STOP WHEN REASONING CONVERGES: SEMANTIC-PRESERVING EARLY EXIT FOR REASONING MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Reasoning Models (LRMs) achieve strong performance by generating long chains of thought (CoT), but often overthink, continuing to reason after a solution has already stabilized and thereby wasting tokens and increasing latency. Existing inference-time early-exit methods rely primarily on answer-level signals, such as confidence or trial-answer consistency, to decide when to stop. However, these signals mainly reflect answer readiness rather than reasoning convergence: they may trigger before the model has finished exploring or self-correcting, causing premature exits that can degrade final-answer accuracy and leave the retained reasoning chain semantically incomplete. We identify reasoning-level semantic redundancy as a complementary signal for semantic-preserving early exit: when successive steps no longer add novel progress and instead revisit established conclusions, the reasoning trajectory has likely converged. Building on this insight, we propose PUMA, a plug-and-play framework that combines a lightweight Redundancy Detector with answer-level verification. The detector flags semantically redundant candidate exits, while verification confirms whether stopping is safe, allowing PUMA to remove redundant continuation while preserving both answer accuracy and a coherent reasoning prefix. Across five LRMs and five challenging reasoning benchmarks, PUMA achieves 20.6% average token reduction while preserving accuracy and retained CoT quality. Additional experiments on code generation, zero-shot vision-language reasoning, and learned stopping-policy internalization further demonstrate that reasoning-level redundancy is a robust, transferable, and learnable signal for efficient reasoning.

1 INTRODUCTION

Recent Large Reasoning Models (LRMs) such as DeepSeek-R1 (Guo et al., 2025) and OpenAI o1 (Jaech et al., 2024) achieve strong performance by generating long chains of thought (CoT) (Wei et al., 2022) and scaling test-time computation (Muennighoff et al., 2025; Yang et al., 2026b; 2025; Snell et al., 2025). Beyond improving final-answer accuracy, these reasoning chains are often surfaced to users as explanations for final answers (Korbak et al., 2025; Gao et al., 2026; Li et al., 2025b) or agent actions (Zhou et al., 2024; Yao et al., 2023; Hsu et al., 2025; Cuadron et al., 2025), serving as an important basis for interpretability and user trust. However, long CoTs also introduce substantial inefficiency: models often continue reasoning after a solution has stabilized, repeatedly re-verifying or rephrasing established conclusions (Chen et al., 2025; Sun et al., 2026). To quantify this redundancy, we locate, for five representative LRMs on five benchmarks, the earliest reasoning step at which a trial answer already matches the final answer, and find that 41–52% of reasoning tokens are generated after this point (Figure 5 in Appendix A). This creates a clear opportunity for more efficient reasoning. However, removing these tokens requires care, because a CoT serves two purposes: the model relies on it to reason its way to the answer, and users often read it as the explanation of that answer. Naive truncation, which stops at a fixed length or interval without checking whether the retained prefix forms a self-contained derivation, can therefore harm both. In our fine-tuning experiments (Section 6.2), training a model on fixed-interval exits instead of PUMA-selected exits lowers accuracy by 3.8 to 12.8 points under the same training recipes. An effective method should remove unnecessary tokens,

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.

namely those generated after the final answer is first reached, while preserving final-answer accuracy and a coherent, semantically complete retained reasoning chain.

A growing body of work has sought to improve reasoning efficiency in LRMs. Training-based methods provide direct control over reasoning length (Zhang et al., 2025b; Arora & Zanette, 2025; Luo et al., 2025; Kang et al., 2025; Ma et al., 2025b; Team et al., 2025c), but typically require per-model retraining. Prompt-based compression methods are lightweight and easy to apply (Xu et al., 2025; Lin et al., 2026; Nayab et al., 2024; Han et al., 2025), but may hurt accuracy when brevity instructions suppress necessary intermediate reasoning. In contrast, inference-time early-exit methods (Rustagi et al., 2025; Huang et al., 2025; Mao et al., 2025; Wang et al., 2025b; Yang et al., 2026a; Fu et al., 2025) are attractive because they can reduce unnecessary reasoning at deployment time without modifying model weights. However, many existing early-exit methods primarily rely on answer-level signals, such as confidence estimates (Yang et al., 2026a; Huang et al., 2026; Rustagi et al., 2025) or trial-answer consistency (Liu & Wang, 2025; Fu et al., 2025; Mao et al., 2025). While these signals are useful for judging whether the current answer appears stable, they do not directly measure whether the reasoning process has converged. For example, a trial answer can reach confidence 0.982 with the wrong sign before the derivation is finished (Appendix G). Confidence and trial-answer consistency can satisfy stopping criteria while the model is still exploring or self-correcting: applied to five LRMs on five benchmarks, confidence-based and consistency-based exits stop at an incorrect answer in 44% and 64% of cases, and 42% and 55% of these failures occur on trajectories that would have reached the correct answer if allowed to continue (Appendix A.2).

This gap motivates a complementary stopping signal that tracks whether the reasoning trajectory is still making semantically novel progress. We draw inspiration from semantic entropy (Kuhn et al., 2023; Farquhar et al., 2024), which estimates uncertainty in LLM outputs by measuring whether multiple generated responses are semantically diverse or collapse to the same meaning, rather than comparing surface forms. Unlike the confidence estimates above, which score the token probabilities of a single trial answer, semantic entropy measures how outputs spread across distinct meanings, so it reflects whether generation still produces new content rather than how certain the current answer appears. We transfer this idea from *across-output* diversity to *within-trajectory* progress: if recent reasoning steps become semantically similar to prior steps and no longer introduce new logical or semantic content, the reasoning process is likely shifting from exploration to convergence (Appendix B.5 summarizes the supporting evidence). Under this view, once recent steps become semantically redundant, continued generation is likely to repeat established reasoning rather than add meaningful progress. We therefore use reasoning-level semantic redundancy as a complementary signal for where to consider stopping.

Building on this signal, we propose **PUMA**, a ***P**rogress-aware **U**nified **M**onitoring framework for **A**daptive early exit* in reasoning models. PUMA reduces redundant reasoning tokens while preserving both final-answer accuracy and the semantic quality of the retained CoT. It pairs a lightweight Redundancy Detector with answer-level verification: the detector monitors the reasoning trajectory and flags candidate exit points when the current step appears semantically redundant with recent context, while verification checks whether the trial answer is stable and sufficiently confident before stopping. This design decouples *where* to consider stopping from *whether* the candidate exit is reliable, allowing PUMA to avoid relying on answer-level readiness alone. To instantiate the detector, we fine-tune Qwen3-Embedding-0.6B (Zhang et al., 2025c) with a contrastive objective (Oord et al., 2018), training it to distinguish steps that introduce new logical or semantic progress from those that merely restate, re-derive, or loop over prior content. The detector is trained once and used unchanged for every evaluated LRM, so PUMA requires no LRM weight modification and no per-target-model training. This design also makes PUMA naturally semantic-preserving: because it favors stopping after meaningful exploration, the retained chain is more likely to form a coherent problem-solving narrative rather than cut mid-exploration.

Across five LRMs from diverse model families, including DeepSeek-R1-Distill (Guo et al., 2025), Llama-Nemotron (Bercovich et al., 2025), and Qwen3 (Yang et al., 2025), and five challenging reasoning benchmarks spanning MATH-500 (Hendrycks et al., 2021), AIME24/25 (Zhang & Math-AI, 2024; 2025), OlympiadBench (He et al., 2024), and GPQA-Diamond (Rein et al., 2024), PUMA achieves 20.6% average token reduction while preserving final-answer accuracy, with token savings translating into practical wall-clock speedups. Importantly, the reasoning chains that PUMA retains receive the highest quality scores among all compared methods, above even the full CoT. Additional experiments on code generation, zero-shot vision-language reasoning, and models trained to internal-

ize PUMA-selected exit positions further show that reasoning-level semantic redundancy is a robust, transferable, and learnable signal for efficient reasoning.

2 Related Work

Overthinking and efficient reasoning in LRMs. Overthinking refers to the phenomenon where extended CoT reasoning, despite improving performance, often generates more tokens than necessary (Chen et al., 2025; Sui et al., 2025; Su et al., 2025; Ghosal et al., 2026; Wu et al., 2025). Wei et al. (2026) characterize this behavior as a transition from active reasoning to a converged phase in which later tokens are largely redundant. Without external feedback, LLMs also struggle to correct their own reasoning (Huang et al., 2024; Kamoi et al., 2024), so post-answer re-verification is often redundant and can even overturn a correct answer. Existing efficient reasoning methods intervene at different stages. **Training-based methods** reshape reasoning through length-penalized RL (Aggarwal & Welleck, 2025; Hou et al., 2026; Dai et al., 2026), compressed-chain distillation (Zhang et al., 2025b; Li et al., 2026), or latent-space reasoning (Hao et al., 2025; He et al., 2025), but require per-model retraining (Ye et al., 2025; Yu et al., 2026; Brantley et al., 2026). **Prompt-based methods** encourage concise reasoning through length budgets or complexity-aware allocation (Xu et al., 2025; Renze & Guven, 2024; Lin et al., 2026), but such constraints can be ignored on difficult problems or suppress necessary intermediate reasoning (Yu et al., 2025). PUMA instead focuses on inference-time early exit, reducing redundant continuation without modifying the LRM's weights or prompts.

Inference-time early exit. Inference-time early-exit methods differ mainly in the signal used to decide when to stop. *Answer-level signals* monitor trial-answer confidence (Yang et al., 2026a; Huang et al., 2026) or agreement across consecutive probes (Liu & Wang, 2025; Fu et al., 2025; Mao et al., 2025). These signals estimate answer readiness but are blind to whether the reasoning trajectory is still making semantic progress (Tian et al., 2023; Chhikara, 2025; Kalai & Vempala, 2024). *Token-level signals* track decoding artifacts such as the rank of the `</think>` token, exit-associated neurons, or reflection-trigger words (Wei et al., 2026; Liu et al., 2026; Wang et al., 2025a), while *representation-level signals* train hidden-state probes to predict answer correctness (Akgül et al., 2025; Zhang et al., 2025a). These approaches are often tied to model-specific delimiters, vocabularies, hidden states, or calibration procedures. Closest to ours, REFRAIN (Sun et al., 2026) stops on reflection-trigger words plus high off-the-shelf embedding similarity to any previous step; PUMA instead trains a redundancy detector, compares each step with its predecessor, and verifies the trial answer before stopping (Appendices B.3–B.4).

3 Methodology

Figure 1 illustrates PUMA. PUMA is a plug-and-play early-exit framework that operates during decoding without modifying the LRM's weights or the base prompt. It consists of a primary *Verified Early Exit* path and a *Loop Breaker* fallback. Verified Early Exit follows a two-stage design: the Redundancy Detector monitors the reasoning trajectory and flags candidate exit points when the current step appears semantically redundant with recent context; Answer Verification is invoked only at these detector-flagged candidates to check whether the trial answer is stable and sufficiently confident. This separates *where* to consider stopping from *whether* the candidate exit is reliable. The Loop Breaker is activated only when no verified exit occurs and the model produces consecutive redundant steps in the later stages of reasoning.

Preliminaries. Given an input question, a reasoning model generates a reasoning chain $R = (r_1, r_2, \ldots, r_T)$ followed by a final answer A, where each r_t denotes a segmented reasoning step. Step segmentation is performed at natural paragraph or step boundaries, with details in Appendix B.1; Appendix B.8 evaluates PUMA under alternative segmenters.

Redundancy Detector. PUMA implements the Redundancy Detector as a lightweight embedding model trained for reasoning-step redundancy detection. We initialize it from Qwen3-Embedding-0.6B (Zhang et al., 2025c) and fine-tune it with LoRA (Hu et al., 2022) using an InfoNCE contrastive objective (Oord et al., 2018; Radford et al., 2021) to distinguish steps that introduce new logical or semantic progress from those that restate, re-derive, or loop over prior content. This task-specific training is important because generic semantic similarity does not directly capture reasoning-step novelty: two steps may be topically similar while still advancing the solution, or lexically different

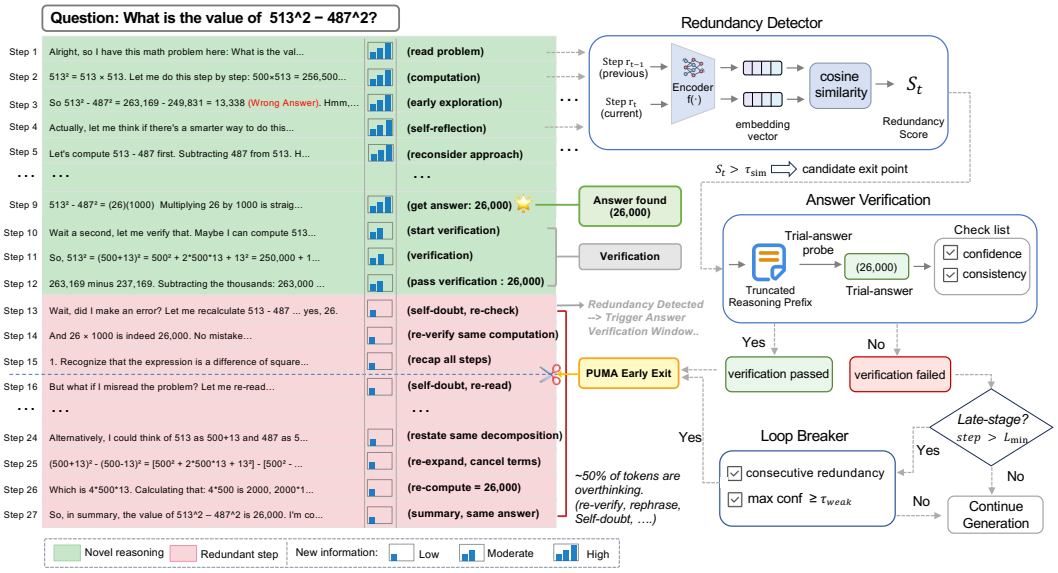


Figure 1: Overview of PUMA. The Redundancy Detector flags candidate exits, and Answer Verification checks the confidence and consistency of trial answers before stopping; the Loop Breaker is a fallback for sustained late-stage redundancy. Left: a DS-7B trace in which PUMA removes the redundant continuation.

while repeating the same verification. Details on supervision construction, labeling prompts, and training hyperparameters are provided in Appendix B.2.

Given the trained detector $f(\cdot)$, PUMA scores the redundancy of step r_t by comparing it with the previous k reasoning steps:

$$s_t^{(k)} = \max_{\max(1,\, t-k) \le j < t} \cos\big(f(r_j), f(r_t)\big). \tag{1}$$

A higher score indicates that the current step is semantically close to recent context and is less likely to add novel reasoning progress. When $s_t^{(k)} \ge \tau_{\text{sim}}$, PUMA flags r_t as redundant and marks the end of the preceding step as a candidate exit, so that a retained prefix never ends with a redundant step; we write t for the last step of a candidate prefix. By default, we use $k = 1$, comparing each step only with its immediate predecessor, which provides a conservative local redundancy signal. Sensitivity to larger lookback windows is analyzed in Appendix B.4; comparisons between our embedding-based detector and Natural Language Inference (NLI)-based (MacCartney, 2009) redundancy signals are provided in Appendix B.3. Conceptually, this detector serves as a lightweight online proxy for the local semantic collapse that semantic entropy would capture through explicit clustering; we discuss this perspective in Appendix B.5.

Answer Verification. A detector flag is not itself a stopping decision. Once a candidate point t is flagged, PUMA appends a task-specific answer-inducing suffix to the current reasoning prefix $r_{\le t}$ and probes the model to produce a trial answer A_t. The associated confidence C_t is a logit-based score computed only over the answer span, namely the geometric mean of token probabilities (Yang et al., 2026a) over the generated answer tokens:

$$C_t = \exp\left(\frac{1}{n}\sum_{i=1}^{n} \log p(a_i^t \mid a_{<i}^t, \mathbf{c}_t)\right), \tag{2}$$

where $\mathbf{c}_t$ denotes the prefix plus the answer-inducing suffix. Given the first detector-flagged candidate t_1, PUMA continues generation until it observes $L-1$ additional detector-flagged candidates, forming a verification window $t_1 < t_2 < \cdots < t_L$. At each candidate t_ℓ, PUMA extracts a trial answer A_{t_ℓ} and confidence C_{t_ℓ}. PUMA exits at the end of the window only if

$$\text{Exit}(t_1, \ldots, t_L) = [C_{t_1} > \lambda] \wedge \left[\bigwedge_{\ell=2}^{L} A_{t_\ell} = A_{t_1}\right] \wedge \left[\bigwedge_{\ell=2}^{L} C_{t_\ell} \ge C_{t_1} - \epsilon\right], \tag{3}$$

where λ is the confidence threshold and ϵ is the stability tolerance. These checks require the candidate answer to be confident, consistent across redundancy-triggered probes, and not materially declining in confidence. If any condition fails, PUMA resumes generation until the next flagged candidate.

Loop Breaker. Verified Early Exit is the primary stopping mechanism, but some trajectories produce consecutive redundant steps without satisfying all verification conditions. PUMA therefore includes a late-stage Loop Breaker fallback. After the reasoning chain exceeds a minimum step threshold, if the Redundancy Detector identifies m consecutive redundant steps, PUMA checks whether the highest-confidence trial answer observed so far exceeds a weak minimum-confidence gate. If the gate is satisfied, PUMA terminates generation; otherwise, generation continues. Verified exits take precedence, so the Loop Breaker is used only when no verified exit occurs. Sensitivity to m is analyzed in Appendix B.7.

4 Experimental Setup

Benchmarks and models. We evaluate PUMA on five challenging reasoning benchmarks covering competition mathematics, olympiad-level STEM reasoning, and graduate-level scientific reasoning: MATH-500 (Hendrycks et al., 2021), AIME24/25 (Zhang & Math-AI, 2024; 2025), OlympiadBench (He et al., 2024), and GPQA-Diamond (Rein et al., 2024). Our main experiments use five LRMs from diverse model families and scales: DeepSeek-R1-Distill-Qwen-7B/14B/32B (DS-7B/14B/32B; Guo et al., 2025), Llama-3.1-Nemotron-Nano-8B (Nemotron-8B; Bercovich et al., 2025), and Qwen3-30B-A3B-Thinking (Qwen3-30B-T; Yang et al., 2025). This suite covers 7B–32B-scale models, dense and mixture-of-experts architectures, and both mathematical and scientific reasoning tasks. Dataset statistics are provided in Appendix C.1.

Baselines. We focus our main comparisons on deployment-compatible efficiency methods that do not modify model weights or require training model-specific hidden-state probes, and report **Full-CoT** as the unmodified generation reference. *Prompt-based:* No-Think (Ma et al., 2025a) prompts the model to skip reasoning entirely; Concise CoT (CCoT; Nayab et al., 2024) imposes a global word budget; Chain of Draft (CoD; Xu et al., 2025) enforces per-step word limits; Plan-and-Budget (Lin et al., 2026) decomposes problems and allocates reasoning budgets by complexity. *Inference-time early exit:* Answer Convergence (Ans. Conv.; Liu & Wang, 2025) stops when consecutive trial answers agree; Dynasor (Fu et al., 2025) combines answer consistency with certainty probing; DEER (Yang et al., 2026a) halts when a trial answer exceeds a confidence threshold. All baselines are reproduced using their official code repositories where available, or following the procedures described in the original papers. Token-level methods need their thresholds recalibrated for each model, so we compare with the closest one, RCPD (Wei et al., 2026), in an additional experiment on DS-7B (Appendix D.2).

Metrics. For the main experiments, we report accuracy (*Acc*), average generated tokens (*Tok*), and token reduction $TR = (1 - Tok_{\text{method}}/Tok_{\text{Full-CoT}}) \times 100$. Higher *TR* indicates greater efficiency. We report accuracy and token reduction jointly, since token savings are meaningful only when answer quality is preserved. For latency experiments, we additionally report wall-clock speedup relative to Full-CoT. Unless otherwise stated, token counts include all generated tokens incurred by a method, including main reasoning tokens, trial-answer tokens, and final-answer tokens, so TR reflects total generation cost rather than only the retained output length; Appendix C.6 breaks PUMA's tokens down by source.

