TERMINATOR: Learning Optimal Exit Points for Early Stopping in Chain-of-Thought Reasoning

Alliot Nagle [1] **Jakhongir Saydaliev** [2] **Dhia Garbaya** [2 3] **Michael Gastpar**[† 2]
Ashok Vardhan Makkuva[† 4] **Hyeji Kim**[† 1]

Abstract

Large Reasoning Models (LRMs) achieve impressive performance on complex reasoning tasks via Chain-of-Thought (CoT) reasoning, which enables them to generate intermediate thinking tokens before arriving at the final answer. However, LRMs often suffer from significant *overthinking*, spending excessive compute time even after the answer is generated early on. Prior work has identified the existence of an optimal reasoning length such that truncating reasoning at this point significantly shortens CoT outputs with virtually no change in performance. However, determining optimal CoT lengths for practical datasets is highly non-trivial as they are fully task and model-dependent. In this paper, we precisely address this and design TERMINATOR, an early-exit strategy for LRMs at inference to mitigate overthinking. The central idea underpinning TERMINATOR is that the first arrival of an LRM's final answer is often predictable, and we leverage these first answer positions to create a novel dataset of optimal reasoning lengths to train TERMINATOR. Powered by this approach, TERMINATOR achieves significant reductions in CoT lengths of 14%–55% on average across four challenging practical datasets: MATH-500, AIME 2025, HumanEval, and GPQA, while outperforming current state-of-the-art methods and reducing inference latency by more than $2\times$ compared to the original LRM. Project page: https://terminator-llm.github.io/

[1]UT Austin, US [2]EPFL, Switzerland [3]ENS Paris-Saclay, France [4]Télécom Paris (IP Paris), France. Correspondence to: Alliot Nagle <acnagle@utexas.edu>.

Workshop on Resource-Adaptive Foundation Model Inference (AdaptFM) at the 43^{rd} International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026.

1. Introduction

The advent of Large Reasoning Models (LRMs) has proven itself to be a critical next step for Large Language Models (LLMs) to surpass human-level performance. LRMs use test-time compute to "think" through a problem before answering, an approach that has led to significant performance gains across many challenging tasks (OpenAI, 2024). However, this improvement does not come for free, as an LRM will generate thousands of additional thinking tokens to solve a single problem, compared to its non-reasoning counterparts (Guo et al., 2025). Worse yet, LRMs spend a significant amount of their reasoning tokens double-checking their work and exploring different solutions when they have already generated the final answer, that they will eventually settle on, much earlier in the CoT, a phenomenon known as *overthinking* (Luo et al., 2025; Chen et al., 2025). Prior work has shown that the length of a CoT can be reduced by 50% or more on average with little drop in accuracy (Kang et al., 2025a; Zhang et al., 2025b; Yang et al., 2025b), demonstrating the extent to which compute is wasted during LRM inference.

Given that reasoning can be wasteful, a natural question to ask is, *for any given accuracy, does there exist an optimal reasoning length?* Previous works have shown that LRM performance, as a function of reasoning length, gradually increases, peaks, and then decreases, suggesting the existence of an optimal reasoning length (Wu et al., 2025b; Lee et al., 2025). Additionally, some recent works propose novel RL-training algorithms to fine-tune LRMs to produce shorter CoTs (Luo et al., 2025; Lou et al., 2025; Gao et al., 2025; Yi et al., 2025; Shrivastava et al., 2025) and establish the Pareto frontier for those methods, showing that gaps still exist between them (Gao et al., 2025). While these works focus on retraining an LRM, inference-time methods such as DEER (Yang et al., 2025b) enable early termination of reasoning without retraining it. However, for practical tasks none of these methods either determines or utilizes the optimal-length reasoning, which in fact provides the best possible reduction in CoT length.

