MICROMOUSE: MASKING HARMFUL SUPERVISION IN LONG COT VIA RELATIVE POTENTIAL

Anonymous authors
Paper under double-blind review

ABSTRACT

Long chain-of-thought (CoT) trajectories have become an important source of supervision for training large reasoning models, yet conventional supervised fine-tuning (SFT) typically optimizes all tokens in these trajectories indiscriminately. However, even successful long CoTs may contain harmful supervision, motivating the need for methods that can automatically identify and mask such signals during SFT. Inspired by the *MicroMouse* maze-solving process, we view a long CoT as a search trajectory from the input problem toward the target answer. This perspective motivates maintaining a goal-relative measure, analogous to a MicroMouse's spatial coordinates, to quantify how close the current reasoning state is to the target answer. Building on this perspective, we propose MICROMOUSE, an automated selective-SFT framework for masking harmful supervision in long CoTs. In MICROMOUSE we introduce a *Relative Potential* (REP) to measure how closely each reasoning state supports the gold answer. Unlike the perplexity of the gold answer alone, REP measures relative support by contrasting the gold answer with a matched foil. This contrast cancels likelihood variations shared by the pair, yielding a more stable signal of progress toward the target answer. By tracking the REP trajectory, MICROMOUSE identifies consecutive potential decreases as erroneous exploration and post-peak reasoning as redundant wandering, and masks these harmful supervision during SFT. Extensive experiments across multiple student model scales and reasoning benchmarks show that MICROMOUSE consistently outperforms Full-CoT SFT and existing selective-SFT baselines, yielding average relative gains of **5.5%** in greedy accuracy and **2.6%** in Pass@1 relative to Full-CoT SFT. Meanwhile, MICROMOUSE improves reasoning efficiency by reducing average output length by **8.8%**, while identifying harmful supervision in only **1.7** seconds per long CoT on average.

1 INTRODUCTION

Recent large reasoning models (LRMs) have demonstrated strong performance on complex reasoning tasks by generating extended chains of thought (CoTs) and leveraging increased inference-time computation (OpenAI, 2024; Guo et al., 2025). CoTs enable models to solve complex problems through verbalized intermediate reasoning steps, substantially improving performance across reasoning-intensive tasks (Wei et al., 2022). Long CoTs have increasingly become an important source of supervision for supervised fine-tuning (SFT), where student models learn reasoning behaviors from high-quality reasoning trajectories (Ye et al., 2025).

However, prevailing full-CoT SFT optimizes all tokens in a reasoning trajectory uniformly, regardless of their actual learning value. This treatment can be suboptimal for long-form CoTs, as even successful trajectories may contain harmful supervision such as erroneous exploration and redundant wandering. Indiscriminately learning from these steps can reinforce undesirable reasoning patterns, thereby degrading both reasoning performance and inference efficiency (Chen et al., 2025a; Cui et al., 2025; Li et al., 2026). This motivates an automated framework that identifies such harmful supervision and selectively masks its contribution to the SFT objective.

As shown in Figure 1 (left), our motivation is inspired by the *MicroMouse* maze-solving problem (Wyard-Scott & Meng, 1995). During the search phase, a MicroMouse explores an unknown maze through a sequence of movements, where erroneous forward and backtracking decisions (i.e., pass-

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

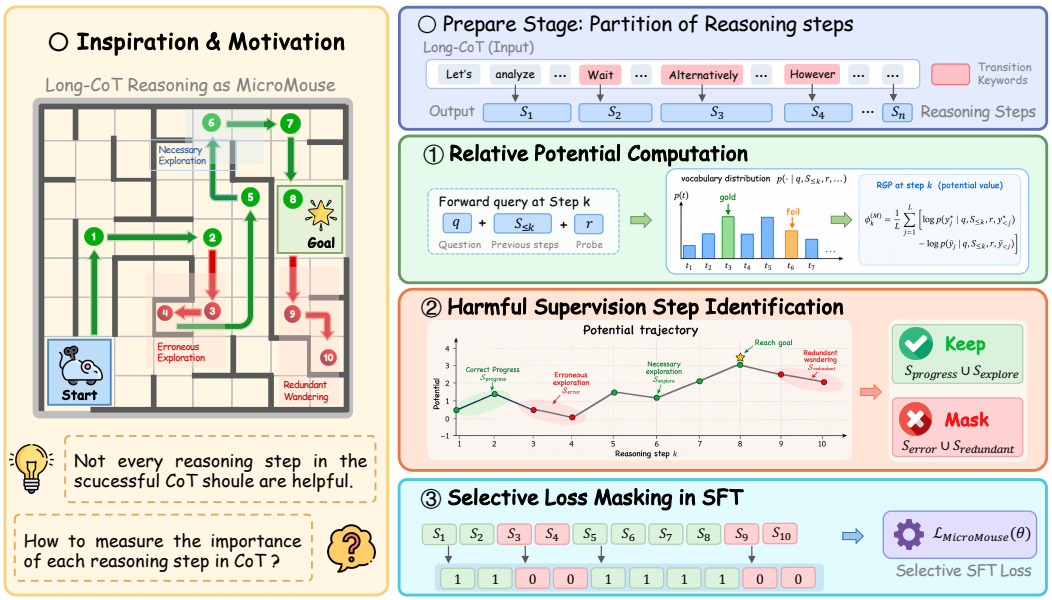

Figure 1: **Overview of MicroMouse**, a framework consisting of three stages.

ing the goal and returning) may occur. During the dash phase, it instead follows the computed fastest route to the destination. We draw an analogy between this search trajectory and a long CoT: reasoning steps may likewise contain erroneous exploration or redundant wandering after the target answer has been reached. This inspires a new perspective: maintaining a goal-relative measure throughout reasoning, analogous to a Micromouse's spatial coordinates, to quantify how close each reasoning step is to the target answer and guide SFT to mask erroneous exploration and redundant wandering.

Building on this intuition, we propose MICROMOUSE, an automated selective-SFT framework for masking harmful supervision in long CoTs. Given a long CoT, we first partition it into a sequence of reasoning steps using naturally occurring transition keywords. We then introduce the *Relative Potential* (REP) to quantify how close each reasoning state is to the target answer. Perplexity provides a simple and intuitive measure of how strongly a reasoning prefix supports the gold answer, but the gold answer's perplexity alone captures only absolute confidence and ignores how competing answers change. REP instead contrasts the gold answer with a matched foil through the difference between their log perplexities, equivalently the log-ratio of their perplexities. This relative formulation cancels likelihood variations shared by the matched pair and more directly tracks changes in the model's preference for the target answer. By tracking the evolution of REP along the trajectory, we systematically categorize reasoning steps into Correct Progress, Necessary Exploration, Erroneous Exploration, and Redundant Wandering. During SFT, tokens belonging to erroneous or redundant steps are masked from the loss while remaining visible in the context.

Extensive experiments show that MICROMOUSE consistently outperforms Full-CoT SFT across student models of different scales and a broad range of reasoning benchmarks. Averaged over Qwen3-4B, Qwen3-8B, and Qwen3-14B, MICROMOUSE achieves relative improvements of **5.5%** in accuracy under greedy decoding and **2.6%** in Pass@1 under temperature sampling, while also clearly surpassing existing selective-SFT baselines. In addition, MICROMOUSE reduces the average output length by **8.8%**. The harmful supervision identification is also efficient, requiring only **1.7** seconds on average for a long CoT containing 22 reasoning steps.

2 METHODOLOGY

2.1 PRELIMINARY

Problem Definition. We focus on reasoning tasks with objectively verifiable target answers, including mathematical reasoning and multiple-choice reasoning. A long-form CoT $\mathcal{T}$ =

$\{S_1, S_2 \ldots, S_n\}$ typically consists of multiple reasoning steps serving different roles, such as problem decomposition, intermediate derivation, exploration, self-correction, and verification. Even when the CoT eventually reaches the correct answer, it may still contain harmful supervision, such as persistent erroneous exploration and redundant wandering after the solution has effectively been reached. Indiscriminately optimizing all such steps may reinforce undesirable reasoning behaviors and impair both reasoning quality and efficiency. Our goal is to identify reasoning steps that should not contribute to supervision and selectively exclude their token-level losses during SFT.

Relative Potential. Relative Potential (REP) is designed to measure how close the current reasoning state is to the gold answer along a long CoT. Prior work (Xiao et al., 2025; Cui et al., 2025) often uses the perplexity of the gold answer to estimate how strongly a reasoning prefix supports the target answer. However, gold-answer perplexity alone only reflects the absolute likelihood of the target and does not capture how the probability of other answers changes. In particular, the gold answer may become more likely while a nearby incorrect answer increases even more, in which case perplexity would suggest improved progress despite a deterioration in the model's relative preference for the gold answer. Motivated by this limitation, we instead adopt a contrastive view and compare the conditional log-likelihood of the gold answer with that of a foil answer (Warstadt et al., 2020). The foil is constructed as a minimal contrast to the gold answer: it preserves the answer format and token length while introducing a minimal rule-based perturbation (Gardner et al., 2020) that makes the answer incorrect (see details in Appendix A), e.g., $18 \to 19$, $A \to B$, or $\frac{4}{5} \to \frac{3}{5}$. Given a scoring model M, we define the *Relative Potential* at reasoning step k as

$$\phi_k^{(M)} = \frac{1}{L} \sum_{j=1}^{L} \left[\log p_M\big(y_j^* \mid q, S_{\le k}, r, y_{<j}^*\big) - \log p_M(\tilde{y}_j \mid q, S_{\le k}, r, \tilde{y}_{<j}) \right] = -\log \frac{\mathrm{PPL}_k^{\mathrm{gold}}}{\mathrm{PPL}_k^{\mathrm{foil}}}, \tag{1}$$

where q denotes the input question, r is an answer probe, and $p_M(\cdot)$ denotes the conditional probability assigned by the scoring model M, and L is the shared token length of the gold answer y^* and its foil $\tilde{y}$. Thus, ϕ_k represents the normalized conditional log-likelihood advantage of the gold answer over its foil, or equivalently, their log perplexity ratio. A larger REP indicates stronger relative support for the gold answer. Compared with gold-answer PPL alone, REP reduces shared likelihood variations, yielding a more robust estimate of changes in the reasoning state's support for the gold answer. We provide a formal analysis of these advantages of REP in Appendix B.