Implementation Details. All experiments are conducted on a single node with 4×NVIDIA GH200 Grace Hopper superchips. We use vLLM for language-model inference and serve the Redundancy Detector on the same node. Unless otherwise specified, we use the models' recommended reasoning settings (`temperature`=0.6, `top_p`=0.95) and report results averaged over three random seeds (0, 42, 123). We also report the standard deviation of PUMA's overall accuracy and token reduction across seeds. PUMA uses conservative stopping hyperparameters to favor high-precision exits: for the Redundancy Detector, local window size $k = 1$ and similarity threshold $\tau_{\text{sim}} = 0.35$, selected on a held-out detector calibration split (Appendix B.2); for Answer Verification, confidence threshold $\lambda = 0.98$, stability tolerance $\epsilon = 0.03$, and verification window length $L = 2$; and for the Loop Breaker, a late-stage activation threshold of 50 reasoning steps, a weak minimum-confidence gate of 0.8, and a consecutive-redundancy threshold m set per model on a held-out AMC23 set. Sensitivity to these choices is analyzed in Appendix B.7.

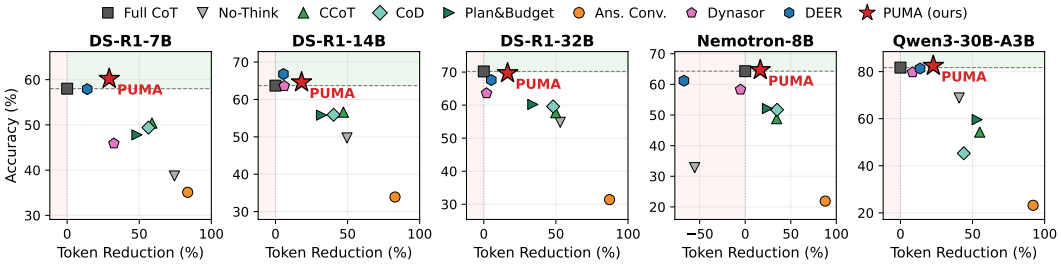


Figure 2: Overall accuracy versus token reduction on five LRMs, averaged over the five benchmarks. Dashed line: Full CoT accuracy. Green: fewer tokens without lower accuracy than Full CoT; red: more tokens than Full CoT.

5 Experimental Results

5.1 Main Results: Accuracy, Efficiency, and Reasoning Quality

Figure 2 places every method in the accuracy–efficiency plane for each of the five LRMs; Table 1 reports per-benchmark results on DS-7B, and Appendix C on all five LRMs. PUMA achieves the best trade-off on every model. It is the only method that reduces tokens without lowering accuracy on DS-7B, Nemotron-8B, and Qwen3-30B-T. On DS-14B, the only other such method, DEER, removes less than a third as many tokens (5.5% vs. 18.1%), and on DS-32B, where no method matches Full CoT accuracy, PUMA comes closest (0.5 points below) while removing 16.5% of tokens. Among all methods that stay within one point of Full CoT accuracy, PUMA removes the most tokens on every model. Averaged over the five LRMs, PUMA reduces tokens by **20.6**% while slightly improving accuracy (68.3% vs. 67.6%). These gains are consistent with the overthinking phenomenon: after reaching a correct answer, LRMs may continue re-verifying or revising and occasionally drift to an incorrect conclusion, which PUMA avoids by stopping once reasoning has converged (Appendix G). On the two largest benchmarks, MATH-500 and OlympiadBench, where accuracy estimates are most reliable, PUMA removes 25.3% and 32.0% of tokens while staying within 0.4 and 1.4 points of Full CoT. Statistical tests further confirm that PUMA's accuracy is not significantly lower than Full CoT on any of the five LRMs (Appendix C.3).

Compared with prompt-based compression, PUMA is more accuracy-preserving. Although CCoT, CoD, and Plan-and-Budget often reduce token usage, their brevity constraints can suppress necessary intermediate reasoning, especially on stronger reasoning models. For example, on Qwen3-30B-A3B-Thinking, their accuracies drop to 54.3%, 45.3%, and 59.6%, respectively, compared with 81.7% for Full CoT. Compared with answer-level early-exit baselines, PUMA provides a more reliable accuracy–efficiency tradeoff: Answer Convergence stops aggressively but suffers severe accuracy collapse, while Dynasor and DEER show inconsistent efficiency due to repeated trial-answer probing. In contrast, PUMA invokes answer verification only after reasoning-level redundancy is detected. Individual benchmarks can still favor a baseline: on DS-7B, DEER is more accurate than PUMA on four of the five benchmarks, by 1.0 to 3.3 points (Table 1), but it removes far fewer tokens (13.9% vs. 29.2% overall) and uses more tokens than Full CoT on GPQA-Diamond. Across the

Table 1: Per-benchmark results on DS-7B. TR: token reduction relative to Full CoT (negative values in red). Overall: mean over the five benchmarks, with the accuracy change relative to Full CoT in parentheses. Table 20 reports all five models.

	MATH-500		AIME24		AIME25		GPQA-D		OlymBench		Overall	
Method	Acc↑	TR↑	Acc↑	TR↑	Acc↑	TR↑	Acc↑	TR↑	Acc↑	TR↑	Acc↑	TR↑
Full CoT	90.0	0.0	50.0	0.0	43.3	0.0	49.0	0.0	57.6	0.0	58.0	0.0
No-Think	79.0	80.0	20.0	67.7	23.3	57.9	28.3	87.3	42.8	79.1	38.7 (-19.3)	74.4
CCoT	85.0	60.9	36.7	59.6	33.3	60.2	48.0	55.6	49.2	57.7	50.4 (-7.6)	58.8
CoD	80.4	59.9	40.0	56.9	33.3	58.7	50.5	50.8	43.0	55.8	49.4 (-8.6)	56.4
Plan&Budget	84.6	48.5	43.3	52.9	23.3	53.9	38.4	36.2	49.3	50.3	47.8 (-10.2)	48.4
Ans. Conv.	61.8	76.5	26.7	83.3	20.0	82.8	34.3	92.4	32.6	82.8	35.1 (-22.9)	83.6
Dynasor	84.2	30.6	23.3	21.7	33.3	9.1	39.4	66.6	49.0	34.1	45.9 (-12.1)	32.4
DEER	90.6	39.6	40.0	14.1	50.0	13.2	51.5	-17.4	57.5	20.2	57.9 (-0.1)	13.9
PUMA (ours)	89.6	38.6	60.0	30.3	46.7	29.8	49.0	4.0	55.6	43.2	60.2 (+2.2)	29.2

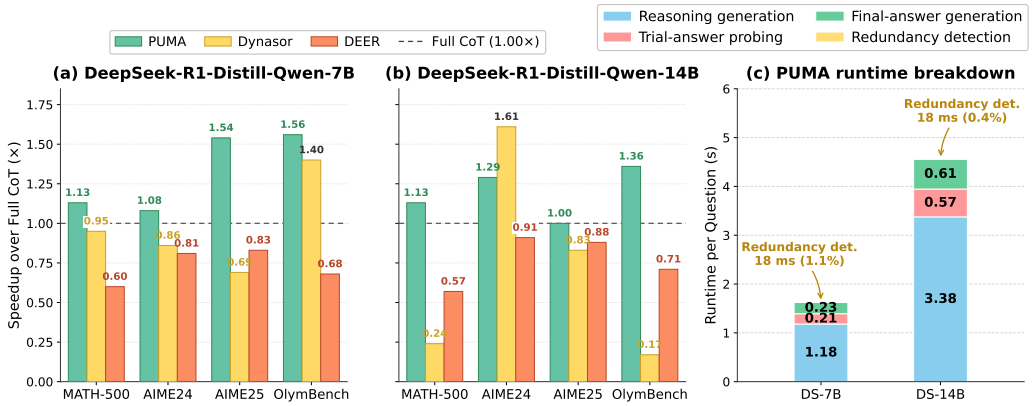


Figure 3: Wall-clock latency on DS-7B and DS-14B. (a,b) Per-benchmark speedup relative to Full-CoT. (c) PUMA runtime breakdown: the Redundancy Detector adds only 0.4–1.1% overhead. Detailed per-benchmark results, including additional models, are reported in Appendix C.5.

five LRMs, PUMA removes more tokens than DEER on every model and is more accurate on four of them (Figure 2), and DEER is slower than Full CoT in every latency experiment (Section 5.2). PUMA's advantage is largest in cross-domain generalization: on code generation and vision-language reasoning (Section 5.3), DEER loses 1.5 to 14.5 accuracy points, whereas PUMA stays within 1.5 points of Full CoT.

Beyond final-answer accuracy, early exit should preserve the retained reasoning chain as a useful explanation. Following the LLM-as-Judge protocol (Li et al., 2025a), Table 2 evaluates retained CoT quality along completeness, coherence, conciseness, and justification. PUMA achieves the highest average retained-chain quality among compared methods, ranking first in coherence, conciseness, and justification while remaining close to Full CoT in completeness. This supports the semantic-preserving nature of PUMA: it shortens reasoning by removing redundant continuation rather than aggressively cutting the chain mid-development. Details of the evaluation setup and judge prompt are provided in Appendix E.

5.2 From Token Savings to Latency Gains

Token reduction does not automatically translate to wall-clock speedup, because early-exit methods incur overhead from trial-answer probing. Figure 3 shows that PUMA turns token savings into practical speedups, achieving 1.40× speedup on DS-7B and 1.28× on DS-14B on average. By contrast, DEER is slower than Full CoT on both models, while Dynasor is faster on DS-7B but substantially slower on DS-14B, highlighting the overhead of frequent answer-level probing. The runtime breakdown in Figure 3(c) further shows that PUMA's overhead is small: the Redundancy Detector contributes only 0.4–1.1% of total wall-clock time per question, and the Answer Verification overhead from trial-answer probing is only 0.2–0.57 seconds per question.

5.3 Generalization Beyond Text-only QA

We further test whether reasoning-level redundancy transfers beyond text-only mathematical and scientific QA. Table 3 evaluates PUMA on code generation with LiveCodeBench (Jain et al., 2025) and vision-language reasoning with MathVista (Lu et al., 2024) and MathVision (Wang et al., 2024).

Table 2: Reasoning-chain quality evaluated by an LLM-as-Judge (GPT-5.4-thinking). Individual chains are rated on a 10–100 rubric in 10-point increments, and the higher is better.

Metric	Full-CoT	PUMA	CCoT	Plan&Budget	CoD	Dynasor	DEER
Completeness↑	**67.6**	64.8	64.1	62.5	62.0	57.2	62.4
Coherence↑	41.5	**57.5**	51.6	48.6	49.0	46.9	37.9
Conciseness↑	15.7	**40.2**	30.5	27.2	26.1	25.0	19.1
Justification↑	51.8	**54.8**	54.5	51.6	50.3	46.5	46.7
Avg.↑	44.1	**54.3**	50.2	47.5	46.8	43.9	41.5

On LiveCodeBench, PUMA reduces tokens by 18–19% with at most 1.5 points of pass@1 change. On vision-language reasoning, PUMA is applied without retraining or hyperparameter tuning, yet still reduces tokens by 23.8–33.6% with at most 1.5 points of accuracy change. These results suggest that reasoning-level redundancy is a transferable stopping signal across code, text, and multimodal reasoning. In an agent setting on τ-bench (Yao et al., 2024), PUMA is the only early-exit method that shortens reasoning (by 27% and 17% on the retail and airline domains) without lowering task success (Appendix C.7). Additional prompt-budget sweeps for CCoT and CoD are provided in Appendix D.1.

Table 3: Generalization to code and vision-language reasoning. Code generation is evaluated on LiveCodeBench (880 problems, pass@1); for PUMA, only the redundancy threshold is adjusted to $\tau_{\text{sim}} = 0.50$. Vision-language reasoning is evaluated on MathVista and MathVision (200 problems each), where PUMA is applied zero-shot without retraining or hyperparameter tuning. **Bold** marks the highest accuracy among the efficiency methods.

	Code (LiveCodeBench)				VL: Qwen3-VL-8B-Thinking				VL: Kimi-VL-16B-A3B-Thinking			
	DS-7B		Nemotron-8B		MathVista		MathVision		MathVista		MathVision	
Method	Acc (Δ)	TR↑	Acc (Δ)	TR↑	Acc (Δ)	TR↑	Acc (Δ)	TR↑	Acc (Δ)	TR↑	Acc (Δ)	TR↑
Full CoT	51.8 (—)	0.0	60.2 (—)	0.0	76.0 (—)	0.0	40.0 (—)	0.0	67.5 (—)	0.0	24.0 (—)	0.0
PUMA (ours)	**50.3** (-1.5)	19.0	**59.7** (-0.5)	18.0	**75.0** (-1.0)	23.8	**38.5** (-1.5)	31.2	**68.0** (+0.5)	28.7	**24.0** (+0.0)	33.6
DEER	49.6 (-2.2)	1.6	52.6 (-7.6)	19.3	71.0 (-5.0)	16.1	36.5 (-3.5)	54.3	53.0 (-14.5)	11.5	22.5 (-1.5)	-1.3
CCoT	45.8 (-6.0)	42.7	49.3 (-10.9)	41.6	72.5 (-3.5)	11.2	36.0 (-4.0)	26.4	53.0 (-14.5)	71.2	21.5 (-2.5)	25.0
CoD	43.2 (-8.6)	40.5	48.9 (-11.3)	41.8	32.5 (-43.5)	60.9	24.5 (-15.5)	58.6	37.5 (-30.0)	38.3	19.5 (-4.5)	42.2
Plan&Budget	39.2 (-12.6)	31.8	50.6 (-9.6)	28.4	62.5 (-13.5)	1.0	30.0 (-10.0)	37.5	57.0 (-10.5)	24.3	22.5 (-1.5)	29.1
Dynasor	43.0 (-8.8)	-207.6	45.5 (-14.7)	-7.8	64.0 (-12.0)	61.8	36.0 (-4.0)	48.4	40.5 (-27.0)	48.3	19.5 (-4.5)	92.6
No-Think	29.1 (-22.7)	88.1	47.6 (-12.6)	13.1	56.0 (-20.0)	63.9	29.5 (-10.5)	73.1	61.5 (-6.0)	70.1	18.5 (-5.5)	68.3

6 Analysis and Discussion

6.1 Component and Exit Behavior Analysis

Table 4 ablates PUMA's candidate-triggering and verification components on DS-7B, with full two-model results and probe statistics provided in Appendix C.4. The full configuration achieves the best accuracy–efficiency balance, reaching 60.2% accuracy with 29.2% token reduction. Removing the Redundancy Detector gate sets the redundancy threshold to zero, causing Answer Verification to be invoked at every step rather than only at redundancy-triggered candidate points. This aggressive variant increases token reduction to 36.2%, but drops accuracy by 4.1 points and requires 3.3× more trial-answer probes per question. This shows that reasoning-level gating is important for both reliability and probe efficiency. Removing the Loop Breaker lowers token reduction from 29.2% to 17.5% but accuracy by only 1.0 point, confirming that it mainly serves as an efficiency-oriented fallback for sustained redundancy. PUMA without the Loop Breaker is therefore a simpler configuration that still exceeds Full CoT accuracy (59.2 vs. 58.0) while removing 17.5% of tokens. The Answer-Verification parts are also important for accuracy preservation: removing Answer Consistency lowers accuracy by 2.9 points, while removing the Confidence Gate causes a larger 6.5-point drop despite higher token reduction. Overall, the ablation shows that PUMA's components are complementary: the Redundancy Detector gate controls when verification is invoked, Answer Verification filters unreliable candidate exits, and the Loop Breaker provides additional savings when reasoning enters sustained redundancy. Combining them adds little cost, as the detector takes only 0.4–1.1% of wall-clock time (Figure 3c).

Table 4: Component ablation on DS-7B; Acc and TR are averaged over the five benchmarks. Probe× denotes trial-answer probes per question relative to PUMA.

Configuration	Acc↑	TR↑	Probe× ↓
PUMA (full)	**60.2**	29.2	1.0×
w/o Redundancy Detector Gate	56.1 (-4.1)	36.2	3.3×
w/o Loop Breaker	59.2 (-1.0)	17.5	1.3×
w/o Answer Consistency	57.3 (-2.9)	25.7	1.2×
w/o Confidence Gate	53.7 (-6.5)	49.7	0.6×

Figure 4 further analyzes when PUMA stops and how different exit modes affect correctness. Figure 4(a) shows that PUMA does not rely on a single exit mode: verified exits, Loop Breaker exits, and full reasoning occur in different proportions across models, reflecting model-specific redundancy patterns. Figures 4(c) and 4(d) further break down correctness transitions for verified exits and Loop Breaker exits, respectively, where $W \to R$, for example, denotes a trajectory that is wrong under Full CoT but correct after PUMA stopping. For verified exits, the $R \to R$ transition dominates (69–72%),

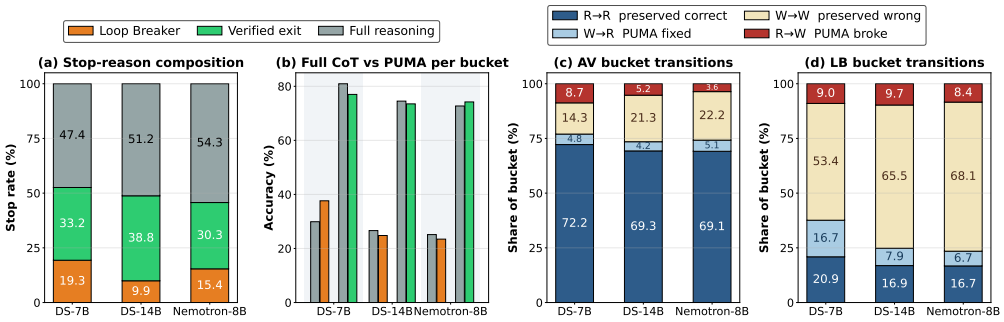


Figure 4: Exit behavior of PUMA across three representative LRMs. **(a)** Stop-reason composition: Loop Breaker, verified exit, or full reasoning. **(b)** Accuracy change of each exit bucket relative to Full CoT. **(c,d)** Correctness transitions for verified exits and Loop Breaker exits, where R/W denote correct/wrong outcomes for Full CoT $\rightarrow$ PUMA.

indicating that Answer Verification usually preserves already-correct trajectories. The $W \rightarrow R$ cases show that PUMA can also fix some Full-CoT errors by stopping before post-convergence drift, while harmful $R \rightarrow W$ transitions remain limited. Loop Breaker exits show a different pattern: $W \rightarrow W$ transitions dominate (53–68%), meaning that the fallback mostly cuts late redundant continuation from trajectories that Full CoT would also answer incorrectly. Together, these results explain how PUMA gains efficiency with limited accuracy risk, and why it can occasionally outperform Full CoT by avoiding late-stage drift. Appendix G presents two examples to illustrate this: in one, Answer Verification rejects a trial answer that is confident but has the wrong sign; in the other, stopping early keeps a correct answer that the full CoT later abandons.

6.2 From Inference-Time Signal to Learned Stopping Policy

Finally, we ask whether a model can learn to stop at PUMA-selected exits on its own, so that no PUMA module is needed at inference time. We run PUMA with DS-7B on 12K MATH training problems, which do not overlap with MATH-500, and use its exit positions as supervision in three ways. PUMA-SFT fine-tunes on reasoning chains truncated at PUMA's exits; PUMA-DPO (Rafailov et al., 2023) prefers these truncated chains over the full chains when both reach the correct answer; and PUMA-RL applies GRPO (Shao et al., 2024) to the answer generated after a PUMA-truncated prefix, rewarding answers that are correct and short (Appendix F). The trained models run with plain vLLM inference. As Table 29 shows, PUMA-RL exceeds inference-time PUMA in both average accuracy (67.0 vs. 66.2) and token reduction (34.9% vs. 24.3%), and PUMA-DPO reaches the highest token reduction (48.8%) while still improving on Full CoT accuracy (64.1 vs. 63.0). To test whether the exit positions themselves matter, the FixedExit baselines use the same recipes but truncate at fixed intervals; they lower average accuracy by 12.8 points for SFT and 3.8 points for RL. Training without exit supervision (Standard-SFT and Standard-GRPO) does not shorten the output on average. Notably, PUMA-RL learns to end its own reasoning earlier even though it is trained only to write the final answer after reasoning that PUMA has already cut (Appendix F). These results show that PUMA's reasoning-level redundancy signal is not only an effective stopping criterion at inference time but also a learnable policy: its exit positions teach models when to stop reasoning, with no PUMA module needed at deployment.