In this paper, we precisely address this by introducing the novel notion of *hindsight-optimal reasoning length*

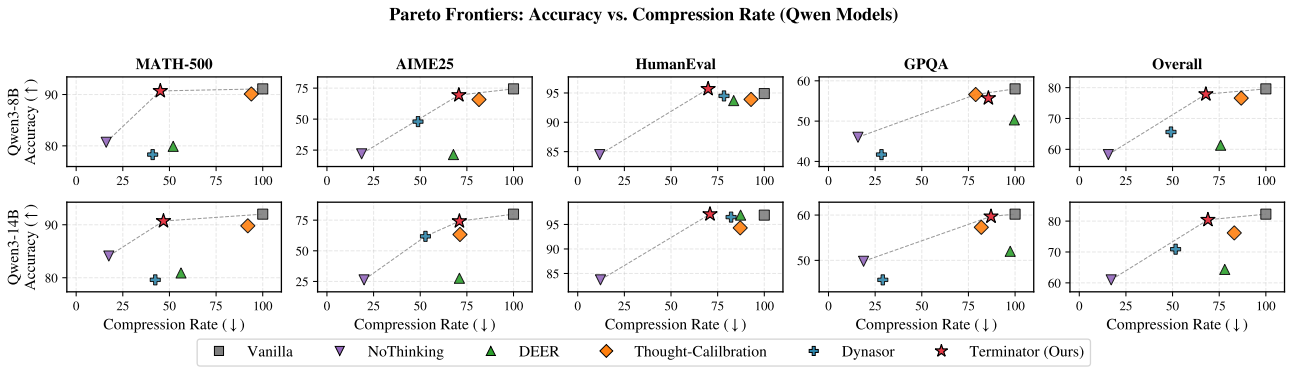

Figure 1. **Pareto Frontier.** TERMINATOR defines the Pareto frontier on 7 of the 8 (LRM, benchmark) pairings, outperforming prior work.ach point represents a method's accuracy and compression rate, with lower compression rates indicating greater token savings and hence compute. The dashed line traces the Pareto frontier connecting non-dominated solutions. We refer to Sec. 5.1, and Fig. 15 and Table 7 in App. B, for more details, including results on two Mistral LRMs.

(Sec. 2.2): given a reasoning task, in hindsight, what is the fewest number of tokens that an LRM needs to generate before providing the same answer it would have provided without shortened reasoning? Namely, we mark the first logical arrival, as opposed to any other occurrence, of the LRM's final answer as the hindsight-optimal exiting position. Leveraging this notion, we design a novel inference-time early-exit algorithm TERMINATOR that significantly outperforms current state-of-the-art methods in reductions to CoT lengths on challenging practical datasets (Fig. 3). In particular, TERMINATOR capitalizes on the fact that the first arrival of the final answer is (1) marked by a distinctive shift in the LRM's token-level confidence and token usage distribution, and (2) can be used as a signal to train a binary probe classifier for effective early-exiting during reasoning.

Main Contributions. In summary, we make the following contributions:

- We introduce the novel notion of hindsight-optimal reasoning, using which we show that the first arrival of an LRM's final answer is marked by observable and meaningful signals (Fig. 2). To the best of our knowledge, this is the first such analysis of its kind.
- We design TERMINATOR, a novel, lightweight, early-exit algorithm for LRMs that is trained on optimal-length CoTs (Sec. 4), resulting in more than a 2× reduction in inference latency compared to the original LRM (Sec. 5.1).
- We introduce a robust pipeline for identifying the first arrival of the final answer in CoTs, using which we construct a novel optimal-length CoT training dataset (App. A.1).