2.2 Overview

Given a long-form reasoning trajectory, we first partition it into reasoning steps at naturally occurring transition keywords, yielding $\mathcal{T} = \{S_1, S_2, \ldots, S_n\}$. MicroMouse then identifies harmful reasoning steps for supervision through three stages, as shown in Figure 1.

Stage 1: Relative Potential Computation. We compute the Relative Potential at each reasoning step to estimate the progress of the current reasoning state toward the target answer.

Stage 2: Harmful Supervision Step Identification. We analyze the changes in REP across reasoning steps to specifically identify erroneous exploration and redundant wandering.

Stage 3: Selective Loss Masking in SFT. Based on the identified reasoning steps, we construct a selective supervision mask and apply it to the loss optimization of downstream SFT.

2.3 Stage 1: Relative Potential Computation

Given a segmented reasoning CoT $\mathcal{T} = \{S_1, S_2, \ldots, S_n\}$, we compute the Relative Potential for each reasoning step sequentially. For the k-th reasoning step, we first construct the input query by concatenating the original question q, the reasoning prefix $S_{\le k}$, and a fixed answer probe r, instantiated as `</think> So, the final answer is \boxed{`. The resulting input can be written as $[q;\, S_{\le k};\, r]$.. We feed it into the scoring model M and compute the REP $\phi_k^{(M)}$ from the conditional log-likelihood difference between the gold answer y^* and its foil $\tilde{y}$, following Eq. 1. Repeating this procedure for $k = 1, \ldots, n$ yields the complete potential trajectory:

$$\Phi^{(M)} = \{\phi_1^{(M)}, \phi_2^{(M)}, \ldots, \phi_n^{(M)}\}, \tag{2}$$

which characterizes the evolution of the reasoning state's proximity to the target answer.

2.4 STAGE 2: HARMFUL SUPERVISION STEP IDENTIFICATION

Given the REP trajectory $\Phi^{(M)} = \{\phi_1^{(M)}, \phi_2^{(M)}, \ldots, \phi_n^{(M)}\}$, we use the potential change $\Delta\phi_k^{(M)} = \phi_k^{(M)} - \phi_{k-1}^{(M)}$ to quantify the effect of reasoning step S_k, and define $k^{\max} = \arg\max_k \phi_k^{(M)}$ as the step reaching the maximum REP. Let σ denote the REP noise scale estimated from the lag-1 autocovariance of its increments (see details in Appendix A).We categorize all reasoning steps into four categories:

(a) Correct Progress:

$$\mathcal{S}_{\text{progress}} = \left\{ S_k \mid k \le k^{\max},\ \Delta\phi_k^{(M)} \ge -\sigma \right\}. \tag{3}$$

These steps either increase the REP or introduce only minor fluctuations within the noise scale, and are retained for supervision.

(b) Necessary Exploration:

$$\mathcal{S}_{\text{explore}} = \left\{ S_k \mid k \le k^{\max},\ \Delta\phi_k^{(M)} < -\sigma,\ \Delta\phi_{k-1}^{(M)} \ge -\sigma,\ \Delta\phi_{k+1}^{(M)} \ge -\sigma \right\}. \tag{4}$$

These isolated potential decreases may reflect necessary local deviations toward the target answer, so we conservatively retain them for supervision.

(c) Erroneous Exploration:

$$\mathcal{S}_{\text{error}} = \left\{ S_k \mid k \le k^{\max},\ \Delta\phi_k^{(M)} < -\sigma,\ \left(\Delta\phi_{k-1}^{(M)} < -\sigma \ \vee\ \Delta\phi_{k+1}^{(M)} < -\sigma \right) \right\}. \tag{5}$$

These steps belong to consecutive potential decreases, indicating persistent movement away from the target answer, and are masked.

(d) Redundant Wandering:

$$\mathcal{S}_{\text{redundant}} = \{ S_k \mid k > k^{\max} \}. \tag{6}$$

These steps occur after the highest-potential reasoning step and may correspond to redundant reasoning that fails to terminate in time, and are therefore masked.

Accordingly, MICROMOUSE keeps $\mathcal{S}_{\text{progress}} \cup \mathcal{S}_{\text{explore}}$ while masking $\mathcal{S}_{\text{error}} \cup \mathcal{S}_{\text{redundant}}$.

2.5 STAGE 3: SELECTIVE LOSS MASKING IN SFT

Directly removing harmful reasoning steps from the training trajectory may break the continuity of the reasoning CoT and disrupt the contextual dependencies required by subsequent reasoning. We therefore preserve the complete reasoning CoT and selectively mask the corresponding token losses during SFT. Given an input token sequence $T = \{t_1, t_2, \ldots, t_l\}$, the MICROMOUSE objective is formulated as

$$\mathcal{L}_{\text{MicroMouse}(\theta)} = -\frac{1}{\sum_{i=1}^{l} \mathbb{I}(t_i)} \sum_{i=1}^{l} \mathbb{I}(t_i) \log P_\theta \left(t_i \mid q, t_{<i}; \theta \right), \tag{7}$$

where $\mathbb{I}(t_i) = 0$ if t_i belongs to a reasoning step in $\mathcal{S}_{\text{error}} \cup \mathcal{S}_{\text{redundant}}$, and $\mathbb{I}(t_i) = 1$ otherwise.

3 EXPERIMENTS

3.1 EXPERIMENTAL SETUP

Datasets. We use the official LIMO training set (Ye et al., 2025), which contains 817 carefully curated mathematical problems paired with long-form reasoning CoTs, as the supervision data for all SFT experiments. Following a similar evaluation protocol to Ye et al. (2025), we evaluate the resulting models on a diverse suite of reasoning benchmarks covering both in-domain and out-of-domain generalization. The in-domain benchmarks include MATH500 (Hendrycks et al., 2021), AMC23, and AIME24, which closely match the mathematical reasoning distribution of the training data. We further include HMMT26 as a recent competition-mathematics benchmark to assess generalization to newly released and unseen problems. For out-of-domain evaluation, we consider GPQA-Diamond (Rein et al., 2023), Minerva (Lewkowycz et al., 2022), and OlympiadBench (He et al., 2024), which cover challenging scientific, quantitative, and Olympiad-level reasoning problems.

Metrics. We evaluate reasoning performance under greedy decoding and temperature sampling. Greedy decoding uses exact-match accuracy (**Acc**), while temperature sampling reports **Pass@1** and **Pass@6**, measuring average correctness and the probability of obtaining at least one correct solution within six samples, respectively.

Implementation Details. During training, we use DeepSeek-R1-Distill-Qwen-1.5B as the default scoring model M to compute the Relative Potential and derive the selective supervision masks. We conduct SFT on three dense Qwen3 models, namely Qwen3-4B-Base, Qwen3-8B-Base, and Qwen3-14B-Base. All models are trained for 9 epochs on the 817 long-CoT trajectories from LIMO with a learning rate of 1×10^{-5}. For a fair comparison, Full CoT SFT, MICROMOUSE, and other existing baselines use exactly the same training data and hyperparameters. During evaluation, we evaluate all models under a zero-shot CoT setting with a maximum generation length of 32,768 tokens. Greedy decoding uses $T = 0$, while temperature sampling uses $T = 0.6$. For MATH500, GPQA-Diamond, Minerva, and OlympiadBench, we sample 6 responses per problem. For the smaller AMC23, AIME24, and HMMT26 benchmarks, we increase the number of sampled responses to 16 per problem to obtain a more stable estimate of average performance. All training and evaluation experiments are conducted on 8 NVIDIA H100 GPUs.

3.2 MAIN RESULTS

Table 1 presents a comparison across seven reasoning benchmarks, three model scales, and two decoding settings. Compared with the corresponding base models before SFT, both fine-tuning strategies substantially improve reasoning performance, while MICROMOUSE delivers more consistent gains over Full CoT SFT across evaluation settings. Specifically, averaged across the three model scales, MICROMOUSE achieves a relative improvement of **5.5%** in accuracy under greedy decoding, and relative improvements of **2.6%** and **0.9%** in Pass@1 and Pass@6 under temperature sampling. The gains are particularly pronounced for Qwen3-8B, where MICROMOUSE improves over Full CoT SFT by **9.9%** under greedy decoding and **4.2%** in Pass@1 under temperature sampling.

We further observe that the improvement is generally larger under greedy decoding than under temperature sampling. One plausible explanation is that greedy decoding directly reflects changes to the model's highest-probability reasoning trajectory, whereas stochastic sampling explores a broader set of candidate trajectories and consequently reduces the sensitivity of aggregate metrics to improvements concentrated around the dominant decoding path. Nevertheless, MICROMOUSE retains consistent gains under temperature sampling, indicating that its benefits are not restricted to a single deterministic decoding trajectory and remain robust under more diverse generation settings.

Performance across Problem Difficulty. To investigate whether MICROMOUSE provides different benefits across problem difficulty, we further break down the MATH500 results on Qwen3-8B according to the five difficulty levels. Following the easy-to-hard split adopted by Sun et al. (2024), we group Levels 1–3 as easy problems and Levels 4–5 as hard problems, as shown in Table 2. Overall, MICROMOUSE improves over Full CoT SFT from 84.4% to 86.5% in Pass@1 and from 95.4% to 96.0% in Pass@6, corresponding to relative improvements of approximately 2.5% and 0.6%. More importantly, the benefit becomes more pronounced as problem difficulty increases. MICROMOUSE improves Pass@1 over Full CoT SFT by only 0.4% on easy problems, but by 4.8% on hard problems, indicating substantially larger gains as problem difficulty increases. A similar tendency is observed for Pass@6, although the margin is smaller due to the near-saturated performance. This pattern is consistent with our intuition that harder problems generally involve more complex and exploratory reasoning, increasing the likelihood of persistent erroneous exploration and redundant reasoning. By selectively suppressing supervision from such reasoning steps, MICROMOUSE provides a larger advantage on problems that require more challenging reasoning trajectories.

