45.0 / 47.2	19.7
	PhaseCoT	213	87.4 / 85.6	2.6

Semantic boundaries satisfy both milestone conditions far more often than either control: the judges rate 90.3% / 88.7% of them as closed and 85.0% / 82.3% as fresh starts, compared with at most 61.0% and 48.7% for Random-matched, which has the same number of boundaries, length distribution, and paragraph alignment. The judge-free resumption gap agrees, with the smallest Δ for Semantic both on all scored tokens and after the first sentence. At test time, 83.0%–87.4% of the pauses of the PhaseCoT model are judged closed, compared with 42.3%–47.2% for InftyThink, so the trained model pauses at milestones rather than after a fixed length.

H.2 Summary fidelity

A summary S_i should be faithful to the reasoning it compresses and retain what later phases need. The bounded-context format adds a third requirement. At inference, a middle round produces S_i after seeing only q, S_{i-1}, and its own segment, whereas the training target $S_i = \mathcal{G}_{\text{sum}}(q, P_{\le i})$ is written from the full prefix (Equation (1)). Content of S_i that is supported by $P_{<i}$ but absent from $(q, S_{i-1}, \widetilde{P}_i)$ cannot be derived from the model's input and would teach the model to state facts it cannot see; we call such content *conditionally inconsistent*. The summarizer and its prompt are shared by InftyThink, PhaseCoT, and all ablation variants; the audit below covers the PhaseCoT summaries.

Object matching The automatic checks compare texts through their mathematical objects, matched exactly after normalization. *Extraction.* Math spans are the contents of `$...$`, `$$...$$`, `\(...\)`, `\[...\]`, and the environments `equation`, `align`, and `gather` (with or without `*`). A span is split into statements at top-level commas, semicolons, and `\text{...}`, and each statement is split into terms at top-level relation symbols ($=$, $\neq$, $<$, $>$, $\le$, $\ge$, $\approx$, $\equiv$). The objects of a span are its terms and the relations between adjacent terms; in a chain of equalities, every pair of terms forms a relation. Numbers outside math spans (integers, decimals, and fractions a/b of integers) are objects as well, except enumeration markers at the start of a line and numbers directly after *step*, *case*, *part*, or *phase*. Single variables (one letter, optionally primed) and single-digit integers are discarded as terms because they occur almost everywhere, but relations that contain them, such as $x = 3$, are kept. Formulas written without math delimiters contribute only their numbers. *Normalization.* We remove whitespace, spacing commands (`\,`, `\;`, `\quad`, ...), sizing commands (`\left`, `\right`, `\big`, ...), `\displaystyle`, and braces around a single token; map `\dfrac` and `\tfrac` to `\frac`, `\le`, `\ge`, and `\ne` to `\leq`, `\geq`, and `\neq`, and `\cdot` and `\times` to `*`; remove thousands separators; and write every numeric literal (an integer, a terminating decimal, or `\frac` or a/b of two integers) as a reduced fraction or an integer, so that `0.5`, `1/2`, and `\frac{2}{4}` coincide. Relations are written in a canonical orientation: the two sides of $=$, $\neq$, $\approx$, and $\equiv$ are sorted lexicographically, and $a < b$ and $a \le b$ are rewritten as $b > a$ and $b \ge a$. *Matching.* An object occurs in a text if it is one of the text's objects; a term that is not a number also occurs if it is a substring of one of the text's normalized math spans. An object occurs in a set of

texts if it occurs in any of them. All rates are pooled over the objects (or heads) of the whole data set, and all checks use the original segments P_j unless $\widetilde{P}_i$ is written.

Automatic checks Over all 11,040 summaries, 0.42% of the objects of S_i occur in neither q nor $P_{\leq i}$ (*unsupported*); of the objects of $P_{\leq i}$ that do not occur in q and occur again in a later segment P_j, $j > i$, 82.6% occur in S_i (*forward recall*); and 0.81% of the objects of S_i occur in q or $P_{\leq i}$ but in none of q, S_{i-1}, and $\widetilde{P}_i$ (*conditionally inconsistent*).

Judged checks The judges label each bullet of S_i as supported, unsupported, or contradicted by $P_{\leq i}$; flag bullets that are not supported by $(q, S_{i-1}, \widetilde{P}_i)$; and, given the next segment P_{i+1}, list the constraints or intermediate results that P_{i+1} uses but S_i omits. Table 13 reports the results.

Table 13: Judged summary fidelity (%). Unsupported, Contradicted, and Cond. incons. (conditionally inconsistent) are rates over bullets; Missing info is the share of summaries that omit a constraint or intermediate result used by the next segment.