2. Preliminaries

2.1. Notation

A Large Reasoning Model $\texttt{LRM}$ takes as input the prompt sequence $\boldsymbol{x} = (x_1, x_2, x_3, \ldots, x_L)$ and produces two outputs $\boldsymbol{r}$ and $\boldsymbol{s}$ auto-regressively (Fig. 3). Here $\boldsymbol{r} = (r_1, r_2, r_3, \ldots, r_M)$ is the CoT sequence generated during the thinking stage, i.e. $r_i = \texttt{LRM}(\boldsymbol{x}, \boldsymbol{r}_{<i})$ for $i \in [M] \triangleq \{1, \ldots, M\}$, and $\boldsymbol{s} = (s_1, s_2, s_3, \ldots, s_N)$ is the solution that summarizes this CoT and contains a final answer $\hat{\boldsymbol{a}}$, which could be a single numerical answer, a math expression, code, a multiple-choice option, etc. Here $s_j = \texttt{LRM}(\boldsymbol{x}, \boldsymbol{r}, \boldsymbol{s}_{<j})$ for $j \in [N]$. Note that the final answer $\hat{\boldsymbol{a}}$ is separate from the ground-truth answer $\boldsymbol{a}$; they may or may not be in agreement with each other. Throughout the paper, $\hat{\boldsymbol{a}}$ always refers to the final answer of the full CoT, not the final answer generated after exiting a CoT early. Furthermore, when referring to $\hat{\boldsymbol{a}}$ with respect to its position in a CoT, we always mean the *earliest logical arrival* of $\hat{\boldsymbol{a}}$ unless stated otherwise explicitly. By the earliest logical arrival of $\hat{\boldsymbol{a}}$, we are referring to the sequence of logical steps in the CoT that yields the final answer $\hat{\boldsymbol{a}}$ for the first time. For any early-exit strategy, a key metric to gauge its performance is the per-sample compression rate (CR): $\frac{M_{\text{early}}}{M}$, where $M_{\text{early}} \in [M]$ is the token index of early exit in $\boldsymbol{r}$. Accuracy (Acc) measures the proportion of problems where the correct answer is produced.

2.2. Hindsight-optimality

We now formally define our novel notion of hindsight-optimality. Given an input prompt $\boldsymbol{x} \in \mathcal{X}^L$ of length L over a vocabulary $\mathcal{X}$, an LRM $\texttt{LRM}$ generates a corresponding CoT $\boldsymbol{r} \in \mathcal{X}^M$ and solution $\boldsymbol{s} \in \mathcal{X}^N$, where the solution contains a final answer $\hat{\boldsymbol{a}} \in \mathcal{X}$. The hindsight-optimal reasoning length (HORL) is defined as the earliest position in the completed CoT at which the final answer $\hat{\boldsymbol{a}}$ has been

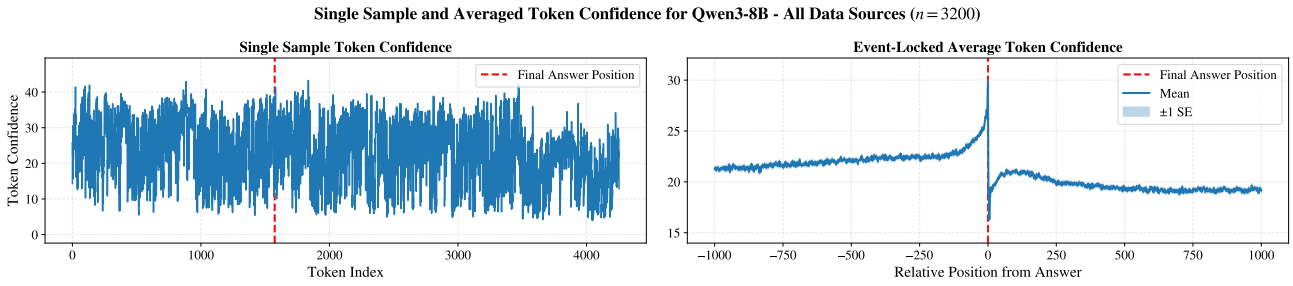

Figure 2. **Event-Locked Averaging of Token-Confidence.** Event-locked averaging shows a consistent agreement on spiking behavior at the answer position in each CoT, but disagrees elsewhere. On the other hand, this phenomenon is not readily observable in the single-sample case. The figure on the **left** shows the Token-Confidence (Fu et al., 2025b) trajectory throughout reasoning for a single, randomly selected sample; the figure on the **right** shows the effect of *event-locked averaging* on the position of the first arrival of the final answer across all CoTs. The 3200 CoTs used are a random subset of our training set, which combines AIME (1983–2024), MATH, OpenCoder-SFT, and OpenScience. Figs. 16 to 19 in App. B show similar trends for each dataset separately, and report log-probabilities in addition to Token-Confidence. The standard error (SE) is shown as a shaded region and becomes clearer when zoomed in.