3.3 COMPARISON WITH EXITING METHODS

In this section, we compare MICROMOUSE on Qwen3-8B against five selective-SFT baselines: (1) **First-Correct** (Chen et al., 2025a), which retains the segment where the correct answer first appears together with all preceding segments; (2) **Segment-Entropy** (Li et al., 2026), which prioritizes segments with higher aggregated token-level entropy, indicating greater information contribution; (3) **Segment-PPL** (Cui et al., 2025), which estimates segment importance by measuring the increase

Table 1: **Main results across model scales and reasoning benchmarks.** We compare Base, Full CoT SFT, and MICROMOUSE on four in-domain and three out-of-domain benchmarks. Overall reports accuracy under greedy decoding and Pass@1/Pass@6 under temperature sampling. "–" indicates that Pass@6 is not applicable to greedy decoding. The best results are highlighted in **bold**.

		In-Domain				Out-of-Domain			Overall	
Model	**Method**	**MATH500**	**AMC23**	**AIME24**	**HMMT26**	**GPQA**	**Minerva**	**Olympiad**	**Acc./P@1**	**P@6**
Greedy Decoding ($T = 0$)										
Qwen3-4B	Base	62.6	47.5	16.7	3.0	17.2	19.9	32.6	28.5	–
	Full CoT SFT	70.8	47.5	13.3	12.1	35.4	**21.3**	38.8	34.2	–
	MICROMOUSE	**71.6**	**55.0**	**13.3**	**12.1**	**36.9**	21.0	**39.3**	**35.6**$^{\uparrow 4.2\%}$	–
Qwen3-8B	Base	76.2	62.5	13.3	9.1	41.4	29.4	40.3	38.9	–
	Full CoT SFT	76.0	57.5	30.0	9.1	42.9	30.1	41.6	41.0	–
	MICROMOUSE	**80.4**	**70.0**	**30.0**	**12.1**	**44.9**	**30.9**	**47.4**	**45.1**$^{\uparrow 9.9\%}$	–
Qwen3-14B	Base	81.0	65.0	13.3	12.1	42.9	32.0	44.9	41.6	–
	Full CoT SFT	**91.4**	85.0	**50.0**	24.2	55.6	**36.0**	**60.4**	57.5	–
	MICROMOUSE	90.8	**87.5**	43.3	**36.4**	**58.6**	**36.0**	59.9	**58.9**$^{\uparrow 2.5\%}$	–
Temperature Sampling ($T = 0.6$)										
Qwen3-4B	Base	35.8	26.6	7.2	3.2	7.2	12.1	20.0	16.0	40.5
	Full CoT SFT	**81.0**	**65.3**	20.0	11.9	41.1	**30.9**	**47.3**	42.5	62.2
	MICROMOUSE	80.5	65.0	**23.8**	**15.2**	**42.6**	28.9	46.4	**43.2**$^{\uparrow 1.6\%}$	**62.4**$^{\uparrow 0.3\%}$
Qwen3-8B	Base	72.9	52.5	11.9	5.1	40.7	26.0	37.3	35.2	57.2
	Full CoT SFT	84.4	71.4	30.6	19.1	48.7	34.0	52.3	48.6	68.9
	MICROMOUSE	**86.5**	**73.4**	**32.9**	**21.8**	**51.5**	**35.8**	**54.5**	**50.9**$^{\uparrow 4.7\%}$	**70.3**$^{\uparrow 2.0\%}$
Qwen3-14B	Base	73.8	53.0	12.9	8.9	42.8	29.6	38.8	37.1	59.1
	Full CoT SFT	**91.5**	**85.2**	46.9	29.7	57.3	35.2	**61.1**	58.1	73.5
	MICROMOUSE	91.3	85.0	**50.8**	**31.1**	**59.9**	**35.3**	59.6	**59.0**$^{\uparrow 1.5\%}$	**73.8**$^{\uparrow 0.4\%}$

Table 2: **Qwen3-8B temperature sampling performance across MATH500 difficulty levels.**

Setting	**Metric**	**Easy**				**Hard**			**Total Avg.**
		Level 1	**Level 2**	**Level 3**	**Avg.**	**Level 4**	**Level 5**	**Avg.**	
Full CoT SFT	Pass@1	89.1	**93.9**	91.0	91.8	82.6	73.1	77.7	84.4
	Pass@6	95.3	98.9	**99.0**	98.3	95.3	90.3	92.7	95.4
MICROMOUSE	Pass@1	**92.6**	92.0	**92.1**	**92.2**$^{\uparrow 0.4\%}$	**85.8**	**77.2**	**81.4**$^{\uparrow 4.8\%}$	**86.5**$^{\uparrow 2.5\%}$
	Pass@6	**100.0**	**98.9**	98.1	**98.7**$^{\uparrow 0.4\%}$	**96.1**	**91.0**	**93.5**$^{\uparrow 0.9\%}$	**96.0**$^{\uparrow 0.6\%}$

in CoT perplexity after removing a reasoning segment; (4) **Gemini Annotate**, which directly uses Gemini-3.6-Flash to annotate the importance of each reasoning step, with the annotation prompt provided in the Appendix C; and (5) **Segment Selective** (Wang et al., 2026b), which identifies important segments by aggregating token-level Integrated Gradients (IG) signals. For Segment-Entropy, Segment-PPL, Segment Selective, and MICROMOUSE, we use the same scoring model and match the supervision budget to ensure a fair comparison.

As shown in the upper row of Figure 2, we compare MICROMOUSE with Full CoT SFT and five selective-SFT baselines across seven reasoning benchmarks. Several selective-SFT strategies consistently outperform Full CoT SFT, providing further evidence that long CoTs contain harmful supervision and that masking such signals can yield stable gains in reasoning performance. Among the six selective-SFT methods, MICROMOUSE achieves the best overall performance under both decoding settings. These results further suggest that measuring each reasoning step by its relative potential enables more accurate identification and masking of harmful supervision in long CoTs. Interestingly, Gemini Annotate achieves the second-best overall performance, indicating that direct reasoning-step annotation with a strong LLM is also effective for selective supervision.

Reasoning Efficiency. The bottom row of Figure 2 further compares the average output length of Full CoT SFT and the six selective-SFT strategies under both greedy decoding and temperature

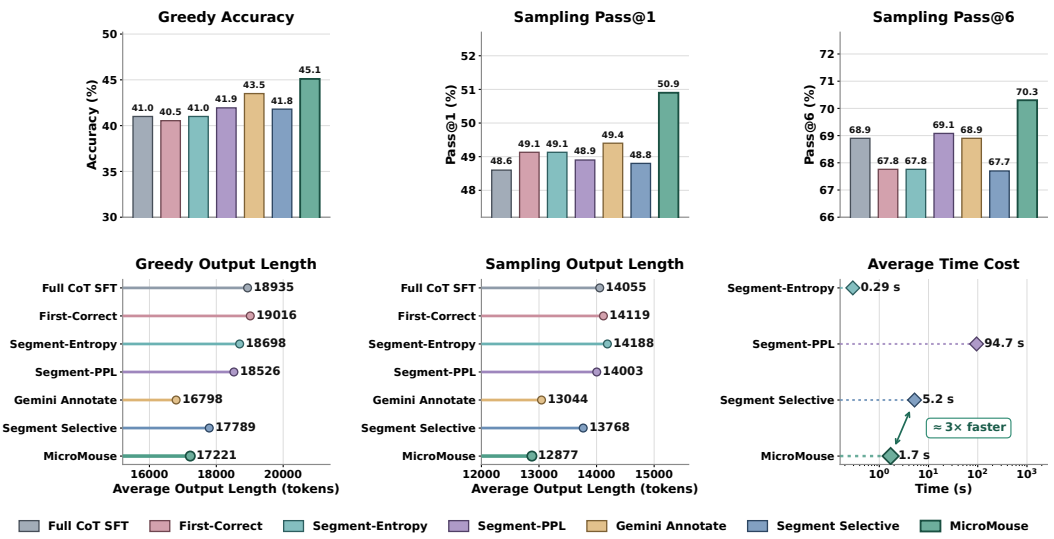

Figure 2: **Comparison with other selective-SFT baselines on Qwen3-8B.** The top row reports reasoning performance under greedy decoding and temperature sampling, while the bottom row compares output length and the computational cost of masking harmful supervision.

sampling. First-Correct, Segment-Entropy, and Segment-PPL produce output lengths comparable to Full CoT SFT, whereas Gemini Annotate, Segment Selective, and MICROMOUSE yield noticeably shorter generations. In particular, MICROMOUSE reduces the average output length by 9.1% under greedy decoding and 8.4% under temperature sampling relative to Full CoT SFT. Notably, MICROMOUSE-SFT improves reasoning performance while producing shorter generations. This joint improvement suggests that masking harmful supervision during SFT encourages the student model to learn more effective and concise reasoning behaviors, reducing erroneous exploration and redundant wandering while improving reasoning performance. Although Gemini Annotate also reduces output length, MICROMOUSE achieves stronger performance, resulting in a better trade-off.

Computational Overhead. The bottom-right panel of Figure 2 compares the average time cost of the annotation-free methods on all 817 LIMO trajectories, each containing about 22 reasoning steps. Segment-Entropy is the fastest because it requires only one forward pass to compute token-level entropy, whereas Segment-PPL repeatedly removes reasoning steps and recomputes perplexity, resulting in a much higher cost of 94.7 seconds per trajectory. Segment Selective takes 5.2 seconds due to token-level IG computation and aggregation. In contrast, MICROMOUSE only evaluates the conditional log-likelihoods of the gold answer and its matched foil at each step, requiring 1.7 seconds per trajectory and running about $3\times$ faster than Segment Selective. Although Segment-Entropy is cheaper, MICROMOUSE yields substantially stronger downstream performance, achieving a better balance between supervision quality and computational cost for large scale long CoT SFT.