7 Conclusion

We presented PUMA, a plug-and-play early-exit framework that uses reasoning-level semantic redundancy to identify when a reasoning trajectory appears to have converged, and pairs this signal with answer-level verification for reliable early stopping. Across five LRMs and five challenging reasoning benchmarks, PUMA reduces tokens by 20.6% on average while preserving accuracy, delivering practical wall-clock speedups, and maintaining higher-quality retained reasoning chains than competing methods. PUMA further transfers to code generation and zero-shot vision-language reasoning, and its selected exit positions can supervise SFT, DPO, and GRPO, enabling models to internalize concise stopping behavior without PUMA modules at inference time.

AI USE STATEMENT

We used generative AI tools, including ChatGPT, Codex, and Claude Code, to polish writing, to help with code development and debugging, and to assist with data analysis. Separately, LLMs are part of our research methodology: GPT-5-mini annotates seed redundancy labels and GPT-4o-mini synthesizes redundant steps for training the Redundancy Detector (Appendix B.2), GPT-5.4-thinking judges the quality of retained reasoning chains (Appendix E), and DeepSeek-V4-Pro simulates the user in the τ-bench experiments (Appendix C.7). The authors reviewed and revised all AI-assisted code and text, validated the LLM judge against human preferences (Appendix E), and take responsibility for the research methods and reported results.

ETHICS STATEMENT

This work adheres to the ICLR Code of Ethics. All models and benchmarks we use are publicly available and are used under their licenses (Appendix C.2). The reasoning traces are generated by these models on public benchmark problems and contain no personal data, and the τ-bench experiments involve only simulated users. The only human involvement is one annotator who compared pairs of anonymized reasoning chains to validate the LLM judge (Appendix E); this task involves no personal or sensitive content, and the annotator's identity is not disclosed.

REPRODUCIBILITY STATEMENT

Section 3 and Appendix B describe PUMA, including step segmentation (Appendix B.1), Redundancy Detector training with its data construction, annotation prompts, and LoRA settings (Appendix B.2), and trial-answer induction (Appendix B.6). Section 4 specifies the models, benchmarks, sampling settings, random seeds, hardware, and stopping hyperparameters, and Appendix B.7 gives the per-model Loop Breaker thresholds. Appendix C.1 gives dataset statistics, Appendix C.3 reports per-benchmark results for all five LRMs with standard deviations of the overall results over three seeds, and Appendices D.2 and C.7 describe the RCPD reimplementation and the τ-bench setup. The judge protocol and prompt are in Appendix E, and the data construction, hyperparameters, and reward for the fine-tuning experiments are in Appendix F. An anonymized repository with the PUMA code and instructions to reproduce the main results is available at `https://anonymous.4open.science/r/puma-code-0428`, and we will release the trained Redundancy Detector upon publication.

REFERENCES

Pranjal Aggarwal and Sean Welleck. L1: Controlling how long a reasoning model thinks with reinforcement learning. In *Second Conference on Language Modeling*, 2025. URL `https://openreview.net/forum?id=4jdIxXBNve`.

Ömer Faruk Akgül, Yusuf Hakan Kalaycı, Rajgopal Kannan, Willie Neiswanger, and Viktor Prasanna. Lynx: Learning dynamic exits for confidence-controlled reasoning. *arXiv preprint arXiv:2512.05325*, 2025.

Daman Arora and Andrea Zanette. Training language models to reason efficiently. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL `https://openreview.net/forum?id=AiZxn84Wdo`.

Akhiad Bercovich, Itay Levy, Izik Golan, Mohammad Dabbah, Ran El-Yaniv, Omri Puny, Ido Galil, Zach Moshe, Tomer Ronen, Najeeb Nabwani, Ido Shahaf, Oren Tropp, Ehud Karpas, Ran Zilberstein, Jiaqi Zeng, Soumye Singhal, Alexander Bukharin, Yian Zhang, Tugrul Konuk, Gerald Shen, Ameya Sunil Mahabaleshwarkar, Bilal Kartal, Yoshi Suhara, Olivier Delalleau, Zijia Chen, Zhilin Wang, David Mosallanezhad, Adi Renduchintala, Haifeng Qian, Dima Rekesh, Fei Jia, Somshubra Majumdar, Vahid Noroozi, Wasi Uddin Ahmad, Sean Narenthiran, Aleksander Ficek, Mehrzad Samadi, Jocelyn Huang, Siddhartha Jain, Igor Gitman, Ivan Moshkov, Wei Du, Shubham Toshniwal, George Armstrong, Branislav Kisacanin, Matvei Novikov, Daria Gitman, Evelina Bakhturina, Jane Polak Scowcroft, John Kamalu, Dan Su, Kezhi Kong, Markus Kliegl, Rabeeh Karimi, Ying Lin, Sanjeev Satheesh, Jupinder Parmar, Pritam Gundecha, Brandon Norick, Joseph

Jennings, Shrimai Prabhumoye, Syeda Nahida Akter, Mostofa Patwary, Abhinav Khattar, Deepak Narayanan, Roger Waleffe, Jimmy Zhang, Bor-Yiing Su, Guyue Huang, Terry Kong, Parth Chadha, Sahil Jain, Christine Harvey, Elad Segal, Jining Huang, Sergey Kashirsky, Robert McQueen, Izzy Putterman, George Lam, Arun Venkatesan, Sherry Wu, Vinh Nguyen, Manoj Kilaru, Andrew Wang, Anna Warno, Abhilash Somasamudramath, Sandip Bhaskar, Maka Dong, Nave Assaf, Shahar Mor, Omer Ullman Argov, Scot Junkin, Oleksandr Romanenko, Pedro Larroy, Monika Katariya, Marco Rovinelli, Viji Balas, Nicholas Edelman, Anahita Bhiwandiwalla, Muthu Subramaniam, Smita Ithape, Karthik Ramamoorthy, Yuting Wu, Suguna Varshini Velury, Omri Almog, Joyjit Daw, Denys Fridman, Erick Galinkin, Michael Evans, Katherine Luna, Leon Derczynski, Nikki Pope, Eileen Long, Seth Schneider, Guillermo Siman, Tomasz Grzegorzek, Pablo Ribalta, Monika Katariya, Joey Conway, Trisha Saar, Ann Guan, Krzysztof Pawelec, Shyamala Prayaga, Oleksii Kuchaiev, Boris Ginsburg, Oluwatobi Olabiyi, Kari Briski, Jonathan Cohen, Bryan Catanzaro, Jonah Alben, Yonatan Geifman, Eric Chung, and Chris Alexiuk. Llama-nemotron: Efficient reasoning models, 2025. URL https://arxiv.org/abs/2505.00949.

Kianté Brantley, Mingyu Chen, Zhaolin Gao, Jason D. Lee, Wen Sun, Wenhao Zhan, and Xuezhou Zhang. Accelerating RL for LLM reasoning with optimal advantage regression. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026. URL https://openreview.net/forum?id=T1V8BJO0iG.

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. Do NOT think that much for 2+3=? on the overthinking of long reasoning models. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=MSbU3L7V00.

Prateek Chhikara. Mind the confidence gap: Overconfidence, calibration, and distractor effects in large language models. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=lyaHnHDdZl.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Alejandro Cuadron, Dacheng Li, Wenjie Ma, Xingyao Wang, Yichuan Wang, Siyuan Zhuang, Shu Liu, Luis Gaspar Schroeder, Tian Xia, Huanzhi Mao, Nicholas Thumiger, Aditya Desai, Ion Stoica, Ana Klimovic, Graham Neubig, and Joseph E. Gonzalez. The danger of overthinking: Examining the reasoning-action dilemma in agentic tasks, 2025. URL https://arxiv.org/abs/2502.08235.

Mz Dai, Chenxu Yang, and Qingyi Si. S-GRPO: Early exit via reinforcement learning in reasoning models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026. URL https://openreview.net/forum?id=wNMK5o0Vfg.

DeepSeek-AI, Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. DeepSeek-V4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

Sebastian Farquhar, Jannik Kossen, Lorenz Kuhn, and Yarin Gal. Detecting hallucinations in large language models using semantic entropy. *Nature*, 630(8017):625–630, 2024.

Yichao Fu, Junda Chen, Yonghao Zhuang, Zheyu Fu, Ion Stoica, and Hao Zhang. Reasoning without self-doubt: More efficient chain-of-thought through certainty probing. In *ICLR 2025 Workshop on Foundation Models in the Wild*, 2025. URL https://openreview.net/forum?id=wpK4IMJfdX.

Jiaxuan Gao, Shu Yan, Qixin Tan, lu Yang, Shusheng Xu, Wei Fu, Zhiyu Mei, Kaifeng Lyu, and Yi Wu. How far are we from optimal reasoning efficiency? In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026. URL https://openreview.net/forum?id=NhAi1w3s8Z.

Soumya Suvra Ghosal, Souradip Chakraborty, Avinash Reddy, Yifu Lu, Mengdi Wang, Dinesh Manocha, Furong Huang, Mohammad Ghavamzadeh, and Amrit Singh Bedi. Does thinking more always help? mirage of test-time scaling in reasoning models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026. URL https://openreview.net/forum?id=tKPqbamNb9.

Etash Kumar Guha, Ryan Marten, Sedrick Keh, Negin Raoof, Georgios Smyrnis, Hritik Bansal, Marianna Nezhurina, Jean Mercat, Trung Vu, Zayne Rea Sprague, Ashima Suvarna, Benjamin Feuer, Leon Liangyu Chen, Zaid Khan, Eric Frankel, Sachin Grover, Caroline Choi, Niklas Muennighoff, Shiye Su, Wanjia Zhao, John Yang, Shreyas Pimpalgaonkar, Kartik sharma, Charlie Cheng-Jie Ji, Yichuan Deng, Sarah M Pratt, Vivek Ramanujan, Jon Saad-Falcon, Stutee Acharya, Jeffrey Li, Achal Dave, Alon Albalak, Kushal Arora, Blake Wulfe, Chinmay Hegde, Greg Durrett, Sewoong Oh, Mohit Bansal, Saadia Gabriel, Aditya Grover, Kai-Wei Chang, Vaishaal Shankar, Aaron Gokaslan, Mike A Merrill, Tatsunori Hashimoto, Yejin Choi, Jenia Jitsev, Reinhard Heckel, Maheswaran Sathiamoorthy, Alex Dimakis, and Ludwig Schmidt. Openthoughts: Data recipes for reasoning models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=7xjoTuaNmN.

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.

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. Token-budget-aware LLM reasoning. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 24842–24855, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.1274. URL https://aclanthology.org/2025.findings-acl.1274/.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason E 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=Itxz7S4Ip3.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. OlympiadBench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3828–3850, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.211. URL https://aclanthology.org/2024.acl-long.211/.

Yinhan He, Wendy Zheng, Yaochen Zhu, Zaiyi Zheng, Lin Su, Sriram Vasudevan, Qi Guo, Liangjie Hong, and Jundong Li. Semcot: Accelerating chain-of-thought reasoning through semantically-aligned implicit tokens. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=1ZuzFUMtx6.

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 *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. URL https://openreview.net/forum?id=7Bywt2mQsCe.

Bairu Hou, Yang Zhang, Jiabao Ji, Yujian Liu, Kaizhi Qian, Jacob Andreas, and Shiyu Chang. Thinkprune: Pruning long chain-of-thought of LLMs via reinforcement learning. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=V51gPu1uQD.

Chan-Jan Hsu, Davide Buffelli, Jamie McGowan, Feng-Ting Liao, Yi-Chang Chen, Sattar Vakili, and Da shan Shiu. Group think: Multiple concurrent reasoning agents collaborating at token level granularity, 2025. URL https://arxiv.org/abs/2505.11107.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Chengsong Huang, Langlin Huang, Jixuan Leng, Jiacheng Liu, and Jiaxin Huang. Efficient test-time scaling via self-calibration. In *NeurIPS 2025 Workshop on Efficient Reasoning*, 2025. URL https://openreview.net/forum?id=RvMjxGpVOa.

Jiameng Huang, Baijiong Lin, Guhao Feng, Jierun Chen, Di He, and Lu Hou. Efficient reasoning for large reasoning language models via certainty-guided reflection suppression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 31176–31184, 2026.

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

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

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

Adam Tauman Kalai and Santosh S. Vempala. Calibrated language models must hallucinate. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, STOC 2024, pp. 160–171, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400703836. doi: 10.1145/3618260.3649777. URL https://doi.org/10.1145/3618260.3649777.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. When can LLMs actually correct their own mistakes? a critical survey of self-correction of LLMs. *Transactions of the Association for Computational Linguistics*, 12:1417–1440, 2024.

Yu Kang, Xianghui Sun, Liangyu Chen, and Wei Zou. C3ot: Generating shorter chain-of-thought without compromising effectiveness. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 24312–24320, 2025.

Tomek Korbak, Mikita Balesni, Elizabeth Barnes, Yoshua Bengio, Joe Benton, Joseph Bloom, Mark Chen, Alan Cooney, Allan Dafoe, Anca Dragan, et al. Chain of thought monitorability: A new and fragile opportunity for ai safety. *arXiv preprint arXiv:2507.11473*, 2025.

Lorenz Kuhn, Yarin Gal, and Sebastian Farquhar. Semantic uncertainty: Linguistic invariances for uncertainty estimation in natural language generation. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=VD-AYtP0dve.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, pp. 611–626, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400702297. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

Dawei Li, Bohan Jiang, Liangjie Huang, Alimohammad Beigi, Chengshuai Zhao, Zhen Tan, Amrita Bhattacharjee, Yuxuan Jiang, Canyu Chen, Tianhao Wu, Kai Shu, Lu Cheng, and Huan Liu. From generation to judgment: Opportunities and challenges of LLM-as-a-judge. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 2757–2791, Suzhou, China, November 2025a. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.138. URL https://aclanthology.org/2025.emnlp-main.138/.

Zeju Li, Jianyuan Zhong, Ziyang Zheng, Xiangyu Wen, Zhijian Xu, Yingying Cheng, Fan Zhang, and Qiang Xu. Making slow thinking faster: Compressing LLM chain-of-thought via step entropy. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=cGLqQfS5wH.

Zhong-Zhi Li, Xiao Liang, Zihao Tang, Lei Ji, Peijie Wang, Haotian Xu, Xing W, Haizhen Huang, Weiwei Deng, Yeyun Gong, Zhijiang Guo, Xiao Liu, Fei Yin, and Cheng-Lin Liu. Tl;dr: Too long, do re-weighting for efficient llm reasoning compression, 2025b. URL https://arxiv.org/abs/2506.02678.

Junhong Lin, Xinyue Zeng, Jie Zhu, Song Wang, Julian Shun, Jun Wu, and Dawei Zhou. Plan and budget: Effective and efficient test-time scaling on reasoning large language models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=ctspw4CqbS.

Kang Liu, Yongkang Liu, Xiaocui Yang, Peidong Wang, Wen Zhang, Shi Feng, Yifei Zhang, and Daling Wang. Neat: Neuron-based early exit for large reasoning models. *arXiv preprint arXiv:2602.02010*, 2026.

Xin Liu and Lu Wang. Answer convergence as a signal for early stopping in reasoning. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 17896–17907, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.904. URL https://aclanthology.org/2025.emnlp-main.904/.

Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=KUNzEQMWU7.

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. In *2nd AI for Math Workshop @ ICML 2025*, 2025. URL https://openreview.net/forum?id=ioYybCRcyW.

Wenjie Ma, Jingxuan He, Charlie Snell, Tyler Griggs, Sewon Min, and Matei Zaharia. Reasoning models can be effective without thinking. *arXiv preprint arXiv:2504.09858*, 2025a.

Xinyin Ma, Guangnian Wan, Runpeng Yu, Gongfan Fang, and Xinchao Wang. CoT-valve: Length-compressible chain-of-thought tuning. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6025–6035, Vienna, Austria, July 2025b. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.300. URL https://aclanthology.org/2025.acl-long.300/.

Bill MacCartney. *Natural language inference*. Stanford University, 2009.

Minjia Mao, Bowen Yin, Yu Zhu, and Xiao Fang. Early stopping chain-of-thoughts in large language models. *arXiv preprint arXiv:2509.14004*, 2025.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candes, and Tatsunori Hashimoto. s1: Simple test-time scaling. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20275–20321, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1025. URL https://aclanthology.org/2025.emnlp-main.1025/.

Sania Nayab, Giulio Rossolini, Marco Simoni, Andrea Saracino, Giorgio Buttazzo, Nicolamaria Manes, and Fabrizio Giacomelli. Concise thoughts: Impact of output length on llm reasoning and cost. *arXiv preprint arXiv:2407.19825*, 2024.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

OpenAI. gpt-oss-120b & gpt-oss-20b model card, 2025. URL https://arxiv.org/abs/2508.10925.

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 Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 8010–8029, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.405. URL https://aclanthology.org/2025.emnlp-main.405/.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 8748–8763. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/radford21a.html.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 53728–53741. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/a85b405ed65c6477a4fe8302b5e06ce7-Paper-Conference.pdf.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

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, 2024. doi: 10.1109/FLLM63129.2024.10852493.

Aaroosh Rustagi, Hsien Xin Peng, Khushal Murthy, Attrey Koul, Ryan Lagasse, and Kevin Zhu. Confidence-coverage gating for early exit. In *NeurIPS 2025 Workshop on Efficient Reasoning*, 2025. URL https://openreview.net/forum?id=Ay7sRmWswq.

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

Charlie Victor 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.

Jinyan Su, Jennifer Healey, Preslav Nakov, and Claire Cardie. Between underthinking and overthinking: An empirical study of reasoning length and correctness in llms. *arXiv preprint arXiv:2505.00127*, 2025.

Yang Sui, Yu-Neng Chuang, Guanchu Wang, Jiamu Zhang, Tianyi Zhang, Jiayi Yuan, Hongyi Liu, Andrew Wen, Shaochen Zhong, Na Zou, Hanjie Chen, and Xia Hu. Stop overthinking: A survey on efficient reasoning for large language models. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=HvoG8SxggZ.