logically reached. Mathematically,

$$\mathsf{HORL}(\boldsymbol{x}, \boldsymbol{r}, \boldsymbol{s}, \hat{\boldsymbol{a}}) \triangleq \min\left\{ i \in [M] : \begin{smallmatrix} \boldsymbol{r}_{\leqslant i} \text{ contains the earliest} \\ \text{logical arrival of } \hat{\boldsymbol{a}} \end{smallmatrix} \right\}. \tag{1}$$

Here, $\boldsymbol{r}_{\leqslant i}$ is said to contain the earliest logical arrival of $\hat{\boldsymbol{a}}$ if, by position i, the sequence of reasoning steps in $\boldsymbol{r}$ has produced the first derivation of the final answer $\hat{\boldsymbol{a}}$. Thus, HORL is a retrospective property of the realized CoT $\boldsymbol{r}$ and final answer $\hat{\boldsymbol{a}}$.

2.3. Token-Confidence

Our analytical experiments require a measure of LRM's confidence during the generation of a CoT. To this end, we use the Token-Confidence metric, that gauges the uncertainty of a chosen token. Mathematically, for every $i \in [M]$, the corresponding Token-Confidence C_i is defined as

$$C_i \triangleq -\frac{1}{K} \sum_{k \in \mathcal{T}_K(i)} \log \mathbb{P}_{\texttt{LRM}}\left(r_i = k \mid \boldsymbol{x}, \boldsymbol{r}_{<i}\right), \tag{2}$$

where $\mathbb{P}_{\texttt{LRM}}\left(r_i = \cdot \mid \boldsymbol{x}, \boldsymbol{r}_{<i}\right)$ is the LRM prediction probability at position i and $\mathcal{T}_K(i) \triangleq \text{Top-}K\left[\mathbb{P}_{\texttt{LRM}}\left(r_i = \cdot \mid \boldsymbol{x}, \boldsymbol{r}_{<i}\right)\right]$ is the set of vocabulary tokens corresponding to the Top-K probabilities. In other words, Token-Confidence is the average (negative) log-probability across the Top-K probabilities (we set $K = 20$ in our experiments). The higher it is, the more confident the model is in its predictions.

This measure is based on the Self-Certainty metric (Kang et al., 2025b), computed as the KL-divergence between the uniform distribution and the token distribution, and is based on the following idea: the higher the confidence of the model, the further its predictions should be from the uniform distribution. We also note that, while Token-level log-probabilities are commonly used as a proxy for confidence, we prefer the Token-Confidence measure (Fu et al., 2025b) here, as it is principled and produces less noise. Both are used, and the same conclusions can be drawn using either.

3. Motivation

Shortly after the breakthrough of LRMs, it was observed that they exhibit an overthinking phenomenon where, despite arriving at the correct answer, they continue to consider alternative solution paths, possibly leading to other incorrect answers (Chen et al., 2025; Luo et al., 2025). While LRMs achieve greatly improved performance over their non-reasoning counterparts, they do so at a much higher inference-time cost: up to thousands of additional tokens are generated to form the CoT before arriving at the final solution (Guo et al., 2025). Many follow-up works have observed the same overthinking phenomenon and developed methods to mitigate wasteful token expenditure (Zhang et al., 2025b; Liu & Wang, 2025; Wu et al., 2025a; Zhang et al., 2025a).

Towards designing an optimal early-exit strategy to stymie overthinking, we build upon the following key observation: once an LRM generates a CoT $\boldsymbol{r}$ and a final solution $\boldsymbol{s}$, we can observe the final answer $\hat{\boldsymbol{a}}$. Then, *in hindsight*, we can determine precisely where the LRM should have exited the CoT to avoid wasting tokens, i.e. HORL, and instead generate the final solution. To this end, we first only need to check for $\hat{\boldsymbol{a}}$, not $\boldsymbol{a}$, in $\boldsymbol{r}$, since the LRM may never even have generated the correct (ground-truth) answer and thus may not exist in $\boldsymbol{r}$. Second, by choosing to terminate reasoning after the arrival of $\hat{\boldsymbol{a}}$ in $\boldsymbol{r}$, all steps that are useful to arriving at $\hat{\boldsymbol{a}}$ are kept, and anything after is skipped as it is redundant.