3.4 ABLATION STUDY

We investigate the contributions of individual MICROMOUSE components and the effect of replacing the scoring model, with results summarized in Table 3. For component ablations, we consider four variants: (1) *Random Mask*, which randomly masks reasoning segments until matching the supervision budget of MICROMOUSE; (2) *w/o Mask* $\mathcal{S}_{\text{error}}$, which does not mask reasoning steps identified as erroneous exploration; (3) *w/o Mask* $\mathcal{S}_{\text{redundant}}$, which does not mask reasoning steps identified as re-

Table 3: **Ablation studies on Qwen3-8B.**

		Greedy		Sampling		
Variant	**Keep (%)**	**Acc.**	**Len. (k)**	**P@1**	**P@6**	**Len. (k)**
Full CoT SFT	100.0	41.0	18.94	48.6	68.9	14.06
Random Mask	80.9	41.7	17.89	49.1	68.2	14.01
MICROMOUSE	80.9	**45.1**	17.22	**50.9**	**70.3**	12.88
w/o Mask $\mathcal{S}_{\text{error}}$	83.5	44.4	16.67	50.7	70.2	12.84
w/o Mask $\mathcal{S}_{\text{redundant}}$	97.1	38.9	19.34	49.0	69.0	14.11
w/ Mask $\mathcal{S}_{\text{explore}}$	74.1	41.8	18.06	48.9	68.2	13.81
w/ Qwen3-4B	85.5	41.9	18.27	49.0	70.2	14.18

dundant wandering; and (4) *w/ Mask* $\mathcal{S}_{\text{explore}}$, which additionally masks reasoning segments exhibiting a single-step decrease in REP. We further replace the default DeepSeek-R1-Distill-Qwen-1.5B with Qwen3-4B-Instruct to assess the sensitivity of MICROMOUSE to the choice of scoring model.

The component ablations reveal that the gains of MICROMOUSE arise primarily from which reasoning steps are excluded from supervision, rather than from simply reducing the amount of training signal. Random Mask, despite using the same supervision budget as MICROMOUSE, yields only marginal changes over Full CoT SFT, while remaining substantially below MICROMOUSE across all performance metrics. Removing the masking of either erroneous exploration or redundant wandering also degrades performance, supporting the necessity of suppressing these two types of harmful supervision. In particular, failing to mask redundant wandering results in relative performance drops of 13.7% in greedy accuracy, 3.4% in Pass@1, and 1.9% in Pass@6 under temperature sampling. Conversely, additionally masking reasoning steps with only a single-step decrease in REP causes a marked performance degradation. This result supports our design choice of preserving isolated potential decreases, which may correspond to necessary intermediate exploration rather than genuinely harmful supervision signals. Finally, replacing the default scoring model with Qwen3-4B-Instruct still yields consistent improvements over Full CoT SFT, including a 1.9% relative gain in Pass@6. This result indicates that the effectiveness of MICROMOUSE is not restricted to the default scorer and remains stable when using another alternative scoring model.

3.5 CASE STUDY

Figure 3 visualizes the Relative Potential trajectory of MICROMOUSE on LIMO sample #539 and compares its step-wise supervision decisions with those of other selective-SFT baselines. The original reasoning trajectory is provided in Appendix D. Several observations emerge from this example. **First, an isolated decrease in Relative Potential does not necessarily indicate harmful supervision.** At Step 5, the model reaches the correct answer through a logical jump, causing a temporary decrease in Relative Potential. Although the argument at this step is insufficiently justified, it still advances toward the target answer, and the subsequent step begins to further verify the result. **Second, consecutive decreases in Relative Potential correspond more clearly to erroneous exploration.** At Steps 7–8, the reasoning repeatedly recomputes an already established counting argument and then transitions into hesitation without introducing new progress. **Third, reasoning steps after the Relative Potential reaches its peak exhibit clear redundant wandering.** By Step 12, the model has repeatedly cross-validated the derivation and reached the target answer with high confidence. Nevertheless, Steps 13–15 continue to revisit alternative formulations and perform additional sanity checks, despite no longer contributing substantive progress toward the solution.

Taken together, this example provides empirical support for MICROMOUSE's identification of erroneous exploration and redundant wandering based on the evolution of REP. It also illustrates why isolated potential drops should be treated conservatively, as they may correspond to necessary exploration rather than genuinely harmful supervision. In contrast, other selective-SFT baselines are either overly aggressive or permissive, leading to less accurate supervision selection on this example.

4 RELATED WORK

Reasoning Efficiency of Large Reasoning Models. Large reasoning models (LRMs) achieve strong performance through extended chains of thought (CoTs), but long-form reasoning often incurs substantial computational overhead and redundant reasoning (Sui et al., 2025; Wang et al., 2026a). Existing approaches improve reasoning efficiency either at inference time or through training. Prompt- and budget-based methods explicitly constrain test-time computation (Han et al., 2025; Renze & Guven, 2024), while latent-reasoning methods reduce explicit verbalized computation by performing intermediate reasoning in continuous representations (Hao et al., 2025). Training-based methods instead refine or compress CoT supervision, for example by identifying critical reasoning steps using perplexity-based criteria (Cui et al., 2025). In contrast, MICROMOUSE targets the quality rather than the length of long-CoT supervision, preserving the complete trajectory as context while excluding harmful reasoning units from loss optimization.

Selective SFT and Importance Measurement for Reasoning CoTs. Recent work challenges the conventional SFT objective that optimizes all target tokens uniformly. Rho-1 selectively applies loss

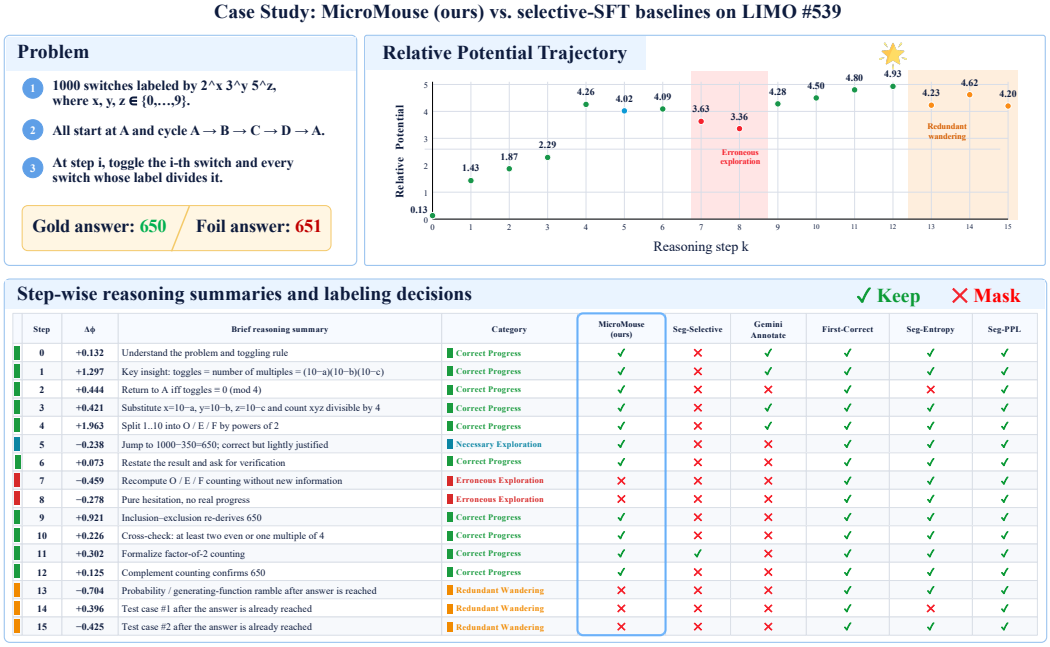

Figure 3: **Case study of step-wise supervision analysis on LIMO #539.** The figure visualizes the Relative Potential trajectory and compares the supervision decisions of MICROMOUSE with selective-SFT baselines across all reasoning steps.

to high-utility tokens identified by a reference model (Lin et al., 2025), Goldfish Loss randomly excludes subsets of target tokens from optimization (Hans et al., 2024), and Token Cleaning removes low-quality token-level supervision during SFT (Pang et al., 2025). At the step level, ATLAS identifies critical steps in agent trajectories and fine-tunes only on selected actions (Chen et al., 2025b), while GLM-5 (GLM-5-Team et al., 2026) masks erroneous trajectory segments while retaining them as context for subsequent correction. For long reasoning CoTs, the central challenge is therefore to determine which parts of a trajectory should contribute to supervision. Existing methods operate primarily at token or segment granularity. Token-level approaches use signals such as logits, entropy, gradients, or contrastive differences to identify informative or error-critical tokens (Cheng et al., 2025; Wang et al., 2025). Since individual tokens may not correspond to semantically complete reasoning decisions, segment-level methods instead evaluate coherent reasoning units using perplexity (Cui et al., 2025), step entropy (Li et al., 2026), or perturbation-based attribution (Xu et al., 2025), while Wang et al. (2026b) aggregate token-level Integrated Gradients into segment-level attribution signals. Rather than assigning each segment an isolated importance score, MICROMOUSE tracks the evolution of Relative Potential along the reasoning trajectory to identify erroneous exploration and redundant wandering, and masks the corresponding harmful supervision during SFT.

5 CONCLUSION

In this work, we propose MICROMOUSE, an automated selective-SFT framework for masking harmful supervision in long CoTs. MICROMOUSE maintains a Relative Potential to measure how close each reasoning state is to the target answer, enabling the identification of erroneous exploration and redundant wandering along long reasoning trajectories. Unlike single answer log-perplexity measures, Relative Potential is defined through the conditional log-likelihood difference between the gold answer and a matched foil answer, which reduces shared likelihood fluctuations and provides a more stable signal for tracking reasoning progress. Extensive experiments s show that MICROMOUSE consistently outperforms Full CoT SFT and achieves stronger overall performance than existing selective-SFT baselines. It also delivers clear advantages in reasoning efficiency and computational overhead. Overall, MICROMOUSE provides an effective and scalable approach for identifying and masking harmful supervision in large-scale long-CoT SFT.

AI USE STATEMENT

In this work, we used generative AI tools for literature research and language editing. Specifically, GPT-5.6 Sol was used to assist in retrieving and summarizing relevant literature and to improve the clarity, grammatical correctness, and readability of the manuscript. Generative AI was not used to originate research ideas, design the methodology or experiments, or interpret experimental results; the research conception, initial manuscript draft, experimental decisions, and experimental execution were independently completed by the authors. Generative AI was also used to assist in implementing and reproducing baseline methods from prior work, with all generated code reviewed and verified by the authors. We reviewed all AI-assisted content, cross-checked cited references against their original publications, and revised all AI-edited text to ensure technical accuracy. We take full responsibility for the final content of this work, including any text, claims, or artifacts produced with the assistance of generative AI.