Annotator	n	Unsupported ↓	Contradicted ↓	Cond. incons. ↓	Missing info ↓
GPT-OSS-120B	500	0.10	0.05	0.10	15.2
Qwen3.5-397B-A17B	500	0.08	0.05	0.08	13.6
Human	150	0.08	0.00	0.08	12.0

Unsupported, contradicted, and conditionally inconsistent content is rare. The automatic checks flag 0.42% of the summary objects as unsupported and 0.81% as conditionally inconsistent; these rates are higher than the judged ones because exact object matching also flags values obtained by simple arithmetic or a change of notation, which the judge instructions count as supported and visible (Appendix H.5). According to either judge, each type occurs in at most 0.10% of bullets, and the annotators found one unsupported and one conditionally inconsistent bullet and no contradicted bullet among the bullets of 150 summaries. The training targets therefore almost never require the model to state facts outside its bounded context.

H.3 Rewrite fidelity

The rewriter may change only the opening paragraph H_i of a segment: it prepends a brief recap of the problem and the summary and must not introduce reasoning that is absent from H_i (Appendix C), while the tail T_i is copied verbatim by construction (Equation (2)).

Automatic checks Over all 11,029 rewritten heads, 98.1% of the mathematical objects of H_i are retained in $\widetilde{H}_i$; 2.9% of the rewritten heads contain an object that occurs in none of q, S_{i-1}, and H_i, i.e., content introduced by the rewriter; and the part of $\widetilde{H}_i$ after the recap differs from H_i by a median normalized edit distance of 0.08. Objects are matched as in Appendix H.2. The recap is the first paragraph of $\widetilde{H}_i$ (see the example in Appendix C); if $\widetilde{H}_i$ has a single paragraph, the recap is the text before the longest common substring of $\widetilde{H}_i$ and H_i. The rest of $\widetilde{H}_i$ is its body, and the normalized edit distance is the character-level Levenshtein distance between the body and H_i after collapsing whitespace, divided by the length of the longer of the two.

Judged checks The judges answer four binary questions per rewrite: whether the mathematical meaning of H_i is changed; whether a claim or conclusion absent from H_i is introduced; whether the recap is supported by q and S_{i-1}; and whether the transition from the recap into the rest of the segment reads naturally. Table 14 reports the results; the human row gives the rates observed in the human audit (Appendix H.4).

Table 14: Judged rewrite fidelity (% of rewrites). Meaning chg.: the mathematical meaning of H_i is changed; New claim: a claim or conclusion absent from H_i is introduced; Unsupp. recap: the recap is not supported by q and S_{i-1}; Natural: the transition reads naturally.

Annotator	n	Meaning chg. ↓	New claim ↓	Unsupp. recap ↓	Natural ↑
Qwen3.5-397B-A17B	500	1.6	2.4	3.0	93.2
Llama3.3-70B-Instruct	500	1.2	2.8	3.6	94.6
Human	150	0.7	2.0	2.0	95.3

Rewrites preserve the mathematical content of the original heads: 98.1% of their mathematical objects are retained, and the text after the recap differs from H_i by a median normalized edit distance of 0.08. The annotators found one change in mathematical meaning among 150 rewrites (0.7%; 95% upper bound 3.7%), consistent with the judged rates of 1.2%–1.6%.

H.4 Human audit and judge calibration

Two annotators (authors of this paper) audited 200 boundaries (100 Semantic, 50 Length, and 50 Random-matched cuts, shuffled together), 150 summaries, and 150 rewrites with the same questions as the judges, without access to the judge labels and, for boundaries, without knowing which segmentation produced a cut. Disagreements between the annotators were resolved by discussion, and the adjudicated labels are used below. For each stage, items were sampled uniformly at random (for boundaries, within each segmentation), so the observed error rates directly estimate the population rates. Intervals for error rates are Wilson 95% intervals; when no error is found among n audited items, the one-sided 95% upper bound is approximately $3/n$ (the rule of three). Table 15 reports inter-annotator agreement, judge–human agreement, and the estimated population error rates.

Table 15: Human audit. κ: Cohen's kappa between the two annotators (H–H) and between each judge and the adjudicated human labels (J1–H, J2–H), where J1 and J2 are the two judges of the stage. Error: estimated population share of outputs that fail the criterion, with a 95% interval. For boundaries, κ is computed on all 200 audited cuts of the three segmentations, and n and Error refer to the 100 Semantic cuts. Criteria as in Tables 11, 13, and 14.