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

GLM Team, Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, Kedong Wang, Lucen Zhong, Mingdao Liu, Rui Lu, Shulin Cao, Xiaohan Zhang, Xuancheng Huang, Yao Wei, Yean Cheng, Yifan An, Yilin Niu, Yuanhao Wen, Yushi Bai, Zhengxiao Du, Zihan Wang, Zilin Zhu, Bohan Zhang, Bosi Wen, Bowen Wu, Bowen Xu, Can Huang, Casey Zhao, Changpeng Cai, Chao Yu, Chen Li, Chendi Ge, Chenghua Huang, Chenhui Zhang, Chenxi Xu, Chenzheng Zhu, Chuang Li, Congfeng Yin, Daoyan Lin, Dayong Yang, Dazhi Jiang, Ding Ai, Erle Zhu, Fei Wang, Gengzheng Pan, Guo Wang, Hailong Sun, Haitao Li, Haiyang Li, Haiyi Hu, Hanyu Zhang, Hao Peng, Hao Tai, Haoke Zhang, Haoran Wang, Haoyu Yang, He Liu, He Zhao, Hongwei Liu, Hongxi Yan, Huan Liu, Huilong Chen, Ji Li, Jiajing Zhao, Jiamin Ren, Jian Jiao, Jiani Zhao, Jianyang Yan, Jiaqi Wang, Jiayi Gui, Jiayue Zhao, Jie Liu, Jijie Li, Jing Li, Jing Lu, Jingsen Wang, Jingwei Yuan, Jingxuan Li, Jingzhao Du, Jinhua Du, Jinxin Liu, Junkai Zhi, Junli Gao, Ke Wang, Lekang Yang, Liang Xu, Lin Fan, Lindong Wu, Lintao Ding, Lu Wang, Man Zhang, Minghao Li, Minghuan Xu, Mingming Zhao, Mingshu Zhai, Pengfan Du, Qian Dong, Shangde Lei, Shangqing Tu, Shangtong Yang, Shaoyou Lu, Shijie Li, Shuang Li, Shuang-Li, Shuxun Yang, Sibo Yi, Tianshu Yu, Wei Tian, Weihan Wang, Wenbo Yu, Weng Lam Tam, Wenjie Liang, Wentao Liu, Xiao Wang, Xiaohan Jia, Xiaotao Gu, Xiaoying Ling, Xin Wang, Xing Fan, Xingru Pan, Xinyuan Zhang, Xinze Zhang, Xiuqing Fu, Xunkai Zhang, Yabo Xu, Yandong Wu, Yida Lu, Yidong Wang, Yilin Zhou, Yiming Pan, Ying Zhang, Yingli Wang, Yingru Li, Yinpei Su, Yipeng Geng, Yitong Zhu, Yongkun Yang, Yuhang Li, Yuhao Wu, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yuxuan Zhang, Zezhen Liu, Zhen Yang, Zhengda Zhou, Zhongpei Qiao, Zhuoer Feng, Zhuorui Liu, Zichen Zhang, Zihan Wang, Zijun Yao, Zikang Wang, Ziqiang Liu, Ziwei Chai, Zixuan Li, Zuodong Zhao, Wenguang Chen, Jidong Zhai, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models, 2025a. URL https://arxiv.org/abs/2508.06471.

Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, Zhuofu Chen, Jialei Cui, Hao Ding, Mengnan Dong, Angang Du, Chenzhuang Du, Dikang Du, Yulun Du, Yu Fan, Yichen Feng, Kelin Fu, Bofei Gao, Hongcheng Gao, Peizhong Gao, Tong Gao, Xinran Gu, Longyu Guan, Haiqing Guo, Jianhang Guo, Hao Hu, Xiaoru Hao, Tianhong He, Weiran He, Wenyang He, Chao Hong, Yangyang Hu, Zhenxing Hu, Weixiao Huang, Zhiqi Huang, Zihao Huang, Tao Jiang, Zhejun Jiang, Xinyi Jin, Yongsheng Kang, Guokun Lai, Cheng Li, Fang Li, Haoyang Li, Ming Li, Wentao Li, Yanhao Li, Yiwei Li, Zhaowei Li, Zheming Li, Hongzhan Lin, Xiaohan Lin, Zongyu Lin, Chengyin Liu, Chenyu Liu, Hongzhang Liu, Jingyuan Liu, Junqi Liu, Liang Liu, Shaowei Liu, T. Y. Liu, Tianwei Liu, Weizhou Liu, Yangyang Liu, Yibo Liu, Yiping Liu, Yue Liu, Zhengying Liu, Enzhe Lu, Lijun Lu, Shengling Ma, Xinyu Ma, Yingwei Ma, Shaoguang Mao, Jie Mei, Xin Men, Yibo Miao, Siyuan Pan, Yebo Peng, Ruoyu Qin, Bowen Qu, Zeyu Shang, Lidong Shi, Shengyuan Shi, Feifan Song, Jianlin Su, Zhengyuan Su, Xinjie Sun, Flood Sung, Heyi Tang, Jiawen Tao, Qifeng Teng, Chensi Wang, Dinglu Wang, Feng Wang, Haiming Wang, Jianzhou Wang, Jiaxing Wang, Jinhong Wang, Shengjie Wang, Shuyi Wang, Yao Wang, Yejie Wang, Yiqin Wang, Yuxin Wang, Yuzhi Wang, Zhaoji Wang, Zhengtao Wang, Zhexu Wang, Chu Wei, Qianqian Wei, Wenhao Wu, Xingzhe Wu, Yuxin Wu, Chenjun Xiao, Xiaotong Xie, Weimin Xiong, Boyu Xu, Jing Xu, Jinjing Xu, L. H. Xu, Lin Xu, Suting Xu, Weixin Xu, Xinran Xu, Yangchuan Xu, Ziyao Xu, Junjie Yan, Yuzi Yan, Xiaofei Yang, Ying Yang, Zhen Yang, Zhilin Yang, Zonghan Yang, Haotian Yao, Xingcheng Yao, Wenjie Ye, Zhuorui Ye, Bohong Yin, Longhui Yu, Enming Yuan, Hongbang Yuan, Mengjie Yuan, Haobing Zhan, Dehao Zhang, Hao Zhang, Wanlu Zhang, Xiaobin Zhang, Yangkun Zhang, Yizhi Zhang, Yongting Zhang, Yu Zhang, Yutao Zhang, Yutong Zhang, Zheng Zhang, Haotian Zhao, Yikai Zhao, Huabin Zheng, Shaojie Zheng, Jianren Zhou, Xinyu Zhou, Zaida Zhou, Zhen Zhu, Weiyu Zhuang, and Xinxing Zu. Kimi k2: Open agentic intelligence, 2025b. URL https://arxiv.org/abs/2507.20534.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*, 2025c.

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

Katherine Tian, Eric Mitchell, Allan Zhou, Archit Sharma, Rafael Rafailov, Huaxiu Yao, Chelsea Finn, and Christopher D Manning. Just ask for calibration: Strategies for eliciting calibrated

confidence scores from language models fine-tuned with human feedback. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://openreview.net/forum?id=g3faCfrwm7.

Henrique Schechter Vera, Sahil Dua, Biao Zhang, Daniel Salz, Ryan Mullins, Sindhu Raghuram Panyam, Sara Smoot, Iftekhar Naim, Joe Zou, Feiyang Chen, et al. EmbeddingGemma: Powerful and lightweight text representations. *arXiv preprint arXiv:2509.20354*, 2025.

Chenlong Wang, Yuanning Feng, Dongping Chen, Zhaoyang Chu, Ranjay Krishna, and Tianyi Zhou. Wait, we don't need to "wait"! removing thinking tokens improves reasoning efficiency. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 7459–7482, Suzhou, China, November 2025a. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.394. URL https://aclanthology.org/2025.findings-emnlp.394/.

Ke Wang, Junting Pan, Weikang Shi, Zimu Lu, Houxing Ren, Aojun Zhou, Mingjie Zhan, and Hongsheng Li. Measuring multimodal mathematical reasoning with math-vision dataset. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=QWTCcxMpPA.

Xi Wang, James McInerney, Lequn Wang, and Nathan Kallus. Entropy after {/Think} for reasoning model early exiting. *arXiv preprint arXiv:2509.26522*, 2025b.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf.

Zihao Wei, Liang Pang, Jiahao Liu, Wenjie Shi, Jingcheng Deng, Shicheng Xu, Zenghao Duan, Jingang Wang, Fei Sun, Huawei Shen, et al. The evolution of thought: Tracking LLM overthinking via reasoning dynamics analysis. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 26905–26920, 2026.

Yuyang Wu, Yifei Wang, Tianqi Du, Stefanie Jegelka, and Yisen Wang. When more is less: Understanding chain-of-thought length in LLMs. In *Workshop on Reasoning and Planning for Large Language Models*, 2025. URL https://openreview.net/forum?id=W8dxn7hBkO.

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

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.

Chenxu Yang, Qingyi Si, Yongjie Duan, Zheliang Zhu, Chenyu Zhu, Qiaowei Li, Minghui Chen, Zheng Lin, and Weiping Wang. Dynamic early exit in reasoning models. In *The Fourteenth International Conference on Learning Representations*, 2026a. URL https://openreview.net/forum?id=NpU7ZXafRi.

Wenkai Yang, Shuming Ma, Yankai Lin, and Furu Wei. Towards thinking-optimal scaling of test-time compute for LLM reasoning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026b. URL https://openreview.net/forum?id=6ICFqmixlS.

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

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Yixin Ye, Zhen Huang, Yang Xiao, Ethan Chern, Shijie Xia, and Pengfei Liu. LIMO: Less is more for reasoning. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=T2TZ0RY4Zk.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, YuYue, Weinan Dai, Tiantian Fan, Gaohong Liu, Juncai Liu, LingJun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Ru Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Yonghui Wu, and Mingxuan Wang. DAPO: An open-source LLM reinforcement learning system at scale. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026. URL https://openreview.net/forum?id=2a36EMSSTp.

Ye Yu, Yaoning Yu, Haibo Jin, and Haohan Wang. PREMISE: Scalable and strategic prompt optimization for efficient mathematical reasoning in large reasoning models. In *NeurIPS 2025 Workshop on Efficient Reasoning*, 2025. URL https://openreview.net/forum?id=8mI3i9LXj3.

Anqi Zhang, Yulin Chen, Jane Pan, Chen Zhao, Aurojit Panda, Jinyang Li, and He He. Reasoning models know when they're right: Probing hidden states for self-verification. In *Second Conference on Language Modeling*, 2025a. URL https://openreview.net/forum?id=O6I0Av7683.

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 Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 13307–13328, Suzhou, China, November 2025b. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.673. URL https://aclanthology.org/2025.emnlp-main.673/.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025c.

Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2024, 2024.

Yifan Zhang and Team Math-AI. American invitational mathematics examination (aime) 2025, 2025.

Yuze Zhao, Jintao Huang, Jinghan Hu, Xingjun Wang, Yunlin Mao, Daoze Zhang, Zeyinzi Jiang, Zhikai Wu, Baole Ai, Ang Wang, Wenmeng Zhou, and Yingda Chen. Swift:a scalable lightweight infrastructure for fine-tuning, 2024. URL https://arxiv.org/abs/2408.05517.

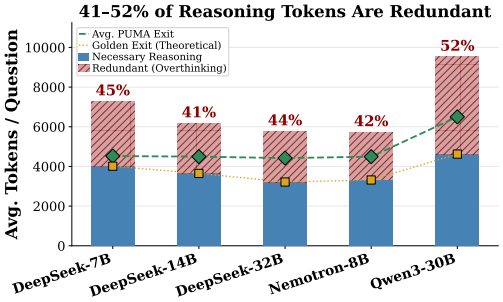
Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

A Overthinking Analysis and Failure Modes of Answer-Level Signals

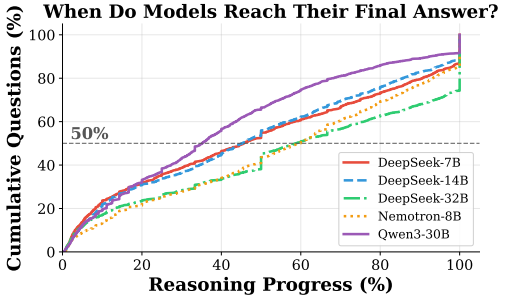
A.1 Overthinking Prevalence

To quantify the extent of overthinking, we perform a counterfactual analysis on full CoT reasoning traces across five representative LRMs and five benchmarks. For each question, we identify the *golden step*: the earliest reasoning step at which the model's intermediate trial answer matches its final answer. The tokens generated after the golden step are classified as post-answer redundancy.

Figure 5 summarizes the results. Across all five models, 41–52% of reasoning tokens are generated after the model has already committed to its final answer. The pattern is consistent across model families and scales: even the strongest model (Qwen3-30B-A3B-Thinking) spends over half its tokens on post-answer re-verification, rephrasing, and re-derivation. The cumulative distribution in Figure 5b further shows that models typically commit to their final answer well before the reasoning chain completes, with most models reaching the 50% mark between 40–60% of reasoning progress.



(a) Token breakdown before and after the model first reaches its final answer.



(b) Cumulative fraction of questions whose final answer has already been reached, as a function of reasoning progress.

Figure 5: **Overthinking analysis across five representative LRMs.** Left: a substantial fraction of reasoning tokens is generated after the model has already reached its final answer, accounting for 41–52% of tokens across models. Right: many examples reach the final answer well before the end of the reasoning chain, further illustrating the prevalence of post-answer overthinking.

A.2 Failure Rates of Answer-Level Stopping Signals

We analyze the failure rates of the two dominant classes of answer-level stopping signals by applying their criteria retroactively to full CoT traces. At each reasoning step, we induce a trial answer and compute its confidence score as the geometric mean of token-level log-probabilities, following DEER (Yang et al., 2026a), and check trial-answer consistency across consecutive steps (Liu & Wang, 2025; Fu et al., 2025).

We evaluate across five models and five benchmarks. For the confidence signal, we flag a step as a candidate exit whenever the confidence score exceeds $\lambda = 0.95$, following DEER (Yang et al., 2026a). For the consistency signal, we flag a step whenever the model produces k=3 consecutive identical trial answers, following Dynasor (Fu et al., 2025). Table 5 reports the failure rate of each signal, defined as the fraction of first-triggered exits where the trial answer is incorrect. Averaged over all 25 (model, benchmark) combinations, the failure rate is **44%** for confidence-based exits and **64%** for consistency-based exits. Failure rates are highest on the most challenging benchmarks: on AIME24 and AIME25, consistency-based exits are incorrect in up to 83% of triggered steps, while even confidence-based exits fail up to 69% of the time.

A.3 Counterfactual Analysis: Do Signal Misfires Prevent Self-Correction?

The failure rates above show how often a signal would stop at an incorrect trial answer, but they do not tell whether early stopping itself changes the final outcome: in some cases, the model may

Table 5: Failure rates (%) of answer-level stopping signals across benchmarks and model families. A failure is defined as a first-triggered exit where the trial answer is incorrect.

	AIME24	AIME25	MATH-500	OlymBench	GPQA-D	Avg.
DeepSeek-R1-Distill-Qwen-7B						
Confidence	50.0	63.3	31.9	59.3	38.4	48.6
Consistency	76.9	70.0	36.5	67.9	67.2	63.7
DeepSeek-R1-Distill-Qwen-14B						
Confidence	53.3	62.1	28.8	52.3	35.3	46.4
Consistency	80.0	79.3	37.1	70.5	61.7	65.7
DeepSeek-R1-Distill-Qwen-32B						
Confidence	34.5	56.7	19.9	46.8	26.8	36.9
Consistency	69.0	72.4	45.5	70.1	59.6	63.3
Llama-3.1-Nemotron-Nano-8B						
Confidence	44.8	69.0	32.7	57.3	54.2	51.6
Consistency	83.3	83.3	34.9	67.9	77.3	69.3
Qwen3-30B-A3B-Thinking						
Confidence	29.6	51.9	26.3	48.4	36.5	38.5
Consistency	73.3	67.9	31.1	64.0	55.3	58.3
All models avg.						
Confidence	42.4	60.6	27.9	52.8	38.2	**44.4**
Consistency	76.5	74.6	37.0	68.1	64.2	**64.1**

Table 6: Counterfactual analysis of answer-level stopping signals, aggregated across all five benchmarks. **Premature%** denotes the fraction of incorrect exits where the corresponding uninterrupted Full-CoT trajectory would eventually reach the correct answer, meaning that early stopping would prevent later self-correction and induce accuracy loss.

	Premature%	
Model	**Confidence**	**Consistency**
DeepSeek-R1-Distill-Qwen-7B	41.1	47.8
DeepSeek-R1-Distill-Qwen-14B	39.8	54.6
DeepSeek-R1-Distill-Qwen-32B	28.7	52.0
Llama-3.1-Nemotron-Nano-8B	45.2	53.5
Qwen3-30B-A3B-Thinking	57.4	66.0
Average	**42.4**	**54.8**

already be on a trajectory that remains wrong even if allowed to continue. To distinguish genuinely premature exits from non-recoverable failures, we compare each failed first-triggered exit against the corresponding full, uninterrupted reasoning chain.

We categorize a failed exit as a **premature exit** if the trial answer at the triggered step is incorrect but the full reasoning chain eventually reaches the correct final answer. In contrast, we categorize it as a **non-recoverable failure** if the full reasoning chain also ends with an incorrect answer. Premature exits are especially harmful because the stopping signal would convert an otherwise correct full-CoT trajectory into an incorrect early-exit output.

Table 6 reports the breakdown aggregated across all five benchmarks. A substantial fraction of signal failures are premature rather than non-recoverable: across all models, **42.4**% of confidence-based failures and **54.8**% of consistency-based failures occur on traces where the model would have self-corrected if allowed to continue. This shows that answer-level signals do not merely fire on trajectories that remain wrong; they often fire while the model is still in the process of correcting or revising its answer.

The counterfactual results also explain why trial-answer consistency can be particularly misleading. A model may produce the same wrong trial answer for several consecutive steps while it is still exploring the problem, creating the appearance of answer stability before the reasoning trajectory has actually converged. Stopping at this point prevents later correction. These findings reinforce the need for a reasoning-level signal: safe early exit should not only ask whether the current answer appears stable, but also whether the reasoning process has stopped making semantically novel progress.

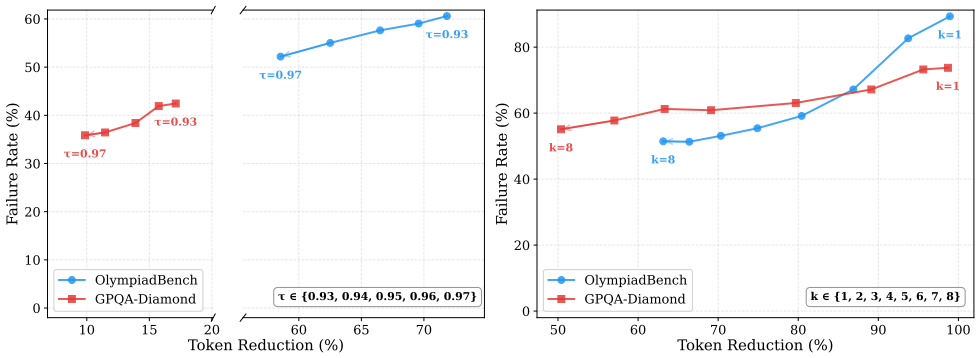


Figure 6: Threshold sensitivity of standalone answer-level stopping signals. We sweep the confidence threshold λ and the consistency window size k on OlympiadBench and GPQA Diamond using DeepSeek-R1-Distill-Qwen-7B. Each point corresponds to one operating point. Across thresholds, standalone answer-level signals exhibit a persistent tradeoff between failure rate and token reduction, without a clear setting that is both safe and efficient.

A.4 Threshold Sensitivity of Answer-Level Stopping Signals

One possible concern is that the high failure rates above may simply reflect suboptimal threshold choices. To test this, we sweep each answer-level signal over a range of operating points and examine the resulting tradeoff between token reduction and failure rate. For confidence-based stopping, we vary the confidence threshold $\lambda \in \{0.93, 0.94, 0.95, 0.96, 0.97\}$. For consistency-based stopping, we vary the required number of consecutive identical trial answers $k \in \{1, 2, 3, 4, 5, 6, 7, 8\}$. We run this diagnostic analysis on DeepSeek-R1-Distill-Qwen-7B over two challenging benchmarks, OlympiadBench and GPQA Diamond.

Figure 6 shows that threshold tuning does not remove the fundamental tradeoff for standalone answer-level stopping signals. For confidence-based stopping on OlympiadBench, even conservative thresholds still yield high failure rates while stopping a large fraction of reasoning tokens. On GPQA Diamond, lower failure rates are possible only when token reduction becomes small, making early exit much less useful. Consistency-based stopping shows a similar pattern: increasing k reduces some premature triggers, but does not produce a clear operating point that is both safe and efficient. These results suggest that the limitation is not merely a poor threshold choice; standalone answer-level stopping signals are insufficient to determine whether the reasoning trajectory has actually converged.

B Implementation Details

B.1 Reasoning Step Segmentation

PUMA operates at the level of reasoning steps rather than individual tokens. We therefore segment each reasoning chain into coherent steps using a deterministic, lightweight procedure. Reasoning step segmentation is a common preprocessing step in recent work on efficient reasoning, where approaches often rely on heuristic segmentations such as blank-line splitting (Li et al., 2026), keyword-based transition detection (Zhang et al., 2025a), or explicit block-marker matching (Qiao et al., 2025). Our procedure follows the blank-line approach and adds a lightweight merging stage to control step granularity. The segmentation is applied identically during Redundancy Detector training and at inference time, ensuring that the detector sees the same type of step units in both settings.