REPRODUCIBILITY STATEMENT

To support reproducibility, we provide a detailed description of our framework in Section 2, together with comprehensive implementation and methodological details in Section 3, Appendix A and Appendix C. We will release our data and code after the anonymous review process.

REFERENCES

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*, 2025a. URL https://openreview.net/forum?id=MSbU3L7V00.

Zhixun Chen, Ming Li, Yuxuan Huang, Yali Du, Meng Fang, and Tianyi Zhou. ATLAS: Agent tuning via learning critical steps. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 25334–25349, Vienna, Austria, July 2025b. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.1299. URL https://aclanthology.org/2025.findings-acl.1299/.

Daixuan Cheng, Shaohan Huang, Xuekai Zhu, Bo Dai, Wayne Xin Zhao, Zhenliang Zhang, and Furu Wei. Reasoning with exploration: An entropy perspective, 2025. URL https://arxiv.org/abs/2506.14758.

Yingqian Cui, Pengfei He, Jingying Zeng, Hui Liu, Xianfeng Tang, Zhenwei Dai, Yan Han, Chen Luo, Jing Huang, Zhen Li, Suhang Wang, Yue Xing, Jiliang Tang, and Qi He. Stepwise perplexity-guided refinement for efficient chain-of-thought reasoning in large language models. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 18581–18597, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.956. URL https://aclanthology.org/2025.findings-acl.956/.

Matt Gardner, Yoav Artzi, Victoria Basmov, Jonathan Berant, Ben Bogin, Sihao Chen, Pradeep Dasigi, Dheeru Dua, Yanai Elazar, Ananth Gottumukkala, Nitish Gupta, Hannaneh Hajishirzi, Gabriel Ilharco, Daniel Khashabi, Kevin Lin, Jiangming Liu, Nelson F. Liu, Phoebe Mulcaire, Qiang Ning, Sameer Singh, Noah A. Smith, Sanjay Subramanian, Reut Tsarfaty, Eric Wallace, Ally Zhang, and Ben Zhou. Evaluating models' local decision boundaries via contrast sets. In Trevor Cohn, Yulan He, and Yang Liu (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 1307–1323, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.117. URL https://aclanthology.org/2020.findings-emnlp.117/.

GLM-5-Team, :, Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, Chenzheng Zhu, Congfeng Yin, Cunxiang

Wang, Gengzheng Pan, Hao Zeng, Haoke Zhang, Haoran Wang, Huilong Chen, Jiajie Zhang, Jian Jiao, Jiaqi Guo, Jingsen Wang, Jingzhao Du, Jinzhu Wu, Kedong Wang, Lei Li, Lin Fan, Lucen Zhong, Mingdao Liu, Mingming Zhao, Pengfan Du, Qian Dong, Rui Lu, Shuang-Li, Shulin Cao, Song Liu, Ting Jiang, Xiaodong Chen, Xiaohan Zhang, Xuancheng Huang, Xuezhen Dong, Yabo Xu, Yao Wei, Yifan An, Yilin Niu, Yitong Zhu, Yuanhao Wen, Yukuo Cen, Yushi Bai, Zhongpei Qiao, Zihan Wang, Zikang Wang, Zilin Zhu, Ziqiang Liu, Zixuan Li, Bojie Wang, Bosi Wen, Can Huang, Changpeng Cai, Chao Yu, Chen Li, Chengwei Hu, Chenhui Zhang, Dan Zhang, Daoyan Lin, Dayong Yang, Di Wang, Ding Ai, Erle Zhu, Fangzhou Yi, Feiyu Chen, Guohong Wen, Hailong Sun, Haisha Zhao, Haiyi Hu, Hanchen Zhang, Hanrui Liu, Hanyu Zhang, Hao Peng, Hao Tai, Haobo Zhang, He Liu, Hongwei Wang, Hongxi Yan, Hongyu Ge, Huan Liu, Huanpeng Chu, Jia'ni Zhao, Jiachen Wang, Jiajing Zhao, Jiamin Ren, Jiapeng Wang, Jiaxin Zhang, Jiayi Gui, Jiayue Zhao, Jijie Li, Jing An, Jing Li, Jingwei Yuan, Jinhua Du, Jinxin Liu, Junkai Zhi, Junwen Duan, Kaiyue Zhou, Kangjian Wei, Ke Wang, Keyun Luo, Laiqiang Zhang, Leigang Sha, Liang Xu, Lindong Wu, Lintao Ding, Lu Chen, Minghao Li, Nianyi Lin, Pan Ta, Qiang Zou, Rongjun Song, Ruiqi Yang, Shangqing Tu, Shangtong Yang, Shaoxiang Wu, Shengyan Zhang, Shijie Li, Shuang Li, Shuyi Fan, Wei Qin, Wei Tian, Weining Zhang, Wenbo Yu, Wenjie Liang, Xiang Kuang, Xiangmeng Cheng, Xiangyang Li, Xiaoquan Yan, Xiaowei Hu, Xiaoying Ling, Xing Fan, Xingye Xia, Xinyuan Zhang, Xinze Zhang, Xirui Pan, Xu Zou, Xunkai Zhang, Yadi Liu, Yandong Wu, Yanfu Li, Yidong Wang, Yifan Zhu, Yijun Tan, Yilin Zhou, Yiming Pan, Ying Zhang, Yinpei Su, Yipeng Geng, Yong Yan, Yonglin Tan, Yuean Bi, Yuhan Shen, Yuhao Yang, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yurong Wu, Yutao Zhang, Yuxi Duan, Yuxuan Zhang, Zezhen Liu, Zhengtao Jiang, Zhenhe Yan, Zheyu Zhang, Zhixiang Wei, Zhuo Chen, Zhuoer Feng, Zijun Yao, Ziwei Chai, Ziyuan Wang, Zuzhou Zhang, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. Glm-5: from vibe coding to agentic engineering, 2026. URL https://arxiv.org/abs/2602.15763.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z.

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. Token-budget-aware LLM reasoning. In 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/.

Abhimanyu Hans, John Kirchenbauer, Yuxin Wen, Neel Jain, Hamid Kazemi, Prajwal Singhania, Siddharth Singh, Gowthami Somepalli, Jonas Geiping, Abhinav Bhatele, and Tom Goldstein. Be like a goldfish, don't memorize! mitigating memorization in generative LLMs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=DylSyAfmWs.

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/.

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. *CoRR*, abs/2103.03874, 2021. URL https://arxiv.org/abs/2103.03874.

Aitor Lewkowycz, Anders Johan Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Venkatesh Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving quantitative reasoning problems with language models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=IFXTZERXdM7.

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.

Zhenghao Lin, Zhibin Gou, Yeyun Gong, Xiao Liu, Yelong Shen, Ruochen Xu, Chen Lin, Yujiu Yang, Jian Jiao, Nan Duan, and Weizhu Chen. Rho-1: Not all tokens are what you need, 2025. URL https://arxiv.org/abs/2404.07965.

OpenAI. Openai o1 system card. *CoRR*, abs/2412.16720, 2024. doi: 10.48550/ARXIV.2412.16720. URL https://doi.org/10.48550/arXiv.2412.16720.

Jinlong Pang, Na Di, Zhaowei Zhu, Jiaheng Wei, Hao Cheng, Chen Qian, and Yang Liu. Token cleaning: Fine-grained data selection for LLM supervised fine-tuning. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=tXkOUS3vLS.

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, 2023. URL https://arxiv.org/abs/2311.12022.

Matthew Renze and Erhan Guven. The benefits of a concise chain of thought on problem-solving in large language models. In *2024 2nd International Conference on Foundation and Large Language Models (FLLM)*, pp. 476–483. IEEE, November 2024. doi: 10.1109/fllm63129.2024.10852493. URL http://dx.doi.org/10.1109/FLLM63129.2024.10852493.

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.

Zhiqing Sun, Longhui Yu, Yikang Shen, Weiyang Liu, Yiming Yang, Sean Welleck, and Chuang Gan. Easy-to-hard generalization: Scalable alignment beyond human supervision, 2024. URL https://arxiv.org/abs/2403.09472.

Rui Wang, Hongru Wang, Boyang Xue, Yixia Li, Jianhui Pang, Shudong Liu, Yi Chen, Jiahao Qiu, Derek Fai Wong, Guanhua Chen, Heng Ji, and Kam-Fai Wong. Harnessing the reasoning economy: A survey of efficient reasoning for large language models, 2026a. URL https://arxiv.org/abs/2503.24377.

Shenzhi Wang, Le Yu, Chang Gao, Chujie Zheng, Shixuan Liu, Rui Lu, Kai Dang, Xiong-Hui Chen, Jianxin Yang, Zhenru Zhang, Yuqiong Liu, An Yang, Andrew Zhao, Yang Yue, Shiji Song, Bowen Yu, Gao Huang, and Junyang Lin. Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for LLM reasoning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=yfcpdY4gMP.

Siyuan Wang, Yanchen Liu, and Xiang Ren. Segment-level attribution for selective learning of long reasoning traces. In *The Fourteenth International Conference on Learning Representations*, 2026b. URL https://openreview.net/forum?id=C1UD4FLFPL.

Alex Warstadt, Alicia Parrish, Haokun Liu, Anhad Mohananey, Wei Peng, Sheng-Fu Wang, and Samuel R. Bowman. BLiMP: The benchmark of linguistic minimal pairs for English. *Transactions of the Association for Computational Linguistics*, 8:377–392, 2020. doi: 10.1162/tacl_a_00321. URL https://aclanthology.org/2020.tacl-1.25/.

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. doi: 10.52202/068431-1800. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf.

L. Wyard-Scott and Q.-H.M. Meng. A potential maze solving algorithm for a micromouse robot. In *IEEE Pacific Rim Conference on Communications, Computers, and Signal Processing. Proceedings*, pp. 614–618, 1995. doi: 10.1109/PACRIM.1995.520440.

Yang Xiao, Jessie Wang, Ruifeng Yuan, Chunpu Xu, Kaishuai Xu, Wenjie Li, and Pengfei Liu. LIMOPro: Reasoning refinement for efficient and effective test-time scaling. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=W9Y0jtf45v.