Stage	Criterion	n	κ H–H	κ J1–H	κ J2–H	Error (%) [95% CI]
Boundaries	Closure (Semantic)	100	0.81	0.72	0.68	10.0 [5.5, 17.4]
	Fresh start (Semantic)	100	0.76	0.66	0.63	13.0 [7.8, 21.0]
Summaries	Unsupp. or contrad.	150	0.74	0.65	0.61	0.7 [0.1, 3.7]
	Cond. incons.	150	0.71	0.62	0.58	0.7 [0.1, 3.7]
Rewrites	Meaning chg.	150	0.66	0.49	0.44	0.7 [0.1, 3.7]
	New claim	150	0.70	0.60	0.57	2.0 [0.7, 5.7]

Scope of the audit The judges never grade their own outputs, but they come from the same model families as the pipeline, and the human audit that calibrates them covers a small sample. The audit characterizes the PhaseCoT training data of the main experiments (not the summaries of the other formats or the rewrites of Appendix F) but does not filter it; we do not retrain on audit-filtered data in this paper.

H.5 Judge prompts

The exact prompts of the three judges are listed below; both judges of a stage receive the same prompt.

Boundary judge This prompt is used both for the constructed boundaries (Table 11) and for the test-time pauses (Table 12), where only the closure answer is used. `[PROBLEM]` is the problem statement, `[BEFORE_CUT]` the last ∼400 tokens before the cut, and `[AFTER_CUT]` the first ∼200 tokens after it (for a test-time pause, the beginning of the next round).

```
You will evaluate a single cut point in a step-by-step solution to a
math problem. The solution was cut into two parts, and you are shown
```

```
the problem, an excerpt of the solution immediately before the cut,
and an excerpt immediately after the cut.

<problem>
[PROBLEM]
</problem>

<before_cut>
[BEFORE_CUT]
</before_cut>

<after_cut>
[AFTER_CUT]
</after_cut>

The excerpts are truncated for length: the text in <before_cut> may
begin in the middle of a sentence, and the text in <after_cut> may end
in the middle of a sentence. Only the boundary between the end of
<before_cut> and the start of <after_cut> is the cut to be evaluated;
ignore the truncated outer edges.

Answer two questions about the cut. Judge only where the cut is
placed. Do not judge whether the mathematics is correct, and do not
judge the writing quality.

Q1 (closure). Is the reasoning step that is in progress just before
the cut finished at the cut? Answer "yes" if the last derivation,
computation, or case being worked on has reached its result or
stopping point, so that nothing that was started is left half-done.
Answer "no" if the cut interrupts it, for example: a sentence or
formula is incomplete; an equation or expression is being written or
simplified and its result has not yet been stated; a computation has
been set up but not carried out; a case is being analyzed but has not
reached its conclusion; or the text right before the cut announces
what the current step is about to do (e.g., "Let's compute this:")
without doing it. The overall problem does not need to be solved at
the cut, and later cases or sub-goals that have only been announced do
not count as unfinished.

Q2 (fresh start). Does the text right after the cut begin a new
reasoning operation? Answer "yes" if it starts a new step, such as a
new sub-goal, a new case, a new approach, a verification or check of
earlier work, or the final answer. Answer "no" if it continues the
operation that was in progress before the cut, for example: it
completes the preceding sentence or formula; it continues the same
calculation or simplification; it states the result of the computation
that was in progress; or it only adds a remark to the immediately
preceding line. An opening word such as "So", "Now", or "Wait" does
not decide the answer by itself; judge what the text after the cut
actually does.

Write a brief justification of at most three sentences. Then output
your answers as a JSON object wrapped in a ```json code fence, where
each value is "yes" or "no":
```json
{"closure": "yes", "fresh_start": "no"}
```
```