Given the raw reasoning text, we first split the chain at blank-line boundaries, which preserves the model's natural paragraph structure. We then assign each paragraph a coarse semantic role using simple string-level cues, such as problem setup, calculation, self-correction, verification, conclusion, or general reasoning. These labels are not used as model inputs; they only guide how adjacent paragraphs are grouped into larger reasoning steps.

The final merging stage balances semantic coherence and step length. Enumerated paragraphs or major semantic transitions are treated as natural step boundaries, while short adjacent paragraphs with compatible roles are merged when doing so preserves coherence. We use a target step length range of $[L_{\min}, L_{\max}] = [200, 1000]$ characters: very short segments are often too noisy for redundancy detection, while overly long segments may hide local repetition. Tiny trailing segments are merged into the preceding step.

The procedure is intentionally string-based and does not invoke any learned segmenter, adding negligible overhead to generation. This design keeps PUMA compatible with standard serving pipelines while providing stable step units for both detector training and online early-exit decisions.

B.2 Redundancy Detector Training

We construct a large-scale contrastive dataset to train the Redundancy Detector. The goal is to teach the detector whether a new reasoning step contributes semantically novel progress relative to prior reasoning, or instead restates, re-derives, or loops over existing content.

Source reasoning traces. We collect long-CoT reasoning traces from models that are not used as main evaluation models, including QwQ-32B (Team, 2025), GPT-OSS-120B (OpenAI, 2025), GLM-4.7-Thinking (Team et al., 2025a), and Kimi-K2-Thinking (Team et al., 2025b). These models are prompted on AMC23 and GSM8K (Cobbe et al., 2021), and we further include reasoning chains sampled from Open-Thoughts-114K (Guha et al., 2026). In total, the source collection covers approximately 5,098 distinct questions. After applying the reasoning-step segmentation procedure in Appendix B.1, we obtain roughly 1.97M candidate step pairs.

Stage 1: seed annotation. We first annotate a seed set of reasoning-step pairs using GPT-5-mini. The annotation prompt asks the model to provide a short rationale and then assign a binary label, where $y = 0$ denotes a novel step and $y = 1$ denotes a redundant step. This stage yields 40,844 annotated examples across the four source models. Each example can contain multiple redundant positives and multiple novel negatives; on average, a seed example contains 3.8 positives and 26.6 negatives. After expanding multi-positive examples into single-positive contrastive rows, this stage contributes approximately 155K anchor–positive training pairs.

Stage 2: redundant-step synthesis. To increase coverage of redundant reasoning patterns, we synthesize redundant counterparts for seed-labeled novel steps. Given a question, a previous reasoning step, and a novel current step, GPT-4o-mini rewrites the current step into a redundant version that preserves the topic but removes substantive new progress. The rewrite prompt covers common redundancy modes such as restatement, hesitation, paraphrase, and transitional padding. This stage adds approximately 560K synthetic redundant pairs. The total annotation cost across both stages is around $2,000.

Final contrastive dataset. We convert all data into the InfoNCE format used for training, where each row contains an anchor step, one redundant positive, and a set of novel negatives. After expanding multi-positive annotations, adding synthetic redundant pairs, deduplicating examples, and removing rows without valid negatives, the final dataset contains 701,641 contrastive training rows.

Training setup. We implement detector training using the MS-Swift framework (Zhao et al., 2024). Instead of full-parameter fine-tuning, we adapt Qwen3-Embedding-0.6B with LoRA to reduce overfitting risk. We use LoRA rank $r = 32$, $\alpha = 64$, dropout 0.1, and apply LoRA to the `q_proj`, `k_proj`, `v_proj`, `o_proj`, `gate_proj`, `up_proj`, and `down_proj` modules. The detector is trained with an InfoNCE contrastive objective, batch size 16 with gradient accumulation 4 (effective batch size 64), learning rate 1×10^{-4}, and 5 epochs.

Detector quality and threshold selection. We evaluate the trained detector on a held-out set of anchor-positive-negative triples, where the positive is a redundant step and the negative is a novel step relative to the anchor. The detector achieves 91.26% pairwise ranking accuracy, where a prediction is counted as correct if

$$\mathrm{sim}(\text{anchor}, \text{redundant}) > \mathrm{sim}(\text{anchor}, \text{novel}). \tag{4}$$

The average cosine similarity is 0.333 for redundant pairs and 0.184 for novel pairs, yielding an average margin of 0.148. For the default PUMA setting, we set $\tau_{\text{sim}} = 0.35$, selected on a held-out

calibration set as a conservative operating point for candidate-exit detection. At this threshold, the detector achieves 91.54% absolute classification accuracy and 93.58% true-negative rate. The high true-negative rate makes the detector conservative: it avoids over-triggering on genuinely novel reasoning steps, while missed redundant steps mainly reduce potential token savings rather than directly harming correctness. Moreover, all detector-flagged candidate exits are still filtered by Answer Verification before PUMA stops.

Effect of the synthetic redundant pairs. To measure how much the redundant pairs synthesized by GPT-4o-mini add to the seed annotations, we train two detectors from the same base model, one on the seed annotations only and one on the seed plus synthetic data, and run PUMA end-to-end with each on DS-7B and MATH-500, reusing the same full CoT traces (Table 7). Adding the synthetic pairs raises token reduction from 25.4% to 30.2%, and the detector trained on seed plus synthetic data is the only configuration that compresses without accuracy loss. The synthesis step only rewrites an already-labeled novel step into a redundant counterpart under a fixed set of redundancy modes, a constrained transformation rather than open-ended generation. Together with the gap between the base and fine-tuned embeddings (Table 11), this suggests that task-specific contrastive training, rather than the identity of the rewriting model, determines detector quality.

Table 7: Effect of the detector's training data on DS-7B and MATH-500. *No detector*: Answer Verification is run at every step. *Seed only*: the detector is trained on the GPT-5-mini seed annotations. *Seed + synthetic*: the detector is additionally trained on the redundant pairs synthesized by GPT-4o-mini (PUMA's default). All rows use the same full CoT traces (accuracy 89.8%); parentheses give the accuracy change relative to them.

Detector training data	**Acc**	**TR**
No detector (verification at every step)	89.0 (-0.8)	21.7
Seed only	89.4 (-0.4)	25.4
Seed + synthetic (PUMA)	89.8 (0.0)	30.2

LLM Annotation Prompts for Redundancy Detector Training We use two LLM-based prompts to construct supervision for the Redundancy Detector: a seed novelty annotation prompt (Table 8) and a redundant-step synthesis prompt (Table 9). For clarity, we present both prompts under the label convention used in our final training data, where $y = 0$ denotes a novel reasoning step and $y = 1$ denotes a redundant reasoning step.

B.3 Choice of Redundancy Signal

Semantic Entropy commonly operationalizes semantic equivalence using Natural Language Inference (NLI), by asking whether two textual outputs mutually entail each other. Motivated by this connection, we compare PUMA's embedding-based Redundancy Detector with two NLI-style alternatives. The first variant, **ICL-NLI**, uses Qwen3-0.6B with 4-shot in-context prompting: given two reasoning steps and four labelled examples, the model emits a binary "Yes/No" judgment for whether the current step is redundant with respect to the previous one. The second variant, **FT-NLI**, uses the same Qwen3-0.6B backbone fine-tuned on 10K NLI-style redundancy examples. For both variants, the binary judgment replaces PUMA's default redundancy score; all other PUMA components are kept unchanged.

Table 11 reports results on DS-7B and Nemotron-8B across five benchmarks. The embedding-based detector achieves the best accuracy–efficiency trade-off: it is the only signal that preserves (and even slightly improves) accuracy over Full CoT (+1.3 on average) while still achieving meaningful token reduction (22.8%). ICL-NLI is too conservative: it rarely fires and leaves most reasoning uncompressed (6.5% TR). FT-NLI shows the opposite failure mode: it achieves 31.3% TR but drops accuracy by 3.8 points, indicating that NLI-style judgments do not align well with reasoning-level convergence even after task-specific fine-tuning.

Beyond accuracy and token reduction, the embedding detector is substantially cheaper at inference time, averaging about 21 ms per question compared with $\sim$93 ms for both NLI variants. These results support using the trained embedding model as PUMA's default redundancy signal.

Table 8: Prompt template for seed novelty annotation. The displayed label convention matches the remapped labels used in the final Redundancy Detector training data.

Seed novelty annotation

Role	You are an AI assistant for assessing the quality of logical reasoning. The user will provide a question, a previous reasoning step, and a current reasoning step. Your job is to judge whether the current step provides additional information compared to the previous step. Do not solve the problem yourself; only evaluate the novelty of the current step.
Labels	Output `0` if the current step introduces new logical or semantic information. Output `1` if the current step is redundant, i.e., it mainly restates, rephrases, verifies, or loops over information already present in the previous step.
Instructions	First provide a short explanation enclosed in `<explanation>...</explanation>`. Then output the final label enclosed in `<result>...</result>`. The result must be exactly one character, either `0` or `1`. Output nothing after the closing `</result>` tag.
Examples	*[Four worked demonstrations illustrating novel vs. redundant reasoning-step judgments.]*
Input	Problem: `<PROBLEM>` Previous step: `<PREVIOUS STEP>` Current step: `<CURRENT STEP>`

Output	Respond in the following format:
	`<explanation>`
	`<WHY THE CURRENT STEP IS NOVEL OR REDUNDANT>`
	`</explanation>`
	`<result>0 or 1</result>`

Table 9: Prompt template for redundant-step synthesis. GPT-4o-mini rewrites seed-labeled novel steps into redundant counterparts to augment positive examples for contrastive training.

Redundant-step synthesis

Role	You are an expert at analyzing mathematical reasoning steps. Your task is to generate a redundant version of a given reasoning step.
Definition	A redundant step is one that does not advance the solution compared to the previous step. It may restate or summarize what was already said, express hesitation or self-checking without new insight, rephrase the same mathematical relationship in different words, or add transitional phrases without actual progress.
Instructions	Given a mathematical problem, a previous reasoning step, and a current step that provides new information, generate a redundant version of the current step. The redundant version should: • not provide new information beyond what is already in the previous step; • sound natural as a reasoning step; • be similar in length to the current step.
Examples	*[Examples demonstrate common redundancy modes, including restatement, hesitation, paraphrase, and transitional padding.]*
Input	Problem: `<PROBLEM>` Previous step: `<PREVIOUS STEP>` Current step (novel): `<CURRENT STEP THAT INTRODUCES NEW INFORMATION>`
Output	Redundant version: `<REWRITTEN STEP THAT SOUNDS NATURAL BUT ADDS NO NEW PROGRESS>`

We additionally compare against the base (unfine-tuned) Qwen3-Embedding-0.6B to isolate the contribution of contrastive training. Base-Emb achieves higher token reduction but at a substantial accuracy cost, and varying its similarity threshold does not recover the gap, suggesting that task-specific fine-tuning is essential for reliable redundancy detection.

Robustness to the detector backbone. To test whether PUMA depends on a particular embedding model, we retrain the Redundancy Detector with the same data and objective on two additional backbones: Qwen3-Embedding-4B (Zhang et al., 2025c), from the same family but 6.7× larger, and EmbeddingGemma-300M (Vera et al., 2025), from a different family and 2× smaller. Each detector uses its own similarity threshold ($\tau_{\text{sim}} = 0.64$ for EmbeddingGemma-300M and 0.59 for Qwen3-Embedding-4B), and all other PUMA components are unchanged. Table 10 shows that PUMA's accuracy stays within 0.3 points of the default detector on all five LRMs with either backbone. EmbeddingGemma-300M, although half the size of the default detector, gives slightly higher token reduction on four of the five models and on average (21.0% vs. 20.6%). Qwen3-Embedding-4B is more conservative, removing 1.9 to 2.7 fewer points of tokens at the same average accuracy. These experiments show that PUMA's improvements do not depend on a particular detector backbone. They come from the method itself, namely the redundancy signal learned through contrastive training and the answer verification that gates each exit, so even a smaller detector suffices.

Table 10: PUMA with different Redundancy Detector backbones: overall accuracy (Acc, %) and token reduction (TR, %), averaged over the five benchmarks.

	Qwen3-Emb-0.6B (default)		**EmbeddingGemma-300M**		**Qwen3-Emb-4B**	
Model	**Acc**	**TR**	**Acc**	**TR**	**Acc**	**TR**
DS-7B	60.2	29.2	60.0	29.6	60.2	26.8
DS-14B	64.6	18.1	64.3	18.6	64.8	15.9
DS-32B	69.7	16.5	69.4	16.1	69.6	14.3
Nemotron-8B	64.7	16.5	64.5	17.2	64.4	14.6
Qwen3-30B-T	82.5	22.8	82.3	23.5	82.6	20.1
Average	68.3	20.6	68.1	21.0	68.3	18.3

Why an external detector rather than hidden-state probes. A natural alternative is to probe the target LRM's intermediate hidden states, for example through the similarity of layer activations at successive steps or the divergence of the output distribution, as done by hidden-state probes for answer correctness and early exit (Zhang et al., 2025a; Akgül et al., 2025). We use an external detector for three reasons. First, standard serving stacks such as vLLM (Kwon et al., 2023) do not expose intermediate activations, whereas PUMA is designed to work with unmodified deployed models. Second, hidden-state signals are tied to a specific model's layers and representation geometry, so they would require per-model layer selection and recalibration, and existing probes are trained separately for each target model (Zhang et al., 2025a; Akgül et al., 2025); our detector is trained once and applied unchanged to five LRMs and, zero-shot, to two vision-language models. Third, activation similarity conflates topical similarity with a lack of progress, which is exactly the distinction the detector's contrastive training targets, and output-distribution divergence similarly reflects a settled surface pattern rather than semantic convergence. We leave a direct comparison with hidden-state probes to future work.

B.4 Redundancy Detector Lookback Window

The Redundancy Detector compares the current reasoning step against the previous k reasoning steps and flags a candidate exit when the maximum similarity exceeds τ_{sim}. PUMA uses $k = 1$ by default, comparing each step only with its immediate predecessor. Table 12 evaluates $k \in \{1, 2, 4, 8, \text{all}\}$ on DS-7B and Nemotron-8B, where $k = \text{all}$ compares the current step against all preceding steps.

Across both models, $k = 1$ provides the best accuracy-preserving behavior. Increasing k yields more aggressive exits and higher token reduction, but it also worsens ΔAcc on average, from $+1.3$ at $k = 1$ to -2.9 at $k = \text{all}$. This pattern suggests that larger windows introduce additional false-positive triggers: a current step may resemble a much earlier step while still contributing useful progress relative to the immediately preceding context. As a result, looking too far back can incorrectly trigger

Table 11: Redundancy signal and detector comparison. Fine-tuned Embedding is PUMA's default detector: a LoRA-adapted Qwen3-Embedding-0.6B trained with an InfoNCE contrastive objective. Base Embedding uses the off-the-shelf Qwen3-Embedding-0.6B without contrastive fine-tuning, with two representative similarity thresholds. ICL-NLI and FT-NLI replace the detector with NLI-style redundancy judgments while keeping all other PUMA components unchanged. ΔAcc and TR are averaged over the five benchmarks.

	DS-7B		Nemotron-8B		Average	
Signal	**ΔAcc↑**	**TR↑**	**ΔAcc↑**	**TR↑**	**ΔAcc↑**	**TR↑**
Fine-tuned Emb. (PUMA)	**+2.2**	29.2	**+0.4**	16.5	**+1.3**	22.8
Base Emb. ($\tau = 0.42$)	-2.9	36.4	-3.3	31.4	-3.1	33.9
Base Emb. ($\tau = 0.50$)	-4.1	33.1	-3.3	31.0	-3.7	32.0
ICL-NLI	-2.1	8.5	-0.1	4.6	-1.1	6.5
FT-NLI	-4.1	33.5	-3.5	29.2	-3.8	31.3

exits during productive reasoning. We therefore use $k = 1$ as the default because it provides the most favorable accuracy–compression tradeoff in this sweep.

Table 12: Redundancy Detector lookback window: k is the number of preceding reasoning steps the detector compares the current step against (PUMA default: k=1); k=all compares against every prior step. ΔAcc and TR are averaged over the five benchmarks.

	DS-7B		Nemotron-8B		Average	
k	**ΔAcc↑**	**TR↑**	**ΔAcc↑**	**TR↑**	**ΔAcc↑**	**TR↑**
1 (default)	**+2.2**	29.2	**+0.4**	16.5	**+1.3**	22.8
2	+0.0	28.1	-0.8	21.4	-0.4	24.8
4	-1.7	30.3	-3.3	23.7	-2.5	27.0
8	-0.5	32.3	-1.5	26.2	-1.0	29.3
all	-2.0	36.8	-3.8	31.1	-2.9	34.0

How far back redundant steps reach. To see how often a step repeats an earlier step other than its immediate predecessor, we take the DS-7B full CoT traces on MATH-500 and OlympiadBench and, for every step, find the nearest earlier step whose detector similarity reaches τ_{sim} (Table 13). About 30% of steps are redundant with the immediately preceding step. A larger share is redundant only with a step further back: 10.5–12.0% with a step five to eight positions back but none of the four most recent steps, and 25.5–30.1% only with a step more than eight positions back. Similarity to a non-adjacent step is therefore common, and Table 12 shows that treating it as a candidate exit raises token reduction but lowers accuracy, which is why PUMA compares each step only with its predecessor.

Table 13: How far back redundant steps reach in the DS-7B full CoT traces. For each reasoning step, we find the closest earlier step whose detector similarity with it reaches $\tau_{\text{sim}} = 0.35$ and record how many steps back it lies. Entries are percentages of all steps, and each row sums to 100. #Steps: number of reasoning steps analyzed. None: the step is not redundant with any earlier step.

		Steps back to the nearest redundant step (% of steps)					
Benchmark	**#Steps**	**1**	**2**	**3–4**	**5–8**	**>8**	**None**
MATH-500	15,275	31.8	11.3	12.1	12.0	25.5	7.3
OlympiadBench	46,923	30.0	10.2	10.4	10.5	30.1	8.8

B.5 A SEMANTIC-ENTROPY PERSPECTIVE ON REASONING CONVERGENCE

We provide a semantic-entropy perspective to explain why reasoning-step redundancy is a useful signal for convergence. Semantic Entropy (Kuhn et al., 2023; Farquhar et al., 2024) measures uncertainty in LLM outputs through semantic diversity rather than surface-form variation. We adapt this intuition from multiple sampled outputs to successive steps within a single reasoning trajectory: if recent steps continue to introduce semantically distinct content, the model is still exploring; if they collapse into repeated verification, restatement, or re-derivation, the trajectory has likely entered a locally converged phase. To formalize this view, consider a recent reasoning window $W_t^{(w)} = (r_{t-w+1}, \ldots, r_t)$ and let $c(r)$ denote the semantic cluster associated with reasoning step r. We define the *Reasoning Semantic Entropy* of this window as

$$\mathrm{RSE}_w(t) = -\sum_c p_t(c) \log p_t(c), \tag{5}$$

where $p_t(c)$ is the fraction of steps in $W_t^{(w)}$ assigned to cluster c. High $\mathrm{RSE}_w(t)$ indicates semantically diverse recent reasoning and continued exploration, whereas low $\mathrm{RSE}_w(t)$ indicates that recent steps have collapsed into a small set of redundant semantic patterns. Thus, low local RSE corresponds to local reasoning convergence.

PUMA does not explicitly compute $\mathrm{RSE}_w(t)$ online. Doing so would require clustering recent reasoning steps, for example using entailment-based semantic equivalence as in Semantic Entropy, which would add substantial overhead for deployment-time early exit. Instead, PUMA uses the Redundancy Detector as a lightweight online proxy for local semantic collapse: high similarity between recent steps suggests that the reasoning trajectory is becoming less semantically diverse and less likely to be making novel progress. This provides an interpretive lens for PUMA: the Redundancy Detector approximates the local semantic collapse that low RSE would capture, without explicitly computing semantic entropy.