Zhentao Xu, Fengyi Li, Albert C. Chen, and Xiaofeng Wang. ProCut: LLM prompt compression via attribution estimation. In Saloni Potdar, Lina Rojas-Barahona, and Sebastien Montella (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 285–309, Suzhou (China), November 2025. Association for Computational Linguistics. ISBN 979-8-89176-333-3. doi: 10.18653/v1/2025.emnlp-industry.20. URL https://aclanthology.org/2025.emnlp-industry.20/.

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.

A FOIL CONSTRUCTION AND NOISE-SCALE ESTIMATION

A.1 FOIL CONSTRUCTION

The potential function is read out as a length-normalised log-likelihood ratio between the gold answer and a *foil*: an answer string that matches the gold one in token count and surface format but carries a minimally different, incorrect value. Matching length and format holds everything except value-correctness fixed, so the ratio measures only whether the reasoning prefix has pinned down the right value; taking the *nearest* wrong value makes the foil the hardest available competitor, which keeps the signal sensitive to reasoning progress instead of saturating early. Answers are numeric: an integer answer v is perturbed to $v + 1$, falling back to $v - 1$ whenever $v + 1$ would change the digit count or equal zero, and a fractional answer is perturbed by incrementing its numerator.

```
import re

def make_foil(ans):
    '''Minimally-perturbed counterfactual answer:
       same digit count, same format, nearest value, but wrong.'''
    a = str(ans).strip()

    m = re.fullmatch(r'-?\d+', a)                        # integer
    answers
    if m:
        v = int(a)
        d = v + 1 if v + 1 != 0 else v + 2               # never use 0 as
     a foil
        if len(str(abs(d))) != len(str(abs(v))):         # keep digit
    count stable
            d = v - 1 if v - 1 != 0 else v - 2
        return str(d)

    num, den = re.fullmatch(r'(-?\d+)/(\d+)', a).groups()   # simple
    fractions
    return '%d/%s' % (int(num) + 1, den)
```

A.2 NOISE SCALE AS THE DESCENT THRESHOLD

Because ϕ is estimated by a finite forward pass of a scorer model, each increment $\Delta\phi_k = \phi_k - \phi_{k-1}$ carries estimation jitter on top of genuine progress, so a threshold is needed before a negative increment may be called a descent. Writing the observed potential as signal plus independent noise, $\phi_k = s_k + \eta_k$ with η_k i.i.d. of variance σ^2, first differencing turns the noise into the MA(1) term $\eta_k - \eta_{k-1}$, whose lag-1 autocovariance is exactly $-\sigma^2$, while a smooth signal drift contributes approximately zero at lag 1:

$$\mathrm{Cov}\big(\Delta\phi_k, \Delta\phi_{k+1}\big) \;\approx\; -\sigma^2.$$

A negative lag-1 autocovariance is therefore the fingerprint of the noise alone, and we take $\sigma = \sqrt{-\mathrm{Cov}(\Delta\phi_k, \Delta\phi_{k+1})}$ as the noise scale, which serves directly as the descent threshold: a step counts as a significant descent only when $\Delta\phi_k < -\sigma$, whereas a step with $|\Delta\phi_k| \le \sigma$ is indistinguishable from noise and is left to be learned, neither breaking a run nor counting toward its length. Since σ shares the units of $\Delta\phi$, the criterion introduces no tunable threshold and adapts to each trace.

```
import numpy as np

def noise_scale(phi):
    '''sigma from the lag-1 autocovariance of the increments:
       Cov(d_k, d_{k+1}) = -sigma^2 for i.i.d. noise.'''
    d  = np.diff(np.asarray(phi, dtype=np.float64))
    x  = d - d.mean()
    c1 = float(np.mean(x[:-1] * x[1:]))            # lag-1
    autocovariance
```

```
    return float(np.sqrt(max(0.0, -c1)))

# the noise scale is the threshold itself
# significant descent:  d < -sigma
# noise-level step (|d| <= sigma): learned, does not break or extend a
    run
```

B WHY REP PROVIDES A ROBUST SIGNAL OF REASONING PROGRESS

We provide a formal interpretation of REP from the perspective of perplexity and show why its change provides a more robust directional signal than gold-answer perplexity alone.

REP as Relative Perplexity. For a candidate answer a, define its length-normalized negative log-likelihood at reasoning step k as

$$\mathcal{N}_k^a = -\frac{1}{L}\sum_{j=1}^{L} \log p_M\left(a_j \mid q, S_{\le k}, r, a_{<j}\right). \tag{8}$$

By definition, $\mathrm{PPL}_k^a = \exp(\mathcal{N}_k^a)$. Therefore, Eq. 1 can be equivalently written as

$$\phi_k = \mathcal{N}_k^{\text{foil}} - \mathcal{N}_k^{\text{gold}} = \log \frac{\mathrm{PPL}_k^{\text{foil}}}{\mathrm{PPL}_k^{\text{gold}}}. \tag{9}$$

Hence, REP measures the perplexity of the foil relative to the gold answer. A larger ϕ_k means that the scoring model finds the gold answer relatively easier to predict than its matched foil.

Robustness to Common-Mode Likelihood Fluctuations. Consider a common-mode perturbation that affects both candidates under the same reasoning prefix. We decompose their observed log perplexities as

$$\log \mathrm{PPL}_k^a = \bar{\mathcal{N}}_k^a + c_k + \epsilon_k^a, \qquad a \in \{\text{gold}, \text{foil}\}, \tag{10}$$

where $\bar{\mathcal{N}}_k^a$ denotes the candidate-specific signal of interest, c_k is a prefix-dependent fluctuation shared by both candidates, and ϵ_k^a is candidate-specific residual noise.

Using gold-answer PPL alone gives

$$\log \mathrm{PPL}_k^{\text{gold}} = \bar{\mathcal{N}}_k^{\text{gold}} + c_k + \epsilon_k^{\text{gold}}, \tag{11}$$

so its variation is directly contaminated by the common term c_k. In contrast, substituting Eq. 10 into Eq. 9 yields

$$\phi_k = \bar{\mathcal{N}}_k^{\text{foil}} - \bar{\mathcal{N}}_k^{\text{gold}} + \epsilon_k^{\text{foil}} - \epsilon_k^{\text{gold}}, \tag{12}$$

where the shared fluctuation c_k cancels exactly. Therefore, when likelihood variation is dominated by common effects shared by the matched gold–foil pair, REP isolates their relative preference more faithfully than either candidate's absolute PPL.

The matched construction of the foil further supports this assumption: the gold and foil share the same reasoning prefix, answer format, and token length, differing only by a minimal answer perturbation. Their nuisance likelihood fluctuations are therefore expected to be strongly correlated, making contrastive cancellation particularly effective.

Directional Meaning of $\Delta\phi_k$. To make the meaning of REP change explicit, define the perplexity improvement of a candidate answer a after incorporating reasoning step S_k as

$$I_k^a = \log \frac{\mathrm{PPL}_{k-1}^a}{\mathrm{PPL}_k^a}, \qquad a \in \{\text{gold}, \text{foil}\}. \tag{13}$$

A positive I_k^a indicates that the candidate becomes easier to predict after adding S_k. From Eq. 1,

$$\begin{aligned}\Delta\phi_k &= \phi_k - \phi_{k-1} \\ &= \log \frac{\mathrm{PPL}_k^{\mathrm{foil}}}{\mathrm{PPL}_k^{\mathrm{gold}}} - \log \frac{\mathrm{PPL}_{k-1}^{\mathrm{foil}}}{\mathrm{PPL}_{k-1}^{\mathrm{gold}}} \\ &= \log \frac{\mathrm{PPL}_{k-1}^{\mathrm{gold}}}{\mathrm{PPL}_k^{\mathrm{gold}}} - \log \frac{\mathrm{PPL}_{k-1}^{\mathrm{foil}}}{\mathrm{PPL}_k^{\mathrm{foil}}} \\ &= I_k^{\mathrm{gold}} - I_k^{\mathrm{foil}}. \end{aligned} \tag{14}$$

Therefore, $\Delta\phi_k > 0$ means that the gold answer improves in predictability more than its matched foil, while $\Delta\phi_k < 0$ means that the foil improves relatively more. Importantly, a positive $\Delta\phi_k$ does not require the foil perplexity to increase; both candidates may become easier to predict, as long as the gold answer improves by a larger relative margin. Thus, $\Delta\phi_k$ directly measures whether reasoning step S_k strengthens or weakens the model's relative support for the target answer.

C Prompt for Gemini-based Segment Annotation

We use the following prompt to instruct Gemini-3.6-Flash to annotate whether each reasoning step should contribute to SFT supervision.

```
You are helping curate training data for supervised fine-tuning (SFT)
of a smaller student reasoning model.  You are given a math PROBLEM,
its correct FINAL ANSWER, and a reasoning trace that has been split
into indexed SEGMENTS. For EACH segment decide whether it is NECESSARY
for the student to learn.
label 1 = NECESSARY: it carries an essential logical step, key
insight, correct setup, or computation that genuinely advances the
derivation toward the answer.
label 0 = UNNECESSARY: it is redundant, repetitive, filler, rambling
self-talk, hesitation, backtracking that leads nowhere, restating
known facts, or an ERRONEOUS step that does not help (or could
mislead) the final correct solution.
Judge each segment on its marginal contribution; a fluent trace can
still contain many skippable segments.

=== PROBLEM ===
<question>

=== CORRECT FINAL ANSWER ===
<answer>

=== REASONING SEGMENTS (indices 0..N-1) ===
[SEGMENT 0]
<segment 0 text>
[SEGMENT 1]
<segment 1 text>
...
[SEGMENT N-1]
<segment N-1 text>

=== TASK ===
Label all N segments (indices 0..N-1).
OUTPUT REQUIREMENT: Return ONLY a single JSON object, no prose, no
markdown fences, with EXACTLY two keys:
"labels":  an object mapping every segment index (as string) to 0 or
1.
"reasons":  an object mapping every segment index (as string) to a
concise (<=25 words) English reason.
You MUST include an entry for every segment index shown.
Example:
{"labels":  {"0":  1, "1":  0}, "reasons":  {"0":  "sets up the key
substitution", "1":  "restates prior step, redundant"}}
```

D CASE STUDY: ORIGINAL PROBLEM AND FULL REASONING TRAJECTORY

This appendix provides the original problem and the complete long-form reasoning trajectory of LIMO #539 used in our case study. We preserve the original reasoning content verbatim and organize it according to the same reasoning-step segmentation used in our analysis.