Summary judge `[COT_PREFIX]` is $P_{\leq i}$; `[SUMMARY]` is S_i with its bullets numbered `[1]`, `[2]`, …; `[PREVIOUS_SUMMARY]` is S_{i-1} ("(none)" for $i = 1$); `[CURRENT_SEGMENT]` is $\widetilde{P}_i$ (P_1 for $i = 1$); and `[NEXT_SEGMENT]` is the original next segment P_{i+1}. A bullet is counted as conditionally inconsistent if it is labeled supported in Task 1 and not visible in Task 2.

```
You will check the quality of a progress summary written for a partial
solution to a math problem.

Background: a solver worked on the problem and produced a partial
solution (Source A). A summary of this partial solution was then
written, so that the solver can later continue from the problem and
the summary alone. The bullets of the summary are numbered.

<problem>
[PROBLEM]
</problem>

<source_A_partial_solution>
[COT_PREFIX]
</source_A_partial_solution>

<summary>
[SUMMARY]
</summary>

When the summary is written in deployment, the solver can see only the
following material (Source B): the problem above, the previous
summary, which condenses the solution before its most recent part, and
the most recent part of the solution.

<source_B_previous_summary>
[PREVIOUS_SUMMARY]
</source_B_previous_summary>

<source_B_recent_part>
[CURRENT_SEGMENT]
</source_B_recent_part>

Finally, this is how the solution actually continues after the partial
solution:

<next_part>
[NEXT_SEGMENT]
</next_part>

Complete the three tasks below. Judge faithfulness to the given texts
only: a bullet that correctly reports a mistaken step of the solution
is still supported. Do not use your own solution of the problem.

Task 1 (support). For each bullet, decide whether its content is
supported by the problem and Source A:
- "supported": every claim, value, equation, and conclusion in the
bullet is stated in the problem or Source A, or follows from them by a
trivial restatement (rewording, change of notation, or simple
arithmetic on stated values). A bullet that reports a goal, open
question, or plan is supported if Source A states that goal, question,
or plan.
- "contradicted": the bullet conflicts with Source A, for example by
giving a different value or conclusion, by reporting as established a
result that Source A rejected or corrected, or by reporting a
tentative guess as a verified result.
- "unsupported": the bullet does not conflict with Source A, but some
of its content is neither stated in nor trivially derivable from the
problem and Source A, for example a value, step, or answer that Source
A never reached.

Task 2 (visibility). For each bullet, decide whether its content is
available in Source B, i.e., stated in the problem, the previous
```

````
summary, or the recent part, or trivially derivable from them. Answer
"yes" or "no" for every bullet, whatever its label in Task 1.

Task 3 (missing information). Read the next part and list the
constraints and intermediate results that it relies on, that were
already established in Source A, and that the summary does not
contain. Do not list information stated in the problem, results that
the next part derives itself, general mathematical facts, or details
that the next part mentions without relying on them. Return an empty
list if nothing is missing.

Output only a JSON object wrapped in a ```json code fence, with one
entry per bullet in order:
```json
{
  "bullets": [
    {"id": 1, "support": "supported", "visible": "yes", "note": ""}
  ],
  "missing": [
    {"item": "short description of the omitted result",
     "evidence": "short quote from the next part that relies on it"}
  ]
}
```
Leave "note" empty for a bullet that is supported and visible;
otherwise give a one-sentence reason.
````

Rewrite judge `[HISTORY]` is S_{i-1}, `[ORIGINAL_HEAD]` is H_i, `[REWRITTEN_HEAD]` is $\widetilde{H}_i$ without the `<reformulated>` tags, and `[TAIL_START]` is the first $\sim$200 tokens of the tail T_i.

```
You will check a rewritten passage from a step-by-step solution to a
math problem.

Background: a solution was split into an earlier part and a later
part. The earlier part was condensed into a summary (the history). The
opening passage of the later part (the original head) was then
rewritten so that the later part can directly follow the problem and
the history. The rewritten head is supposed to (1) begin with a brief
recap of the problem and the history, and (2) then reproduce the
original head with only the minimal wording changes needed for a
smooth transition, without adding any reasoning. The text that follows
the head (the tail) is unchanged; its beginning is shown for context.

<problem>
[PROBLEM]
</problem>

<history>
[HISTORY]
</history>

<original_head>
[ORIGINAL_HEAD]
</original_head>

<rewritten_head>
[REWRITTEN_HEAD]
</rewritten_head>

<tail_start>
[TAIL_START]
</tail_start>
```

```
First identify the recap, i.e., the opening portion of the rewritten
head that restates the problem and the history, and call the rest of
the rewritten head the body. Then answer the four questions below with
"yes" or "no". Judge faithfulness to the given texts only; do not
judge whether the mathematics is correct.

Q1 (meaning_changed). Does the body change the mathematical meaning of
the original head? Answer "yes" if any mathematical statement, value,
expression, condition, case, or conclusion of the original head is
altered or omitted, if steps are reordered in a way that changes the
logic, or if the stance of the reasoning changes (for example, a doubt
or tentative guess becomes a confident claim, or vice versa).
Rewording, added connectives, and changes of notation that preserve
the meaning do not count.

Q2 (new_claim). Does the body introduce a claim, step, or conclusion
that is not in the original head? Answer "yes" if, for example, it
adds a derivation, a result, or a verification that the original head
does not contain. The recap is not considered here; it is assessed in
Q3.

Q3 (recap_supported). Is every statement in the recap supported by the
problem and the history? Answer "no" if the recap misstates them or
contains a value, result, or claim that is neither stated in nor
trivially derivable from them. If the rewritten head has no recap,
answer "no".

Q4 (natural). Read the rewritten head followed by the beginning of the
tail. Does the text read as one coherent passage, with a natural
transition from the recap into the body and from the body into the
tail? Answer "no" if there is an abrupt jump, a reference to something
that is not present in the problem, the history, or the preceding text
(e.g., "this equation" when no equation has been given), or noticeable
repetition.

Output only a JSON object wrapped in a ```json code fence, replacing
the example values with your answers:
```json
{"meaning_changed": "no", "new_claim": "no", "recap_supported": "yes",
"natural": "yes", "note": ""}
```
In "note", give a one-sentence reason for each answer that indicates a
problem; otherwise leave it empty.
```