Evidence that redundancy marks convergence. Several results support the view that recent steps becoming semantically redundant signals a shift from exploration to convergence. First, the property is measurable: the trained detector separates redundant from novel steps with 91.26% pairwise accuracy on held-out data (Appendix B.2), which would not be possible if "no longer adding new content" were not a separable property of reasoning steps. Second, acting on it is useful: stopping at verified redundant points reduces tokens by 20.6% while preserving accuracy (Figure 2 and Table 20) and yields the most coherent retained chains (Table 2), whereas comparing each step with more distant steps, which flags repetition that is not local, lowers accuracy (Table 12). Third, prior work points in the same direction. Li et al. (2026) show that a reasoning step's conditional mutual information with the final answer is upper-bounded by its conditional entropy, so steps that add little new information contribute little to the answer, and Wei et al. (2026) observe that reasoning moves from semantic exploration to convergence and repetitive oscillation around an instance-specific completion point.

B.6 TRIAL ANSWER INDUCTION

At each candidate stopping point, PUMA appends a *confident ending* suffix to the current reasoning prefix and prompts the model to commit to a trial answer. The suffix is task-specific:

- **Math** (AIME, MATH-500, OlympiadBench): `\n**Final Answer**\n\nThe final answer is \boxed{`
- **Multiple choice** (GPQA-Diamond): `\n**Final Answer**\n\nThe answer choice is \boxed{`
- **Code** (LiveCodeBench): `</think>\n\n### Solution Code\n```python\n`

For math and multiple-choice tasks, the suffix intentionally does not close the `</think>` tag, keeping the model in thinking mode so that it commits to an answer without switching to an explanation mode that might introduce new reasoning. For code, we close the tag explicitly because code answers require the model to generate a complete Python program in solution mode.

Confidence computation. For math tasks, we extract the token span inside `\boxed{...}` using brace matching and compute confidence as the geometric mean of token-level log-probabilities over the answer tokens (excluding braces), following DEER (Yang et al., 2026a). For GPQA, the single

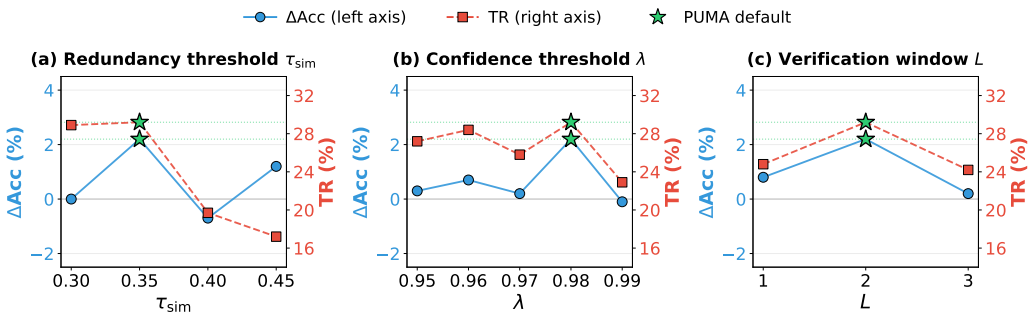


Figure 7: Sensitivity to PUMA's three core stopping hyperparameters on DS-7B, averaged over five benchmarks. Star markers and dashed horizontal lines indicate the default configuration.

answer token (A/B/C/D) competes against the full vocabulary, yielding artificially low absolute log-probabilities. We therefore apply a temperature-scaled softmax restricted to the four answer choices to obtain a calibrated confidence score. For code, we use the log-probabilities of all generated tokens, since code answers lack a `\boxed{}` structure.

Answer consistency. Trial answers across the verification window are compared via exact string match for math and multiple-choice tasks. For code, we use fuzzy matching via Python's `difflib.SequenceMatcher` (sequence similarity ≥ 0.8) to accommodate minor whitespace and formatting differences.

Generation is capped at $\sim$30 tokens for math, which is typically sufficient for a complete boxed answer. For code, we cap at $\sim$50 tokens, which rarely covers a full program but suffices to compute a meaningful confidence estimate from the opening tokens of the solution. Generating complete code trial answers could require hundreds of tokens per probe, making the overhead prohibitive for an online early-exit system.

B.7 Extended Hyperparameter Analysis

Figure 7 sweeps PUMA's three core stopping hyperparameters on DS-7B across five benchmarks. A notable finding is that PUMA is robust across a wide range of settings: out of 9 alternative configurations tested, only two produce a negative ΔAcc (-0.7 at τ_{sim}=0.40 and -0.1 at λ=0.99), and both are within 1pp of Full CoT. This robustness reflects PUMA's layered safety design: even when one hyperparameter is set suboptimally, the remaining components (Redundancy Detector, Answer Verification, Loop Breaker) continue to prevent most unsafe exits.

The redundancy threshold τ_{sim} is the most safety-critical parameter. Lowering it to 0.30 maintains token savings but eliminates the accuracy margin (ΔAcc drops from +2.2 to 0.0), while raising it to 0.40 over-suppresses the redundancy signal, lowering token reduction from 29.2% to 19.7% without recovering accuracy. The confidence threshold λ is stable across [0.95, 0.98], with only minor accuracy–efficiency variation. The sharp drop at 0.99 (-6.3pp TR) indicates that requiring near-perfect trial-answer confidence causes the verifier to veto most candidate exits. The verification window L controls how much evidence PUMA requires before stopping. $L = 1$ exits after the first verified candidate and is therefore more susceptible to single-step false positives. $L = 3$ is overly conservative: the additional verified candidate often does not appear before generation continues, reducing token reduction without improving accuracy. The default $L = 2$ provides the best balance.

The main stopping hyperparameters (τ_{sim}, λ, L) are shared across all models. The Loop Breaker consecutive-redundancy threshold m is configured per model based on validation performance on a held-out AMC23 set, as different models exhibit different redundancy patterns in late-stage reasoning. DS-R1-Distill-Qwen-7B and Nemotron-Nano-8B use m=1; DS-R1-Distill-Qwen-14B uses m=3; DS-R1-Distill-Qwen-32B uses m=4. For Qwen3-30B-A3B-Thinking, the Loop Breaker is not activated, as verified exits alone provide sufficient token savings.

Because m only controls the fallback exit, it does not affect when verified exits fire. To check that PUMA does not hinge on the per-model choice, we also apply a single m to all five models

(Table 14). The uniform settings trace a smooth accuracy–efficiency trade-off rather than breaking the method: each decrease of m by one adds 2.0 to 2.9 points of token reduction for 0.4 to 1.0 points of accuracy. With no per-model tuning at all, m=4 stays within 0.73 accuracy points of the per-model configuration, and m=3 keeps nearly the same token reduction. The per-model values, selected once on a held-out set, therefore add a modest improvement on top of a method that already works with a single setting, and m serves as a knob for users who want more or fewer savings.

Table 14: Uniform vs. per-model Loop Breaker threshold m. Each row reports the change in accuracy and token reduction relative to the per-model configuration used in the paper, averaged over all five LRMs and all five benchmarks.

	Δ vs. paper config	
Setting	**Acc**	**TR**
Per-model m (paper)	0.00	0.00
Uniform m=4	-0.73	-2.71
Uniform m=3	-1.31	-0.69
Uniform m=2	-1.72	+2.23
Uniform m=1	-2.71	+4.56

Sampling temperature. All main experiments use temperature 0.6, the recommended setting in each model's generation configuration rather than a value we tuned. To test whether the confidence gate depends on it, we regenerate the reasoning on DS-7B and MATH-500 at temperatures from 0.4 to 0.8, keeping all other settings at the paper configuration, and compare each compressed run with the uncompressed accuracy at the same temperature (Table 15). The accuracy change varies non-monotonically, with a total spread of 1.6 points (8 of 500 questions), and token reduction stays between 30.2% and 36.9%. The confidence score itself does not depend on temperature, because vLLM computes log-probabilities before temperature scaling, so λ keeps the same meaning at every temperature.

Table 15: Sensitivity to sampling temperature on DS-7B and MATH-500. Each row compares PUMA with the uncompressed accuracy at the same temperature.

Temperature	Uncompressed Acc	PUMA Acc	ΔAcc	TR
0.4	90.0	89.0	-1.0	36.9
0.5	91.0	90.2	-0.8	31.1
0.6 (default)	89.8	89.8	0.0	30.2
0.7	88.6	87.0	-1.6	32.3
0.8	90.4	89.6	-0.8	30.7

Decaying confidence threshold instead of the Loop Breaker. A smooth alternative to the Loop Breaker is to relax the confidence threshold as reasoning grows longer. We implement $\lambda(t) = \max\big(\lambda_{\min}, \lambda - \sigma(k(t - s_0))\big)$, which decays with the step count t around s_0, relax the stability tolerance correspondingly, and disable the Loop Breaker. Table 16 compares a conservative setting (s_0=75, $\lambda_{\min}$=0.92) and an aggressive one (s_0=50, $\lambda_{\min}$=0.85) with the Loop Breaker. The conservative setting is worse on both axes overall (2.11 points lower accuracy and 1.47 points lower TR), and the aggressive setting gains 3.91 points of TR at a cost of 2.42 accuracy points. The decaying threshold also adds two hyperparameters (s_0 and the steepness k) and still needs a floor $\lambda_{\min}$, which acts as a secondary confidence threshold, so we keep the Loop Breaker.

B.8 Robustness to Step Segmentation

PUMA's redundancy score is defined over segmented reasoning steps, so a natural question is whether the method depends on the specific segmentation procedure of Appendix B.1, and in particular on the availability of blank-line boundaries. We therefore hold every stopping hyperparameter (τ_{sim}, λ, ϵ, L) at its paper value, reuse the same underlying full CoT for each (model, dataset) pair, and vary only the segmenter:

Table 16: Decaying confidence threshold (DT, Loop Breaker disabled) versus the Loop Breaker. Accuracy / TR (%) are averaged over the five benchmarks; the Loop Breaker column is PUMA as reported in Table 20.

Model	DT (s_0=75, $\lambda_{\min}$=0.92)	DT (s_0=50, $\lambda_{\min}$=0.85)	Loop Breaker (PUMA)
DS-7B	58.58 / 23.68	56.94 / 35.42	60.16 / 29.18
DS-14B	63.18 / 15.66	62.93 / 17.56	64.60 / 18.10
DS-32B	68.01 / 14.90	66.87 / 18.90	69.65 / 16.50
Nemotron-8B	61.88 / 15.87	63.02 / 20.19	64.66 / 16.48
Qwen3-30B-T	79.29 / 25.61	79.63 / 30.55	82.45 / 22.80
Overall	66.19 / 19.14	65.88 / 24.52	68.30 / 20.61

(A) blank-line splitting with the merging stage of Appendix B.1 (PUMA default);
(B) fixed 120-token windows, using no blank-line information at all;
(C) dense text: all newlines are stripped from the reasoning trace before segmenting, after which fixed 120-token windows are applied as in (B).

Variant (C) is the adversarial case in which the model emits no paragraph structure whatsoever.

Table 17 reports results on two models and three benchmarks. Accuracy varies by at most about one point across the three segmenters, and the dense variant (C), with every newline removed, matches the default. Token reduction is somewhat lower for the fixed-window variants, indicating that segmentation granularity mainly affects how much compression PUMA achieves rather than whether its exits are safe. Blank lines are therefore a convenience of our implementation rather than an assumption of the method: the redundancy score is well defined over any reasonable segmentation of the trace.

Table 17: Robustness to step segmentation. (A) blank-line splitting plus merging (PUMA default); (B) fixed 120-token windows; (C) dense text with all newlines removed before applying fixed 120-token windows. All stopping hyperparameters are held at their paper values and the same underlying full CoT is reused per (model, dataset), so the segmenter is the only variable. Because this controlled setting uses a single fixed set of traces rather than the three-seed protocol of Section 4, absolute values differ slightly from Tables 1 and 20.

	(A) Default		(B) 120-tok		(C) Dense	
Benchmark	**Acc↑**	**TR↑**	**Acc↑**	**TR↑**	**Acc↑**	**TR↑**
DS-7B						
MATH-500	89.8	30.2	90.0	22.4	89.6	23.1
GPQA-D	49.5	7.3	50.0	8.2	50.0	8.7
OlympiadBench	53.9	42.7	55.0	38.2	54.8	40.3
Qwen3-32B						
MATH-500	92.0	34.3	92.0	29.4	91.6	28.3
GPQA-D	69.2	26.3	68.7	25.3	69.2	23.6
OlympiadBench	65.9	44.8	66.7	41.4	65.8	40.2

C Experimental Details and Full Results

C.1 Benchmark and Dataset Statistics

Table 18 summarizes the benchmarks used in our experiments. The main evaluation covers five challenging reasoning benchmarks spanning competition mathematics, olympiad-level STEM, and graduate-level science. For OlympiadBench, we use the standard text-only English open-ended math subset (675 problems). Generalization experiments additionally evaluate on code generation

Table 18: Dataset statistics for all evaluation benchmarks.

Benchmark	# Examples	Domain
Main reasoning benchmarks		
MATH-500 (Hendrycks et al., 2021)	500	Competition mathematics
AIME24 (Zhang & Math-AI, 2024)	30	Contest mathematics
AIME25 (Zhang & Math-AI, 2025)	30	Contest mathematics
OlympiadBench (He et al., 2024)	675	Olympiad-level math and physics
GPQA-Diamond (Rein et al., 2024)	198	Graduate-level science QA
Generalization benchmarks		
LiveCodeBench (Jain et al., 2025)	880	Code generation
MathVista (Lu et al., 2024)	200	Vision-language mathematical reasoning
MathVision (Wang et al., 2024)	200	Vision-language mathematical reasoning

(LiveCodeBench) and vision-language reasoning (MathVista and MathVision). For MathVista, we evaluate on the first 200 problems from the testmini split (1,000 problems total). For MathVision, we evaluate on the first 200 problems from the full test set (3,040 problems total).

C.2 Existing Assets and Licenses

Table 19 summarizes the main existing assets used in this work and their licenses. All assets are publicly available and cited with their original references. We use these assets for research evaluation and method development, and do not redistribute third-party model weights or benchmark datasets as part of this work.

Table 19: Licenses for main existing assets used in this work.

Asset	Type	License
DeepSeek-R1-Distill	Model	MIT
Llama-3.1-Nemotron-Nano-8B	Model	NVIDIA Open Model License
Qwen3-30B-A3B-Thinking	Model	Apache 2.0
Qwen3-32B	Model	Apache 2.0
Qwen3-Embedding-0.6B	Model	Apache 2.0
MATH-500	Dataset	MIT
AIME24/25	Dataset	Apache 2.0
OlympiadBench	Dataset	Apache 2.0
GPQA-Diamond	Dataset	CC-BY 4.0
LiveCodeBench	Dataset	MIT
MathVista	Dataset	CC-BY-SA 4.0
MathVision	Dataset	MIT
τ-bench	Dataset	MIT
vLLM	Software	Apache 2.0
MS-Swift	Software	Apache 2.0

C.3 Full Main Results Across Five Models

Table 1 in the main text reports per-benchmark results for DS-R1-Distill-Qwen-7B, and Figure 2 summarizes all five models. Table 20 reports per-benchmark results for all five models, including token counts alongside accuracy and token reduction. Figure 8 further reports the average number of reasoning steps before and after PUMA's early-exit intervention. PUMA saves 17.6–28.5% of reasoning steps across models, with deeper compression on longer chains (DS-7B: 28.5%, Qwen3-30B-T: 27.9%) and more conservative savings on shorter chains (DS-32B: 17.6%). No-Think is a prompt-only baseline that asks the model to skip explicit reasoning, but it does not impose a decoding-time stopping rule or a hard length budget. Its effectiveness therefore depends strongly

on model-specific instruction following and reasoning-format behavior. On Llama-3.1-Nemotron-Nano-8B, the No-Think prompt often fails to suppress long-form generation, leading to outputs that are longer than Full CoT and substantially less accurate. This explains the negative token reduction observed for No-Think on Nemotron and highlights a limitation of prompt-only compression: asking a reasoning model to skip thinking is not a reliable substitute for monitoring and stopping the reasoning process online.

Table 20: Per-benchmark results for all five LRMs. Acc: accuracy (%). Tok: average number of generated tokens per question. TR: token reduction relative to Full CoT (%), with negative values in red. Overall: mean over the five benchmarks; for PUMA, mean ± standard deviation over three random seeds (0, 42, 123).