While the above procedure requires the explicit knowledge of $\hat{\boldsymbol{a}}$ to check for its arrival, *are there meaningful markers to implicitly detect its arrival?*

Detecting the Answer Early. We analyze trends in Token-Confidence during CoT reasoning on AIME (1983–2024), MATH, OpenCoder-SFT, and OpenScience, as shown in Figs. 2 and 11. Fig. 2 reveals a sharp transition in Token-Confidence around the first occurrence of $\hat{\boldsymbol{a}}$ in event-locked averages, formed by aligning each CoT's first $\hat{\boldsymbol{a}}$ position to

0 and averaging across samples. Because these CoTs span math, science, and coding datasets, this transition suggests a consistent cross-domain signal; per-dataset versions appear in Figs. 16 to 19. In addition, we observe a bias in usage frequency of certain tokens in the CoT before and after the first arrival of $\hat{\boldsymbol{a}}$, which we discuss in detail in App. B.1.

Moving to Online Inference and Challenges. While these results strongly indicate the early arrival of $\hat{\boldsymbol{a}}$, using them during online inference remains a challenge. In the case of Fig. 2, event-locked averaging requires multiple CoTs to be generated simultaneously, each with reasonable estimates of the position of the answer. Under those circumstances, the spiking behavior will emerge. But attaining a reasonable estimate of the answer position for a single CoT during inference is the original problem we are tasked with. While applying the event-locked averaging signal to online inference is limited, it does indicate that an underlying trend can be extracted.

Similarly, each dot in Fig. 11 requires full knowledge of each $\boldsymbol{r}$ and its position of $\hat{\boldsymbol{a}}$ so that the rates can be calculated accordingly. Again, these results show a shift in the usage frequency of certain tokens before and after the first occurrence of $\hat{\boldsymbol{a}}$, but translating the signal into an online inference algorithm remains challenging.

Our Approach. LRMs contain meaningful internal signals indicating when $\hat{\boldsymbol{a}}$ first appears, but these signals are difficult to exploit with hand-designed early-exit rules. We therefore frame early exiting as a prediction problem: given the LRM's final-layer hidden states, predict whether the final answer $\hat{\boldsymbol{a}}$ has already been generated. Our method trains a probe classifier to aggregate these internal signals. While prior work has used hidden states to study whether LRMs recognize when intermediate CoT answers are correct (Zhang et al., 2025a), our goal is different: we probe for the *final answer* $\hat{\boldsymbol{a}}$ itself, a self-contained signal available during the model's reasoning process and requiring no ground-truth labels at inference time. This enables a practical early-exit strategy.

We train the probe at the token level, formulating the data as a token classification task. This provides much finer-grained supervision than prior methods, which typically segment each CoT $\boldsymbol{r}$ using heuristics such as "thinking tokens" or paragraph delimiters like `\n\n`, and then exit when the predicted probability exceeds a data-calibrated threshold (Liu & Wang, 2025; Wu et al., 2025a; Zhang et al., 2025a). Our token-level formulation offers two inference-time benefits: the probe can exit immediately after $\hat{\boldsymbol{a}}$ is generated, and a data-calibrated threshold is optional rather than required. Avoiding mandatory data-calibrated thresholding is important because such thresholds require additional samples from the evaluation distribution and may not transfer across distributions.

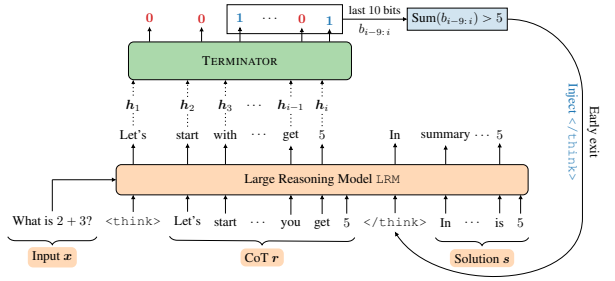

Figure 3. **Early stopping via TERMINATOR.** TERMINATOR is a binary probe classifier that predicts whether to exit or not at every CoT token. Once the majority of prediction bits within a window (10 here) are 1, `</think>` is injected into the LRM's token stream to stop thinking.