D.1 ORIGINAL PROBLEM

```
There is a set of 1000 switches, each of which has four positions, called $A, B, C$ , and
     $D$ .  When the position of any switch changes, it is only from $A$ to $B$ , from $B$ to
      $C$ , from $C$ to $D$ , or from $D$ to $A$ .  Initially each switch is in position $A$
     .  The switches are labeled with the 1000 different integers $(2^{x})(3^{y})(5^{z})$ ,
     where $x, y$ , and $z$ take on the values $0, 1, \ldots, 9$ .  At step $i$ of a 1000-
     step process, the $i$ -th switch is advanced one step, and so are all the other switches
      whose labels divide the label on the $i$ -th switch.  After step 1000 has been
     completed, how many switches will be in position $A$ ?
```

Gold Answer. 650

D.2 FULL REASONING TRAJECTORY

Reasoning Step 0.

```
Okay, let's try to tackle this problem. So, we have 1000 switches, each labeled with a
     number of the form 2^x * 3^y * 5^z where x, y, z range from 0 to 9. Each switch starts
     in position A, and every time a switch is toggled, it cycles through A→B→C→D→A and
     so on.

The process is a 1000-step process where in step i, we advance the i-th switch and all
     switches with labels that divide the label of the i-th switch. After completing all 1000
      steps, we need to find how many switches are back in position A.

First, let's make sure I understand the setup correctly. Each switch is uniquely identified
     by (x, y, z) where each of x, y, z is between 0 and 9 inclusive. The label is 2^x * 3^y
     * 5^z. So, the number of switches is 10*10*10 = 1000, which checks out.

In each step i, we toggle the i-th switch and all switches with labels that divide the i-th
     switch's label. So, for example, if the i-th switch has label 2^3 * 3^2 * 5^1, then any
     switch with label 2^a * 3^b * 5^c where a ≤ 3, b ≤ 2, c ≤ 1 will be toggled. Each
     toggle moves the switch one position forward (A→B, B→C, etc.), and after four toggles,
      it cycles back to A.

Therefore, the final position of a switch depends on how many times it was toggled modulo 4.
      If it was toggled 0 times mod 4, it's in A; 1 mod 4, it's in B; etc. So, switches that
     are toggled a multiple of 4 times will be back in A.

Hence, our problem reduces to: For each switch, count the number of multiples of its label
     in the list of 1000 labels, then compute how many switches have this count ≡ 0 mod 4.
```

Reasoning Step 1.

```


Wait, actually, not quite. Because in each step i, we toggle all switches whose labels
     divide the label of switch i. So, for a given switch j, the number of times it is
     toggled is equal to the number of switches i such that the label of j divides the label
     of i. In other words, the number of multiples of label_j in the set of labels.

Therefore, for each switch j, we need to count how many labels in the set are multiples of
     label_j, and then see if that number is divisible by 4. If yes, then switch j is back in
      A.

Therefore, the problem is equivalent to: For each label in the set, compute the number of
     its multiples in the entire set, then count how many labels have this number ≡ 0 mod 4.

So, first step is to compute, for each switch j (with label L_j), the number of switches i (
     with label L_i) such that L_j divides L_i. Let's denote this count as C_j. Then, we need
      the number of j for which C_j ≡ 0 mod 4.

So, how do we compute C_j for each label L_j?

Given that each label is of the form 2^x * 3^y * 5^z, where x, y, z ∈ {0,1,...,9}. So, for a
      given label L_j = 2^a * 3^b * 5^c, the number of multiples of L_j in the set is equal
     to the number of labels L_i = 2^x * 3^y * 5^z such that x ≥ a, y ≥ b, z ≥ c. Since
     each exponent in L_i must be at least the corresponding exponent in L_j.
```

```
Therefore, for each switch j with exponents (a, b, c), the number of multiples C_j is (10 -
     a) * (10 - b) * (10 - c). Because for each exponent x, y, z, they can range from a to 9,
      inclusive. So, the number of possible exponents for 2 is 10 - a (since x can be a, a+1,
      ..., 9), similarly for y and z. Hence, multiplying them gives the total number of
     multiples.

Therefore, C_j = (10 - a) * (10 - b) * (10 - c), where (a, b, c) are the exponents of the
     prime factors of L_j.

Therefore, we need to compute for each triple (a, b, c), the value (10 - a)(10 - b)(10 - c),
      then check if that number is divisible by 4. Then, count the number of triples (a, b, c
     ) where this is true.

So, the problem reduces to counting the number of triples (a, b, c) where (10 - a)(10 - b)
     (10 - c) ≡ 0 mod 4.

But actually, since (10 - a) can range from 1 (when a=9) to 10 (when a=0), same for (10 - b)
      and (10 - c). So, each of these terms is an integer between 1 and 10. Therefore, the
     product (10 - a)(10 - b)(10 - c) is a number between 1 and 1000.

We need to find the number of triples (a, b, c) such that (10 - a)(10 - b)(10 - c) is
     divisible by 4, i.e., ≡ 0 mod 4.
```

Reasoning Step 2.

```
Wait, but actually, no. Because the count C_j = (10 - a)(10 - b)(10 - c). We need C_j ≡ 0
     mod 4. So, we need the product (10 - a)(10 - b)(10 - c) ≡ 0 mod 4.

But to clarify, (10 - a)(10 - b)(10 - c) must be congruent to 0 modulo 4. So, the product
     must be divisible by 4.

Therefore, for each triple (a, b, c), compute (10 - a)(10 - b)(10 - c) and check if it's
     divisible by 4. The total number of such triples is the answer.

So, the problem becomes: How many triples (a, b, c) with a, b, c ∈ {0,1,...,9} are such that
      (10 - a)(10 - b)(10 - c) is divisible by 4?
```

Reasoning Step 3.

```
Alternatively, since a, b, c ∈ {0,...,9}, 10 - a ∈ {1,...,10}, same for 10 - b and 10 - c.

Therefore, let's let x = 10 - a, y = 10 - b, z = 10 - c, so x, y, z ∈ {1, 2, ..., 10}. Then,
      we need to count the number of triples (x, y, z) where x, y, z ∈ {1, ..., 10} and xyz
     ≡ 0 mod 4.

But the original problem counts each (a, b, c) once, so since x = 10 - a, a is determined by
      x: a = 10 - x. Similarly for y and z. Therefore, each (x, y, z) corresponds to exactly
     one (a, b, c). So, the number of such triples is the same as the number of (x, y, z)
     with x, y, z ∈ {1, ..., 10} and xyz divisible by 4.

Therefore, our problem is equivalent to: How many triples (x, y, z) with x, y, z ∈ {1, 2,
     ..., 10} have xyz divisible by 4?

So, the answer will be the number of such triples, which is the same as the number of
     switches that are toggled a multiple of 4 times, hence ending up back in position A.

So, the problem simplifies to: How many triples (x, y, z) with each component from 1 to 10
     inclusive have their product divisible by 4?

To solve this, perhaps it's easier to compute the total number of triples (10*10*10 = 1000)
     and subtract the number of triples where the product is not divisible by 4. So, total -
     number of triples with product ≡ 1, 2, or 3 mod 4. But since 4 divides the product if
     and only if the product has at least two factors of 2. So, equivalently, the product is
     divisible by 4 iff the product has at least two 2s in its prime factorization.

Therefore, we can think in terms of the number of triples (x, y, z) where the exponents of 2
      in x, y, z sum to at least 2. But perhaps another approach is better.
```

Reasoning Step 4.

```
Alternatively, note that in the numbers 1 to 10, the even numbers are 2, 4, 6, 8, 10. The
     odd numbers are 1, 3, 5, 7, 9. Let's categorize each number as:

- Odd: 1, 3, 5, 7, 9 (numbers not divisible by 2)
```

```
- Even but not divisible by 4: 2, 6, 10 (numbers divisible by 2 but not by 4)
- Divisible by 4: 4, 8 (numbers divisible by 4)

Let me check:

- 1: odd
- 2: even, not divisible by 4
- 3: odd
- 4: divisible by 4
- 5: odd
- 6: even, not divisible by 4
- 7: odd
- 8: divisible by 4
- 9: odd
- 10: even, not divisible by 4

So, counts:

- Odd numbers (O): 5
- Even but not divisible by 4 (E): 3 (numbers 2,6,10)
- Divisible by 4 (F): 2 (numbers 4,8)

Therefore, for each x, y, z, they can be O, E, or F.

We need to compute the number of triples (x, y, z) where the product xyz is divisible by 4.
     Let's think in terms of the contributions from each component.

The product xyz will have a certain number of factors of 2. For the product to be divisible
     by 4, the total number of factors of 2 must be at least 2.

Each O contributes 0 factors of 2.

Each E contributes 1 factor of 2 (since they are divisible by 2 but not 4).

Each F contributes 2 factors of 2 (since they are divisible by 4).

Therefore, the total number of factors of 2 in the product is:

If a component is O: 0

If E: 1

If F: 2

So, for each triple (X, Y, Z), where each is O, E, or F, the total number of factors of 2 is
     :

Total = factors from X + factors from Y + factors from Z.

We need Total ≥ 2.

Therefore, the number of triples where Total ≥ 2 is equal to the total number of triples
     minus the number of triples where Total < 2, which is Total = 0 or 1.

Total = 0: all three components are O. So, O, O, O.

Total = 1: exactly one component is E or F contributing 1 or 2 factors, but total factors
     sum to 1. But wait, if a component is F, it contributes 2 factors, which would already
     make Total ≥ 2. So, to get Total = 1, we need exactly one of the components to be E (
     contributing 1 factor) and the rest to be O. Because if any component is F, it would add
      2, which would overshoot.

Therefore, Total = 1 is only possible when exactly one component is E and the other two are
     O.

Therefore, the number of triples where Total < 2 is:

Number of O,O,O triples + number of triples with exactly one E and two O's.

Compute these:

Number of O,O,O triples: since there are 5 O's in each position. So, 5*5*5 = 125.

Number of triples with exactly one E and two O's: For each position (x, y, z), we can have E
      in x, O in y and z; E in y, O in x and z; or E in z, O in x and y. For each case, the
     number is (number of E choices) * (number of O choices)^2. There are 3 E's, 5 O's. So,
     for each single position being E, it's 3 * 5 * 5. Since there are 3 positions, total is
     3 * 3 * 5 * 5 = 225.

Therefore, total number of triples with Total < 2 is 125 + 225 = 350.
```

```
Therefore, the number of triples with Total ≥ 2 is total triples - 350 = 1000 - 350 = 650.
```