	MATH-500			AIME24			AIME25			GPQA-D			OlymBench			Overall	
Method	Acc↑	Tok↓	TR↑	Acc↑	Tok↓	TR↑	Acc↑	Tok↓	TR↑	Acc↑	Tok↓	TR↑	Acc↑	Tok↓	TR↑	Acc↑	TR↑
DeepSeek-R1-Distill-Qwen-7B																	
Full CoT	90.0	4145	0.0	50.0	14280	0.0	43.3	15040	0.0	49.0	7153	0.0	57.6	8988	0.0	58.0	0.0
No-Think	79.0	830	80.0	20.0	4615	67.7	23.3	6326	57.9	28.3	904	87.3	42.8	1882	79.1	38.7	74.4
CCoT	85.0	1622	60.9	36.7	5775	59.6	33.3	5989	60.2	48.0	3178	55.6	49.2	3804	57.7	50.4	58.8
CoD	80.4	1662	59.9	40.0	6151	56.9	33.3	6214	58.7	50.5	3518	50.8	43.0	3973	55.8	49.4	56.4
Plan&Budget	84.6	2133	48.5	43.3	6722	52.9	23.3	6938	53.9	38.4	4565	36.2	49.3	4470	50.3	47.8	48.4
Ans. Conv.	61.8	974	76.5	26.7	2390	83.3	20.0	2583	82.8	34.3	544	92.4	32.6	1550	82.8	35.1	83.6
Dynasor	84.2	2878	30.6	23.3	11175	21.7	33.3	13677	9.1	39.4	2388	66.6	49.0	5923	34.1	45.9	32.4
DEER	90.6	2503	39.6	40.0	12261	14.1	50.0	13058	13.2	51.5	8398	-17.4	57.5	7171	20.2	57.9	13.9
PUMA (ours)	89.6	2546	38.6	60.0	9957	30.3	46.7	10558	29.8	49.0	6866	4.0	55.6	5107	43.2	60.2±0.9	29.2±1.6
DeepSeek-R1-Distill-Qwen-14B																	
Full CoT	91.0	3526	0.0	63.3	10209	0.0	50.0	12663	0.0	52.5	6132	0.0	61.5	7678	0.0	63.7	0.0
No-Think	80.8	1010	71.3	46.7	8689	14.9	40.0	9964	21.3	35.4	966	84.2	45.5	3348	56.4	49.7	49.6
CCoT	89.6	1771	49.8	50.0	5873	42.5	33.3	6376	49.6	57.6	3352	45.3	52.3	3970	48.3	56.6	47.1
CoD	88.8	2211	37.3	53.3	6189	39.4	30.0	6581	48.0	52.5	4009	34.6	54.8	4441	42.2	55.9	40.3
Plan&Budget	90.2	2703	23.3	43.3	6755	33.8	36.7	7059	44.3	55.0	4803	21.7	53.8	4898	36.2	55.8	31.9
Ans. Conv.	59.0	858	75.7	26.7	1947	80.9	13.3	2805	77.8	42.4	332	94.6	27.9	1117	85.5	33.9	82.9
Dynasor	89.2	3606	-2.3	56.7	10561	-3.5	53.3	13398	-5.8	58.6	4253	30.6	60.1	6805	11.4	63.6	6.1
DEER	91.2	2608	26.0	70.0	9936	2.7	50.0	12397	2.1	60.6	6918	-12.8	62.1	6950	9.5	66.8	5.5
PUMA (ours)	91.2	2776	21.3	66.7	8549	16.3	50.0	11788	6.9	55.6	5270	14.1	59.6	5229	31.9	64.6±0.8	18.1±1.3
DeepSeek-R1-Distill-Qwen-32B																	
Full CoT	89.2	2957	0.0	76.7	10011	0.0	63.3	12606	0.0	62.6	6186	0.0	59.3	7241	0.0	70.2	0.0
No-Think	84.2	1264	57.2	56.7	5781	42.2	36.7	7435	41.0	47.0	1196	80.7	49.3	4060	43.9	54.8	53.0
CCoT	87.6	1521	48.6	60.0	5600	44.1	26.7	6162	51.1	60.6	2968	52.0	53.5	3356	53.6	57.7	49.9
CoD	86.4	1558	47.3	56.7	5390	46.2	36.7	6103	51.6	64.7	3277	47.0	53.6	3724	48.6	59.6	48.1
Plan&Budget	90.8	2462	16.7	56.7	6257	37.5	40.0	6492	48.5	59.6	4335	29.9	54.1	4522	37.5	60.2	34.0
Ans. Conv.	65.6	631	78.7	16.7	1244	87.6	6.7	1638	87.0	37.9	310	95.0	30.2	889	87.7	31.4	87.2
Dynasor	92.0	3651	-23.5	53.3	10120	-1.1	50.0	12822	-1.7	58.1	4121	33.4	64.6	7050	2.6	63.6	1.9
DEER	94.2	2424	18.0	66.7	10439	-4.3	46.7	11753	6.8	67.2	6222	-0.6	63.1	6804	6.0	67.6	5.2
PUMA (ours)	88.4	2296	22.3	73.3	9046	9.6	60.0	11144	11.6	66.7	5394	12.8	59.9	5344	26.2	69.7±0.8	16.5±1.2
Llama-3.1-Nemotron-Nano-8B																	
Full CoT	93.6	3109	0.0	66.7	10463	0.0	50.0	10898	0.0	48.0	7001	0.0	63.1	6857	0.0	64.3	0.0
No-Think	62.8	7012	-125.5	26.7	13842	-32.3	16.7	15628	-43.4	25.8	7154	-2.2	31.9	11935	-74.0	32.8	-55.5
CCoT	86.6	2035	34.5	36.7	6701	36.0	20.0	6966	36.1	43.9	4824	31.1	56.9	4485	34.6	48.8	34.5
CoD	79.0	2173	30.1	50.0	6189	40.9	33.3	6610	39.4	45.0	4766	31.9	51.3	4665	32.0	51.7	34.9
Plan&Budget	89.0	2852	8.3	43.3	7022	32.9	33.3	7460	31.6	40.9	5133	26.7	53.9	5299	22.7	52.1	24.4
Ans. Conv.	55.8	735	76.3	6.7	849	91.9	3.3	1050	90.4	20.2	331	95.3	23.6	1024	85.1	21.9	87.8
Dynasor	91.4	3388	-9.0	50.0	10572	-1.0	43.3	12467	-14.4	45.0	6894	1.5	61.9	7059	-3.0	58.3	-5.2
DEER	91.8	2403	22.7	53.3	12814	-22.5	50.0	14805	-35.9	49.5	28931	-313.2	61.2	6004	12.4	61.2	-67.3
PUMA (ours)	92.6	2621	15.7	70.0	9180	12.3	50.0	8928	18.1	48.5	6349	9.3	62.2	5007	27.0	64.7±0.9	16.5±1.5
Qwen3-30B-A3B-Thinking																	
Full CoT	94.4	5206	0.0	83.3	16354	0.0	83.3	18473	0.0	72.7	7372	0.0	75.0	12668	0.0	81.7	0.0
No-Think	91.8	2516	51.7	60.0	7107	56.5	50.0	9246	50.0	73.7	8221	-11.5	68.4	5503	56.6	68.8	40.7
CCoT	89.2	2361	54.6	36.7	7217	55.9	20.0	7341	60.3	72.7	4021	45.5	53.0	5220	58.8	54.3	55.0
CoD	82.8	3652	29.9	23.3	7770	52.5	10.0	7788	57.8	65.2	5247	28.8	45.3	6297	50.3	45.3	43.9
Plan&Budget	91.8	2532	51.4	43.3	7100	56.6	36.7	7306	60.5	64.7	4635	37.1	61.3	5039	60.2	59.6	53.2
Ans. Conv.	57.0	635	87.8	0.0	921	94.4	0.0	1738	90.6	35.9	333	95.5	23.1	1050	91.7	23.2	92.0
Dynasor	94.8	5432	-4.3	86.7	14372	12.1	76.7	17959	2.8	67.7	5001	32.2	72.4	12748	-0.6	79.7	8.4
DEER	94.8	3416	34.9	83.3	14682	10.5	80.0	17181	7.3	74.8	8062	-8.2	73.5	9749	23.4	81.3	13.6
PUMA (ours)	94.2	3707	28.8	90.0	12914	21.0	80.0	15767	14.7	75.8	6058	17.8	72.3	8652	31.7	82.5±0.7	22.8±1.4

Statistical significance. Because AIME24 and AIME25 contain only 30 problems each, we test whether the differences in overall accuracy are significant (Table 21). We pair the two methods on each question and run a paired bootstrap stratified by benchmark, resampling questions within each benchmark. PUMA is not significantly less accurate than Full CoT on any of the five LRMs, and on DS-7B it is significantly more accurate ($p = 0.03$). Pooled over the five models, PUMA is significantly more accurate than DEER ($p = 0.04$), while it also removes more tokens than DEER on every model.

Table 21: Is PUMA's accuracy significantly different from Full CoT and DEER? ΔAcc: PUMA's overall accuracy minus that of the compared method, in points (from Table 20); positive values favor PUMA. p: significance of the difference, from a paired bootstrap that pairs the two methods on each question; $p < 0.05$ indicates a significant difference. The last row pools all five models.

	PUMA − Full CoT		**PUMA − DEER**	
Model	ΔAcc	p	ΔAcc	p
DS-7B	+2.2	0.03	+2.3	0.07
DS-14B	+0.9	0.21	-2.2	0.18
DS-32B	-0.5	0.84	+2.1	0.16
Nemotron-8B	+0.4	0.24	+3.5	0.06
Qwen3-30B-T	+0.8	0.07	+1.2	0.20
All five models	+0.8	0.09	+1.4	0.04

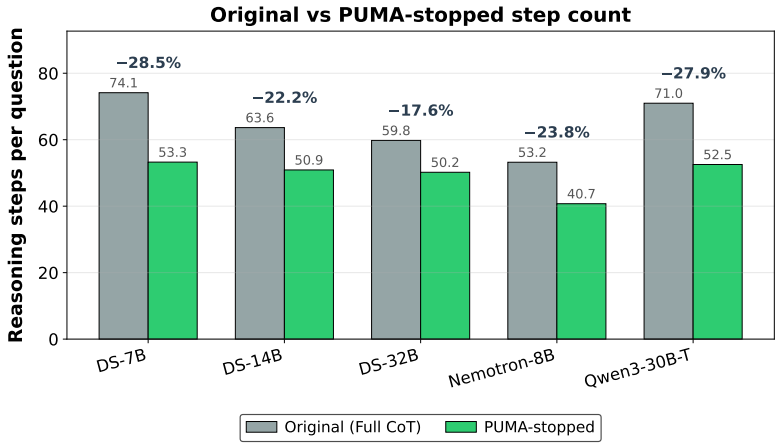


Figure 8: Reasoning-step count per question: original Full-CoT (gray) vs PUMA-stopped (green), averaged over five benchmarks. Annotated percentages are the average per-question fraction of steps saved, $(\text{orig} - \text{stopped})/\text{orig}$.

C.4 Full Component Ablation

Table 4 in the main text reports a compact ablation on DS-7B. Table 22 extends the analysis to both DS-7B and Qwen3-30B-A3B-Thinking, reports trial-answer probe statistics, and includes two additional variants: *AV only* and *w/o AC and CG*. *Probes/q* is the average number of trial-answer probes per question, and Probe× is normalized by full PUMA on the same model.

The full results provide a more detailed view of how each component affects accuracy, token reduction, and probing overhead. First, disabling the RD gate causes Answer Verification to be invoked at every eligible generated reasoning step. This increases token reduction, but also lowers accuracy and increases probe overhead substantially: 3.3× more probes on DS-7B and 4.3× more probes on Qwen3-30B-T. This shows that answer-level verification without redundancy-based candidate filtering is both less reliable and more expensive. Second, the *AV only* variant disables both the RD gate and Loop Breaker, leaving Answer Verification as the only stopping mechanism. Although it can still reduce tokens, it requires 4.0–4.3× more probes than full PUMA and does not match the full model's accuracy–efficiency balance, confirming the value of reasoning-level stopping signals. Third, removing the Loop Breaker has model-dependent effects: it sharply reduces token savings on DS-7B, while having smaller effect on Qwen3-30B-T, suggesting that different LRMs rely on the fallback to different degrees. Finally, disabling the answer-verification gates shows their role in preventing premature exits. Removing either Answer Consistency or the Confidence Gate degrades accuracy, and disabling both gates causes the largest accuracy collapse despite high token reduction. Overall, the full ablation shows that PUMA's components are complementary: the RD gate controls

Table 22: Full component ablation and probe overhead on DS-R1-Distill-Qwen-7B and Qwen3-30B-A3B-Thinking; Acc and TR are averaged over AIME24, AIME25, GPQA-Diamond, MATH-500, and OlympiadBench. *Probes/q* is the average number of trial-answer probes per question, and Probe× is normalized by full PUMA on the same model. "w/o RD Gate" disables redundancy-based candidate filtering by setting the redundancy threshold to zero, causing Answer Verification to be invoked at every step.

	DS-7B				Qwen3-30B-T			
Configuration	**Acc↑**	**TR↑**	**Probes/q↓**	**Probe× ↓**	**Acc↑**	**TR↑**	**Probes/q↓**	**Probe× ↓**
PUMA (full)	**60.2**	29.2	13.75	1.0×	**82.5**	22.8	10.46	1.0×
w/o RD Gate	56.1 (-4.1)	36.2	45.69	3.3×	78.6 (-3.9)	34.4	45.11	4.3×
w/o Loop Breaker	59.2 (-1.0)	17.5	17.48	1.3×	80.5 (-2.0)	23.4	11.88	1.1×
AV only (w/o RD and LB)	58.3 (-1.9)	28.5	55.27	4.0×	77.6 (-4.9)	34.5	45.13	4.3×
w/o Answer Consistency	57.3 (-2.9)	25.7	15.96	1.2×	79.9 (-2.6)	27.9	9.04	0.9×
w/o Confidence Gate	53.7 (-6.5)	49.7	8.64	0.6×	72.6 (-9.9)	44.5	5.11	0.5×
w/o AC and CG	44.0 (-16.2)	69.5	4.55	0.3×	61.9 (-20.6)	57.7	2.95	0.3×

when verification is invoked, Answer Verification filters unreliable candidates, and the Loop Breaker provides additional savings when reasoning enters sustained redundancy.

C.5 Detailed Latency Analysis

Table 23: Per-benchmark wall-clock latency across three models. For each benchmark we report accuracy (Acc, %, ↑), token reduction (TR, %, ↑), and average wall-clock seconds per question (s/q, ↓). The latency study covers four benchmarks (GPQA-Diamond is not included). Overall Acc and TR are averaged over these four benchmarks; overall s/q and Speedup are computed from total wall-clock time, with Speedup relative to Full CoT. Speedup below 1× (in red) indicates slower wall-clock performance than Full CoT. For the s/q columns, the **best** (lowest) and second-best values within each model are highlighted.

	MATH-500			AIME24			AIME25			OlympiadBench			Overall			
Method	Acc↑	TR↑	s/q↓	Acc↑	TR↑	s/q↓	Acc↑	TR↑	s/q↓	Acc↑	TR↑	s/q↓	Acc↑	TR↑	s/q↓	Speedup↑
DeepSeek-R1-Distill-Qwen-7B																
Full CoT	90.0	0.0	0.98	50.0	0.0	10.18	43.3	0.0	10.56	57.6	0.0	2.57	60.2	0.0	2.30	1.00×
DEER	90.6	39.6	1.63	40.0	14.1	12.64	50.0	13.2	12.80	57.5	20.2	3.78	59.5	21.8	3.35	0.69×
Dynasor	84.2	30.6	1.03	23.3	21.7	11.86	33.3	9.1	15.26	49.0	34.1	1.84	47.5	23.9	2.08	1.11×
PUMA (ours)	89.6	38.6	**0.87**	60.0	30.3	**9.41**	46.7	29.8	**6.88**	55.6	43.2	**1.64**	63.0	35.5	**1.64**	**1.40×**
DeepSeek-R1-Distill-Qwen-14B																
Full CoT	91.0	0.0	2.09	63.3	0.0	20.20	50.0	0.0	20.95	61.5	0.0	7.32	66.5	0.0	5.84	1.00×
DEER	91.2	26.0	3.68	70.0	2.7	22.21	50.0	2.1	23.88	62.1	9.5	10.36	68.3	10.1	8.27	0.71×
Dynasor	89.2	-2.3	8.59	56.7	-3.5	**12.54**	53.3	-5.8	25.09	60.1	11.4	42.96	64.8	-0.1	27.87	0.21×
PUMA (ours)	91.2	21.3	**1.84**	66.7	16.3	15.62	50.0	6.9	**20.92**	59.6	31.9	**5.39**	66.9	19.1	**4.58**	**1.28×**
Llama-3.1-Nemotron-Nano-8B																
Full CoT	93.6	0.0	1.07	66.7	0.0	**7.68**	50.0	0.0	**9.82**	63.1	0.0	2.83	68.3	0.0	2.41	1.00×
DEER	91.8	22.7	1.83	53.3	-22.5	23.10	50.0	-35.9	26.61	61.2	12.4	4.99	64.1	-5.8	4.67	0.52×
Dynasor	91.4	-9.0	0.99	50.0	-1.0	9.17	43.3	-14.4	11.37	61.9	-3.0	24.32	61.7	-6.9	14.19	0.17×
PUMA (ours)	92.6	15.7	**0.85**	70.0	12.3	8.71	50.0	18.1	11.31	62.2	27.0	**2.30**	68.7	18.3	**2.09**	**1.15×**

Figure 3 in the main text reports average speedup across benchmarks. Table 23 provides per-benchmark wall-clock latency for three models. PUMA achieves consistent speedup across all settings (1.15–1.40×), while DEER is uniformly slower than Full CoT (0.52–0.71×) and Dynasor ranges from 1.11× to 0.17× depending on model size. These results show that token reduction alone does not guarantee wall-clock gains. Both DEER and Dynasor can reduce generated tokens yet still run slower than Full CoT, because frequent trial-answer probing introduces additional forward-pass overhead. When this overhead outweighs the time saved from shorter reasoning traces, positive token reduction does not translate into end-to-end speedup.

Why probing costs more time than its token count suggests. All methods, including our reimplementations of DEER and Dynasor that follow their official algorithms, run on the same serving stack (vLLM 0.15.0 with the V1 engine) with identical settings, including prefix caching and chunked

prefill. For online early exit, the dominant cost is how often and how coarsely decoding is interrupted, not the number of probe tokens. Under continuous batching, Full CoT issues a single generation call in which every sequence finishes independently at full throughput. An early-exit method instead pauses decoding at each trigger point to run a probe, and each pause acts as a synchronization barrier whose length is set by the slowest sequence in the batch. On MATH-500, for example, DEER generates 26–40% fewer tokens than Full CoT on DS-7B and DS-14B yet runs 66–76% slower, and Dynasor probes at every chunk boundary, reaching 55–66 probes per question on these two models; on DS-14B OlympiadBench, Dynasor takes 28,995 s in total versus 3,635 s for PUMA. PUMA avoids this regime because the Redundancy Detector gates interruptions: decoding proceeds in long uninterrupted blocks, which account for 76–83% of PUMA's runtime, and probes are issued in batches only at detector-flagged checkpoints, each taking 22–81 ms on average across the five models.

C.6 Token Accounting

PUMA's generated tokens consist of the retained reasoning, the answer tokens of each trial-answer probe, and the final answer. Table 24 breaks them down by source, averaged over the five benchmarks. Trial-answer probes account for only 0.2–0.7% of all tokens: each probe is capped at about 30 tokens for math (Appendix B.6), and the Redundancy Detector gate keeps the number of probes to 7.7–13.8 per question. This is consistent with the small probing overhead in wall-clock time (Figure 3c).

Table 24: Breakdown of PUMA's generated tokens (%) by source, averaged over the five benchmarks, and the average number of trial-answer probes per question.

Model	Reasoning	Probes	Final answer	Probes/q
DS-7B	91.9	0.7	7.3	13.75
DS-14B	91.7	0.6	7.7	13.67
DS-32B	91.8	0.5	7.7	11.61
Nemotron-8B	91.1	0.2	8.7	7.66
Qwen3-30B-T	90.8	0.4	8.8	10.46

C.7 Agent Setting: τ-bench

To test PUMA in an agent setting, we evaluate it on τ-bench (Yao et al., 2024), a tool-agent-user benchmark (retail: 115 tasks; airline: 50 tasks). Each agent turn's `<think>` block is treated as one PUMA problem, with the turn's next action as the answer on which PUMA verifies convergence, and the detector is used zero-shot. The agent model is Qwen3-32B served with vLLM with greedy decoding, and the user simulator is DeepSeek-V4-Pro (DeepSeek-AI et al., 2026) at temperature 0, held fixed across all methods and both domains. Table 25 shows that PUMA is the only method that shortens reasoning without lowering task success in either domain, whereas DEER and Dynasor lose 5 to 14 points. To estimate how much agent reasoning can be removed without changing the agent's behavior, we also compute an oracle: for each turn, the shortest reasoning prefix from which the model regenerates exactly the same action as with full reasoning. This oracle removes about 32% of agent reasoning in both domains (32.6% on retail and 31.7% on airline), confirming that agent traces contain substantial redundant reasoning; PUMA's 27% reduction on retail reaches most of this bound without knowing the final action in advance.

Table 25: Agent setting on τ-bench with Qwen3-32B: task success (%) and reduction of reasoning tokens (%). Parentheses give the change in task success relative to Full CoT.

	Retail		Airline	
Method	**Success**	**Reduction**	**Success**	**Reduction**
Full CoT	39.1	–	42.0	–
PUMA (ours)	43.5 (+4.4)	27	42.0 (0.0)	17
DEER	30.4 (-8.7)	34	28.0 (-14.0)	42
Dynasor	33.9 (-5.2)	22	34.0 (-8.0)	35

D Additional Baseline Experiments

D.1 Budget Tuning Does Not Rescue Prompt-Based Baselines

A natural question is whether the accuracy gap between PUMA and prompt-based baselines can be closed by relaxing their word-budget instructions. We therefore sweep the word budgets specified in the prompts for CCoT and CoD on DS-7B LiveCodeBench. We note that LRMs do not always strictly follow these requested budgets; thus, the reported token reduction reflects the actual generated outputs rather than the nominal prompt budget. Table 26 shows that budget tuning does not close the gap: CCoT remains in the 42.7–45.8% accuracy range across requested budgets from 30 to 300 words, and CoD remains in the 43.2–44.9% range across requested budgets from 5 to 30 words per step, both below PUMA's 50.3%.

Table 26: Budget-sweep analysis of prompt-based baselines on DS-7B LiveCodeBench. Relaxing CCoT's global word budget or CoD's per-step word budget does not close the accuracy gap to PUMA. Δ is accuracy change relative to Full CoT. "def." denotes the default budget used in the main experiments.

	Reference		CCoT: global word budget					CoD: per-step word budget		
Metric	**Full CoT**	**PUMA**	**30**	**45 (def.)**	**75**	**150**	**300**	**5 (def.)**	**10**	**30**
Acc↑	51.82	**50.34**	45.57	45.80	42.73	44.20	44.32	43.18	43.52	44.89
Δ↑	—	**-1.48**	-6.25	-6.02	-9.09	-7.62	-7.50	-8.64	-8.30	-6.93
TR↑	0.0	**19.0**	42.9	42.7	41.8	41.4	40.0	40.5	40.6	41.3

D.2 Comparison with a Token-Level Method: RCPD

Our main baselines neither modify model weights nor train model-specific probes (Section 4). Token-level methods also avoid both, but they read decoding signals tied to a specific model's vocabulary and termination token, so their thresholds must be recalibrated for each model. Among them, we compare with RCPD (Wei et al., 2026), which is the closest to PUMA in motivation because it also builds on the observation that reasoning moves from exploration to convergence. RCPD monitors the rank of the `</think>` token and exits when one of its threshold rules fires. No official implementation is available, so we reimplement it following the paper, and its published thresholds produce no early exits on DS-7B. We therefore calibrate RCPD on the same held-out split used for PUMA, while PUMA keeps its paper configuration, and run the comparison on DS-7B, the model used for the per-benchmark results in Table 1 and for our ablations.