Reasoning Step 5.

```
Wait, but hold on. Wait, total triples are 10*10*10 = 1000. Wait, no, in our transformed
     problem, x, y, z are from 1 to 10, which corresponds to the original 1000 switches.
     Therefore, the total number of triples is indeed 1000. So, 1000 - 350 = 650. Therefore,
     650 switches are toggled a number of times divisible by 4, hence ending in position A.
```

Reasoning Step 6.

```
But wait, hold on. Wait, no. Wait, in the transformed problem, we have x, y, z each from 1
     to 10, corresponding to (10 - a), (10 - b), (10 - c). So, the count of triples (x, y, z)
      where xyz is divisible by 4 is 650, which would mean 650 switches are toggled a number
     of times ≡ 0 mod 4, hence in position A.

But the answer is 650? Let me verify.
```

Reasoning Step 7.

```
Wait, let's check if the logic is correct.

First, in the transformed variables, we have x, y, z from 1 to 10, and we need xyz divisible
      by 4. So, we categorize numbers 1-10 into O, E, F as before.

Total triples: 1000.

Number of bad triples (Total < 2 factors of 2):

- All O: 5*5*5 = 125

- Exactly one E and two O's: 3 positions * 3 E numbers * 5 O * 5 O = 3*3*25 = 225

Total bad: 125 + 225 = 350

Thus, good triples: 1000 - 350 = 650

Therefore, the answer is 650.

But let me think again. Wait, but hold on. When we have a component that is F (divisible by
     4), it contributes 2 factors of 2. So, if there's at least one F in the triple, then the
      total factors of 2 is at least 2. Similarly, if there are two E's, since each E
     contributes 1 factor of 2, two E's would give 2 factors. Similarly, one E and one F
     would give 3 factors, etc.

Therefore, the number of triples with product divisible by 4 is:

- All triples with at least one F: Let's compute this.

Number of triples with at least one F: total triples - triples with no F.

Triples with no F: all components are either O or E. There are (5 + 3) = 8 choices per
     component. So, 8*8*8 = 512. But within these, some may still have products divisible by
     4. Wait, no. Wait, if there are no Fs, then all components are either O or E. So, the
     factors of 2 come only from E's, which contribute 1 each. So, to get the product
     divisible by 4, we need at least two E's. So, in the absence of F's, the number of
     triples with product divisible by 4 is the number of triples with at least two E's.
```

Reasoning Step 8.

```
Wait, this seems conflicting with the previous approach. Let me check.
```

Reasoning Step 9.

```
Alternatively, maybe I should use inclusion-exclusion properly.

Total number of triples with product divisible by 4 = number of triples with at least two
     factors of 2.

Which can happen in two distinct cases:
```

```
1. At least one F (each F gives two factors), so even if the rest are O, the total factors
     would be at least 2.

2. No F's, but at least two E's (each E gives one factor, so two E's give two factors).

Therefore, total number of good triples = (number of triples with at least one F) + (number
     of triples with no F's and at least two E's).

Compute each part.

First, number of triples with at least one F: total triples - triples with no F's. As before
     , triples with no F's: each component can be O or E, 8 choices per component. So 8^3 =
     512. Therefore, triples with at least one F = 1000 - 512 = 488.

Second, number of triples with no F's and at least two E's: Within the 512 triples with no F
     's, how many have at least two E's.

In triples with no F's, each component is O or E.

Number of such triples with at least two E's = total triples with no F's - triples with
     fewer than two E's.

Which is 512 - [triples with zero E's + triples with one E].

Number of triples with zero E's (all O's): 5^3 = 125.

Number of triples with one E: 3 positions * 3 E's * 5^2 O's = 3*3*25 = 225.

Therefore, triples with at least two E's and no F's: 512 - 125 - 225 = 512 - 350 = 162.

Therefore, total good triples = 488 (with at least one F) + 162 (with no F's but two E's) =
     650.

Same as before. Therefore, 650 is the correct count. Therefore, the answer is 650.
```

Reasoning Step 10.

```
But wait, let me cross-verify with another method.

Alternative approach: For each triple (x, y, z), the product xyz is divisible by 4 iff:

Either:

- At least two of x, y, z are even, or

- At least one of x, y, z is divisible by 4.
```

Reasoning Step 11.

```
Wait, actually, not exactly. Let's think:

A number is divisible by 4 if:

- It has at least two factors of 2. So, in the product xyz, the total number of factors of 2
      must be at least two.

Each even number contributes at least one factor of 2, and numbers divisible by 4 contribute
      two.

So, the product's number of factors of 2 is the sum over each component's factors of 2.

Therefore, possible scenarios:

1. At least one component is divisible by 4 (i.e., contributes 2 factors), and the rest can
     be anything. Then, total factors is at least 2.

2. No components are divisible by 4, but at least two components are even (i.e., each
     contributes 1 factor), totaling 2.

Therefore, total good triples are:

Number of triples with at least one divisible by 4 + number of triples with no divisible by
     4 but at least two even.

Compute each:
```

```
First, number of triples with at least one divisible by 4: as before, 1000 - 512 = 488.

Second, number of triples with no divisible by 4 but at least two even: In this case, all
     components are either O or E (not F). So, components can be O or E.

Number of such triples: 8^3 = 512 (as before).

Within these, number of triples with at least two E's: which is 162, as before.

Therefore, total is 488 + 162 = 650. Same result.
```

Reasoning Step 12.

```
Alternatively, perhaps even another way. Let's compute the number of triples where product
     is not divisible by 4, i.e., product is 1 or 2 mod 4.

So, product is odd (1 mod 4) or product is 2 mod 4.

Product is odd: all components are odd. Number of such triples: 5^3 = 125.

Product is 2 mod 4: exactly one component is 2 mod 4 and the rest are odd. But wait, in our
     numbers, E is numbers divisible by 2 but not 4, i.e., 2 mod 4 or 6 mod 4 (which is 2 mod
      4) or 10 mod 4 (which is 2 mod 4). So, E numbers are 2, 6, 10, which are all 2 mod 4.

Therefore, if exactly one component is E (2 mod 4) and the rest are O (1 mod 4), then the
     product is 2*1*1 = 2 mod 4. So, number of such triples is 3 positions * 3 E's * 5 O's *
     5 O's = 3*3*5*5 = 225.

Therefore, total bad triples: 125 + 225 = 350. Hence, good triples: 1000 - 350 = 650. Same
     answer.

Therefore, regardless of the approach, we get 650. So, the answer should be 650.

But let me verify with an example.

Take a smaller case. Suppose we have numbers from 1 to 2 instead of 1 to 10, with similar
     rules. Let's see if the formula works.

But maybe that's overcomplicating. Alternatively, check if the counts make sense.

Given that 5 numbers are odd, 3 are E (2 mod 4), and 2 are F (0 mod 4). So, total 10 numbers
     .

Compute the number of triples (x, y, z) with product divisible by 4.

Using the formula, total triples 1000, bad triples 350, good triples 650. Hmm. Alternatively
     , perhaps we can compute the exact number.
```

Reasoning Step 13.

```
Alternatively, think of the multiplicative contributions.

For each position x, y, z:

- The probability that x is O is 5/10 = 1/2.

- The probability that x is E is 3/10.

- The probability that x is F is 2/10 = 1/5.

But since we are counting exact numbers, not probabilities, but maybe using generating
     functions.

But perhaps overkill. Since all methods so far point to 650, and the logic seems consistent,
      I think 650 is the correct answer.

But to be thorough, let's check a specific case.

Take a switch with label 1 (i.e., x=y=z=0). Then, the number of multiples is (10 - 0)*(10 -
     0)*(10 - 0) = 1000. So, 1000 toggles. 1000 mod 4 is 0, so this switch would be in A. But
      according to our count, does (x, y, z) = (10, 10, 10) correspond to (a, b, c) = (0, 0,
     0). Then, the product xyz = 10*10*10 = 1000. 1000 mod 4 is 0, so it's counted as a good
     triple. Therefore, this switch is included in the 650.
```

```
Similarly, take a switch with label 2^9 * 3^9 * 5^9. Its number of multiples is (10 - 9)*(10
     - 9)*(10 - 9) = 1*1*1 = 1. So, 1 toggle. 1 mod 4 is 1, so it's in position B. Therefore
    , not in A. So, this switch is excluded from the 650, which is correct.
```

Reasoning Step 14.

```
Another test case: take a switch with label 2^8 (i.e., x=8, y=0, z=0). Then, the number of
    multiples is (10 - 8)*(10 - 0)*(10 - 0) = 2*10*10 = 200. 200 mod 4 is 0, so this switch
    would be in A. Indeed, in our transformed variables, x = 10 - a = 2, y = 10 - b = 10, z
    = 10 - c = 10. The product xyz = 2*10*10 = 200. 200 is divisible by 4, so this triple is
     counted in the 650.
```

Reasoning Step 15.

```
Another test case: take a switch with label 2^1 * 3^0 * 5^0 (i.e., x=1, y=0, z=0). Then, the
     number of multiples is (10 - 1)*(10 - 0)*(10 - 0) = 9*10*10 = 900. 900 mod 4 is 900 / 4
     = 225, remainder 0. So, 900 mod 4 = 0. Therefore, this switch is in A. In transformed
    variables, x = 9, y = 10, z = 10. The product is 9*10*10 = 900, which is divisible by 4.
     Wait, 9 is odd, 10 is even. 9*10*10 = 9*100 = 900. 900 / 4 = 225. Yes, divisible by 4.
    So, even though x is odd (9), the product is still divisible by 4 because y and z are
    both even. So, this is accounted for in our count. Since y and z are 10, which are E
    numbers (divisible by 2 but not 4). So, each contributes 1 factor of 2. Therefore, total
     factors of 2 is 2, so product is divisible by 4. Hence, included in the 650.

Therefore, all test cases check out.

Therefore, I'm confident that the answer is 650.

**Final Answer**
\boxed{650}
```