Table 27 reports the results, with TR counting all tokens each method generates, including probes. PUMA preserves or improves accuracy on four of the five benchmarks, whereas RCPD loses accuracy on four of the five at comparable or lower token reduction. On OlympiadBench, PUMA removes 16 percentage points more tokens while losing about half as much accuracy, and on MATH-500 the two reach similar reduction but PUMA is 2.5 points more accurate. With more aggressive thresholds, RCPD reaches a reduction comparable to that reported by its authors (45–46% on AIME), but loses 10–13 accuracy points there. Even after per-model calibration, the token-level signal therefore does not match the accuracy–efficiency balance that PUMA reaches with a detector trained once and shared across models.

E Reasoning Quality Evaluation Details

We evaluate the quality of retained reasoning chains using an LLM-as-Judge protocol. The goal is to measure whether an early-exit method preserves a readable and sufficient reasoning chain, beyond merely preserving final-answer accuracy.

Judge model and protocol. We use GPT-5.4-thinking as the judge via the OpenAI Batch API. Each instance contains the original question and the retained reasoning chain produced by the evaluated method. The judge is instructed to first provide a brief rationale and then assign scores, following standard LLM-as-Judge practice (Li et al., 2025a). The judge prompt template is summarized in Table 28.

Table 27: Comparison with RCPD on DS-7B. Both methods are applied to a single fixed set of full CoT traces, so absolute values differ slightly from Table 1. Parentheses give the accuracy change relative to Full CoT.

	Full CoT	PUMA		RCPD (calibrated)	
Benchmark	**Acc**	**Acc**	**TR**	**Acc**	**TR**
MATH-500	89.8	89.8 (0.0)	30.2	87.3 (-2.5)	27.0
GPQA-D	49.0	49.5 (+0.5)	7.3	48.0 (-1.0)	8.9
OlympiadBench	55.4	53.9 (-1.5)	42.7	52.6 (-2.8)	26.3
AIME24	43.3	50.0 (+6.7)	27.1	43.3 (0.0)	28.0
AIME25	43.3	43.3 (0.0)	21.0	33.3 (-10.0)	22.8

Scope. We evaluate three (model, benchmark) combinations: DS-R1-Distill-Qwen-7B on GPQA-Diamond (198 questions), Nemotron-Nano-8B on GPQA-Diamond (198 questions), and Qwen3-30B-A3B-Thinking on MATH-500 (500 questions). All questions are evaluated regardless of answer correctness, avoiding subset-selection bias. The judge is not given gold answers, method names, or answer-correctness labels. It evaluates each retained reasoning chain as written, including whether the chain provides sufficient and coherent support for its own stated final answer. Since LLM-as-Judge evaluation over long reasoning chains is substantially more expensive than answer-only evaluation, we use these representative combinations rather than the full model–benchmark grid. The reported reasoning-quality evaluation costs approximately $300 in OpenAI API usage.

Rubric. Each reasoning chain is scored on a 10–100 scale, in increments of 10, along four dimensions:

- **Completeness**: whether the chain contains a semantically sufficient derivation from the problem statement to the final answer.
- **Coherence**: whether the chain forms a smooth and logically connected line of reasoning, without abrupt jumps, contradictions, or disruptive interruptions.
- **Conciseness**: whether the chain avoids unnecessary repetition, redundant verification, and non-productive loops.
- **Justification**: whether a reader can understand why the final answer follows from the main derivation.

Few-shot calibration. We include six manually written in-context examples covering distinct quality patterns: complete and concise, complete but redundant, brief but sufficient, brief and incomplete, derivation with a corrected false start, and unresolved contradiction. All examples are based on a single math problem to avoid confounding rubric calibration with problem difficulty.

Evaluator bias control. To mitigate evaluator bias, the judge is not shown method names and evaluates all retained reasoning chains using the same rubric. The judge model is not used to train PUMA's Redundancy Detector or to select stopping hyperparameters. Although the detector supervision uses LLM-generated redundancy annotations, it is constructed at the reasoning-step-pair level, whereas the judge evaluates complete retained reasoning chains; the two stages use different inputs, prompts, and objectives. To further validate judge reliability, we randomly sample 100 pairwise comparisons from the evaluation results. Each comparison contains two anonymized retained reasoning chains for the same question, and a human annotator independently selects the chain with higher overall explanation quality. The human judgments agree with GPT-5.4-thinking's score-induced preferences in 85% of cases, suggesting that the judge-based relative comparisons are broadly aligned with human assessment. The absolute scores are modest across all methods, with the best average score in the mid-50s. This is because the rubric evaluates the quality of the retained reasoning chain, rather than final-answer correctness alone: a chain must be complete, coherent, concise, and sufficiently justified to receive a high score. A correct final answer is therefore not sufficient if the retained reasoning is incomplete, hard to follow, overly repetitive, or weakly justified. We therefore focus on relative comparisons across methods under the same anonymized judge and rubric.

Table 28: Judge prompt template for reasoning-chain quality evaluation. The full prompt includes per-dimension anchors and six calibration examples.

Reasoning-chain quality judge

Role	You are an expert evaluator of reasoning chains. Your task is to evaluate how well a reasoning chain communicates a sufficient, productive derivation of the stated answer.
Principle	Reward one clear successful line of reasoning. Do not reward extra exploratory branches, repeated verification, generic tutoring preambles, or exhaustive enumeration once the answer is already derivable.
Scoring	Score each chain from 10 to 100, in steps of 10, on Completeness, Coherence, Conciseness, and Answer Justification.
Constraints	Evaluate the reasoning chain as written. If the stated final answer is unsupported, contradicted, or does not follow from the reasoning, reflect this in Completeness and Answer Justification. Score Coherence and Conciseness independently. Do not use method names, gold answers, or external knowledge. Do not fill in missing proof steps with your own solution.
Rubric	Completeness measures whether the chain contains a sufficient derivation from the problem to the stated final answer. Coherence measures whether the reasoning flow is logically connected, without abrupt jumps, contradictions, or unresolved self-corrections. Conciseness measures whether the chain avoids repetition, redundant verification, and non-productive loops. Justification measures whether a reader can understand why the stated final answer follows from the main derivation.
Examples	*[Six calibration examples covering concise sufficient reasoning, complete but redundant reasoning, brief but incomplete reasoning, false starts, and unresolved contradictions.]*
Input	Question: `<QUESTION>` Reasoning chain: `<RETAINED REASONING CHAIN>`
Output	Return only a JSON object with a brief rationale and four scores:

```
{
  "brief_rationale": "<1-2 SENTENCES>",
  "completeness": <10-100>,
  "coherence": <10-100>,
  "conciseness": <10-100>,
  "justification": <10-100>
}
```

F Internalizing PUMA into Model Weights: Implementation Details

This appendix provides full details for the internalization experiments reported in Section 6.2. All variants fine-tune DS-R1-Distill-Qwen-7B with LoRA (rank 64, alpha 128, applied to all linear layers) and are evaluated with pure vLLM inference, without any RD or AV at test time. Our RL training design is partly inspired by DEER (Yang et al., 2026a).

Training data. We use the 12K problems from the Lightman split of MATH-benchmark (Hendrycks et al., 2021), which is mathematics-only; consequently, MATH-500 performance can be regarded as in-distribution behavior, while the resulting models generalize zero-shot to AIME24 (harder math) and GPQA-D (cross-domain science). We verify that no problem in this train split overlaps with the MATH-500 test split, ensuring zero contamination of our in-distribution evaluation. For each problem, we generate one Full CoT trajectory from the base model and run the PUMA inference pipeline to record RD-flagged candidate exit positions and the verified stopping point.

Per-variant data construction. PUMA-SFT keeps rows $(Q, R_{\leq t^*}, A)$ where PUMA verifies an early exit at t^* and the regenerated answer is correct, filtered to aggressive compressions ($t^*/|R| < 0.6$), yielding $\sim$6.5K examples. FixedExit-SFT replaces t^* with the earliest fixed-interval position ($K{=}3$) yielding a correct forced-stop answer; filtering is identical. PUMA-DPO pairs PUMA-truncated

Table 29: Internalizing PUMA's stopping signal with LoRA on DS-R1-Distill-Qwen-7B. PUMA-selected exit positions supervise SFT, DPO, or GRPO; all trained variants use pure vLLM inference without PUMA modules.

	MATH-500		AIME24		GPQA-D		Avg.	
Method	Acc↑	TR↑	Acc↑	TR↑	Acc↑	TR↑	Acc↑	TR↑
Full CoT (base)	90.0	0.0	50.0	0.0	49.0	0.0	63.0	0.0
PUMA (inference)	89.6	38.6	60.0	30.3	49.0	4.0	66.2	24.3
Imitation learning (SFT)								
Standard-SFT	91.4	-4.5	43.3	-31.0	49.5	-61.9	61.4	-32.5
FixedExit-SFT	85.8	46.3	36.7	17.2	39.9	21.3	54.1	28.3
PUMA-SFT	91.4	21.9	53.3	32.7	56.1	3.3	**66.9**	19.3
Preference learning (DPO)								
PUMA-DPO	90.6	39.3	56.7	59.4	45.0	47.7	64.1	**48.8**
Reinforcement learning (GRPO)								
Standard-GRPO	91.4	1.2	50.0	-1.5	56.1	-9.8	65.8	-3.4
FixedExit-RL	91.4	41.5	46.7	38.0	51.5	32.8	63.2	37.4
PUMA-RL	90.2	42.4	56.7	35.7	54.0	26.6	**67.0**	34.9

chains (chosen) with Full CoT chains (rejected) when both are correct (∼5.8K pairs). PUMA-RL and FixedExit-RL use a pre-expanded GRPO dataset where each row is a question concatenated with a reasoning prefix and the closing ⟨/think⟩ token, with the prefix truncated at a PUMA-flagged or fixed-interval position; the model is trained to generate only the answer phase that follows (∼15K samples each). Standard-SFT and Standard-GRPO use no exit-position signal: Standard-SFT trains on the base model's untruncated Full CoT trajectories, and Standard-GRPO performs standard GRPO on $(Q, \text{ground truth})$ pairs without any prefix conditioning, generating the entire ⟨think⟩ . . . ⟨/think⟩ chain from scratch at each rollout.

Training. SFT runs for 3 epochs with learning rate 2×10^{-4}, batch size 1 with gradient accumulation 16, and a maximum sequence length of 16,384 tokens. DPO runs for 1 epoch with the same learning rate and the default β; we add an SFT loss on the chosen response with weight 0.1, which we found necessary to prevent drift. GRPO uses learning rate 1×10^{-5} and four rollouts per group, generated with vLLM on the training GPUs. PUMA-RL and FixedExit-RL run for 1,500 steps with a maximum completion length of 4,096 tokens; Standard-GRPO, which generates the full reasoning, runs for 400 steps with a maximum completion length of 8,192 tokens. For Standard-GRPO, the reward is pure answer correctness. For PUMA-RL and FixedExit-RL, which generate only the answer phase after a provided reasoning prefix, we use the following reward:

$$R = r_{\text{correct}} \cdot \big(1.0 + 0.5\ \max(0,\ 1 - \ell/4096)\big) + r_{\text{rank}}, \tag{6}$$

where $r_{\text{correct}} \in \{0, 1\}$ indicates answer correctness, ℓ is the length of the completion in characters, and $r_{\text{rank}} \in \{+0.5, +0.25, 0, -0.25\}$ is a within-group rank bonus favoring the shortest correct trajectory among the four rollouts. All training uses bfloat16 with gradient checkpointing on 4×NVIDIA GH200, implemented in ms-swift (Zhao et al., 2024).

Why training on the answer shortens the reasoning. PUMA-RL shortens the model's reasoning even though it never trains on reasoning tokens. Each training example gives the model a question together with a reasoning prefix that PUMA has already cut at a selected exit and closed with `</think>`. The model only generates the answer that follows, and GRPO rewards answers that are correct and short. At test time, the trained model writes its whole output, reasoning included, and ends its reasoning much earlier than the base model. Almost all of the saved tokens come from the reasoning: the answer after `</think>` is no longer than the base model's, so the model does not simply move its reasoning into the answer.

Two baselines show where this behavior comes from. Standard-GRPO is trained with a correctness reward on rollouts in which the model writes its own full reasoning, and it does not shorten the output (Table 29). The shortening therefore comes from training on reasoning that has already been cut, not from reinforcement learning on the same problems. Because the LoRA update is learned only in states where the reasoning has just ended at a sufficient point, we hypothesize that it shifts the model toward treating its own reasoning as complete earlier. FixedExit-RL is trained in the same way but on prefixes cut at fixed intervals; it also shortens its reasoning but is less accurate (63.2% vs. 67.0%).

What the model learns therefore depends on where the training prefixes end, not only on how short they are, which is why PUMA-selected exits provide better supervision.

G Case Studies

We present two examples from DS-7B to illustrate how PUMA's exits differ from stopping on an answer-level signal alone. Tables 30 and 31 quote the reasoning steps around each candidate exit. For each candidate step t, we report the detector similarity between steps $t+1$ and t that raised it, the trial answer probed from the prefix ending at step t, its confidence, and the outcome of Answer Verification ($\lambda = 0.98$, $\epsilon = 0.03$, $L = 2$). Steps outside these excerpts are summarized in italics.

Case 1: a confident answer before the derivation is finished. At step 14 the model has expanded $(\sin x)^7$ but has not yet matched the expansion against the target form, and its trial answer $-35/64$ already has confidence 0.982, above λ. Stopping on confidence alone at this point would return the wrong sign. PUMA requires the next candidate to give the same answer; step 15 reads off $d = 35/64$, so generation continues, and PUMA exits after step 16 once two candidates agree. PUMA uses 2,388 tokens, while the full CoT spends 14,925 tokens to reach the same answer.

Table 30: Case 1 (MATH-500, DS-7B): reasoning steps around PUMA's candidate exits.

Trace excerpt

Problem: There exist constants a, b, c, and d such that $(\sin x)^7 = a \sin 7x + b \sin 5x + c \sin 3x + d \sin x$ for all angles x. Find d. (Answer: $35/64$.)

Full CoT: 120 steps, 14,925 tokens, final answer $35/64$. PUMA: exits after step 16, 2,388 tokens, final answer $35/64$.

[Steps 1–13 summarized] The model restates the problem and expands $(\sin x)^7$ with complex exponentials and the binomial theorem, arriving at $-\frac{1}{64}\sin 7x + \frac{7}{64}\sin 5x - \frac{21}{64}\sin 3x + \frac{35}{64}\sin x$. Candidates after steps 1 and 2 have confidence 0.735 and 0.717, below λ. From step 3 to step 14, consecutive steps have similarity 0.10–0.33, so no candidate is raised while the expansion is computed.

Step 14 — *similarity of step 15 to step 14: 0.44*

But wait, according to the problem statement, $(\sin x)^7$ is equal to $a \sin 7x + b \sin 5x + c \sin 3x + d \sin x$. Comparing the two expressions:

Trial answer $-35/64$, confidence 0.982 — *above λ; opens a verification window*

Step 15 — *similarity of step 16 to step 15: 0.51*

$(\sin x)^7 = \left(-\frac{1}{64}\sin 7x + \frac{7}{64}\sin 5x - \frac{21}{64}\sin 3x + \frac{35}{64}\sin x\right)$. Therefore, the coefficients are: $a = -1/64$, $b = 7/64$, $c = -21/64$, $d = 35/64$. So, the value of d is $35/64$.

Trial answer $35/64$, confidence 1.000 — *differs from step 14, window fails; opens a new window*

Step 16 — *similarity of step 17 to step 16: 0.41*

Wait, let me double-check my steps to make sure I didn't make any errors. Especially, when I simplified the complex exponentials to sine terms, I had to factor out the i and adjust the signs accordingly. Let me go through that part again. [...]

Trial answer $35/64$, confidence 1.000 — *same answer as step 15, stable confidence*

PUMA exits here. *[Steps 17–120 of the full CoT summarized] The model re-derives the expansion and then checks it by substituting specific values of x. One of these checks is set up incorrectly and later corrected (step 119: "my system of equations was wrong because I incorrectly thought that plugging $x = \pi/4$ would give 1/64, but in reality, it gives 1/8"). The chain ends with $35/64$.*

Case 2: stopping before the model talks itself out of a correct answer. Here every trial answer in the first 54 steps is the correct 1003. The confidence first exceeds λ at step 15, the next candidate at step 18 agrees, and PUMA exits with 2,508 tokens. The full CoT continues for 107 more steps. At steps 54–55 it adopts the incorrect claim that every triangle in the triangulation has two polygon sides, its trial answers then alternate between 1003 and 2004, and it ends with 2004 after 11,104 tokens. By

exiting once the answer has stabilized, PUMA uses 77% fewer tokens and also avoids this later error, the pattern behind the $W \to R$ transitions in Figure 4.

Table 31: Case 2 (OlympiadBench, DS-7B): reasoning steps around PUMA's exit and the later error in the full CoT.

Trace excerpt

Problem: A diagonal of a regular 2006-gon is called odd if its endpoints divide the boundary into two parts, each composed of an odd number of sides. Sides are also regarded as odd diagonals. Suppose the 2006-gon has been dissected into triangles by 2003 nonintersecting diagonals. Find the maximum possible number of isosceles triangles with two odd sides. (Answer: 1003.)

Full CoT: 125 steps, 11,104 tokens, final answer 2004 (wrong). PUMA: exits after step 18, 2,508 tokens, final answer 1003.

[Steps 1–14 summarized] The model works out which diagonals are odd. Nine candidates are raised; every trial answer is 1003, with confidence rising from 0.879 to 0.979, below λ.

Step 15 *similarity of step 16 to step 15: 0.38*

Therefore, sides are odd diagonals, and any diagonal that connects two vertices with an odd number of edges along one arc is also an odd diagonal. So, a side is an odd diagonal because it splits the polygon into 1 and 2005 sides, but 1 is odd and 2005 is odd, so that's why sides are considered odd diagonals.

Trial answer 1003, confidence 0.984 *above λ; opens a verification window*

Step 18 *similarity of step 19 to step 18: 0.53*

Wait, maybe another approach. Let's suppose n is even, which it is (2006). Then, for a diagonal to split the polygon into two arcs each with an odd number of sides, both arcs must have an odd number of sides, meaning that the number of sides between the two vertices is odd on both arcs.

Trial answer 1003, confidence 0.988 *same answer as step 15, stable confidence*

PUMA exits here. The remaining steps are from the full CoT.

Steps 54–55 *trial answer changes to 2004*

[. . .] Each triangle in a convex polygon triangulation has exactly two sides of the polygon, so all triangles are isosceles with two polygon edges. Therefore, in the case of a regular polygon, all triangles in a triangulation are isosceles triangles with two polygon edges, which are odd diagonals. Therefore, if all triangles in the triangulation are isosceles with two odd sides, then the number of such triangles would be 2004.

[Steps 56–125 summarized] The trial answers alternate between 1003 and 2004, and the chain ends with "Therefore, I think the answer is 2004."

H LIMITATIONS

PUMA relies on step-level reasoning traces and may be less effective when model outputs are very short; segmentation granularity mainly affects how much it compresses rather than its accuracy (Appendix B.8). Because PUMA stops conservatively, its savings are smaller when chains contain little redundancy, as on GPQA-Diamond (4.0–17.8% token reduction). Although the Redundancy Detector transfers beyond text-only QA, it is primarily trained on text reasoning traces, and our evaluation covers mathematical, scientific, code, vision-language, and agent tasks, so other domains may require additional calibration. Our internalization experiments are also limited to one base model and math-focused training data, leaving larger-scale and cross-domain learned stopping policies for future work.