However, obtaining the $\hat{\boldsymbol{a}}$ positions to create our HORL-dataset in a scalable way is challenging and highly non-trivial, which we precisely address in the next section.

4. TERMINATOR: Methodology

Given a full CoT and the corresponding final solution from an LRM, the earliest logical arrival to the LRM's final answer can be detected in the CoT. However, reliably detecting tens of thousands of CoTs is a unique challenge, which we address through our training dataset curation pipeline in App. A.1. Our approach for training TERMINATOR, which is a probe classifier, in **??**.

Binary Probing Classifier. Our approach entails training a small classification model $\boldsymbol{\theta}$ on the LRM's final-layer hidden states $\boldsymbol{h}_i$ and making a binary prediction b_i at each CoT position $i \in [M]$ (Fig. 3). More specifically, our model reuses the same transformer block from the LRM and adds a prediction head. The weights of the transformer block are copied from the final block of the LRM, which we found performs slightly better than random initialization, and the prediction head is randomly initialized. During training, the task is to predict whether the first occurrence of the final answer has been generated (label 1) or not (label 0). Given the causality of the transformer block, every prediction depends on the history of the CoT up to that point, but the predictions themselves are made independently of each other. Due to the inherent class imbalance of this early-exiting prediction task, our model is trained with class-weighted binary cross-entropy loss, which for a single sample $(\boldsymbol{x}, \boldsymbol{r}, \boldsymbol{s}, i^\star)$ is computed as:

$$L(\boldsymbol{\theta}) = -\frac{1}{M}\sum_{i=1}^{M}\Big[w_1 \cdot y_i \cdot \log p_i + w_0 \cdot (1-y_i) \cdot \log(1-p_i)\Big], \tag{3}$$

where $y_i = \mathbb{1}(i < i^\star) \in \{0, 1\}$ denotes the ground-truth label corresponding to answer arrival and $p_i = \mathbb{P}_{\boldsymbol{\theta}}\left(b_i = 1 \mid \boldsymbol{x}, \boldsymbol{r}_{\leqslant i}\right)$ is the predicted probability for each $i \in [M]$, with M being the CoT length,

and w_0 and w_1 the class weights. These weights are automatically computed from the training dataset using inverse frequency weighting as $w_0 = (n_0 + n_1)/2n_0$ and $w_1 = (n_0 + n_1)/(2n_1)$, where n_0 and n_1 are the total number of 0 and 1 labels in the training dataset, respectively.

TERMINATOR is inspired by the findings of optimal-length reasoning literature; we seek to train a model on hindsight-optimal CoTs to encourage TERMINATOR to early-exit as soon as the final answer is generated. Unlike other methods, TERMINATOR is free of data-calibrated thresholding and is trained on several data sources (math, coding, and STEM problems) simultaneously.

5. Experiments

We provide and discuss our main results Sec. 5.1 and additional ablation studies Sec. 5.2. Implementation details are provided in App. A.2.

5.1. TERMINATOR: Main Results

Fig. 15 shows the performance of TERMINATOR and relevant baselines with respect to the Compression Rate (lower is better) and Accuracy (higher is better). To ensure a fair comparison, all methods are evaluated using the same CoTs that are used in the vanilla baseline, except the NoThinking baseline, for which no CoT is generated. While Fig. 15 shows that TERMINATOR achieves a better overall compression-performance trade-off over prior methods, the same results—presented in table format—in Table 7 show that TERMINATOR also achieves best or second best performance on 28 out of 32 metrics. Notably, while methods such as *Dynasor* achieve aggressive token reduction, they do so at the cost of significant accuracy degradation. TERMINATOR consistently occupies a favorable position on the accuracy-efficiency Pareto frontier across all four evaluated LRMs, demonstrating that its advantages are robust to model architecture and scale.

Table 1. **Latency Analysis.** Latency and throughput benchmarks on MATH-500 problems (batch size 1) for Qwen3-based vanilla and TERMINATOR models. TERMINATOR reduces latency costs by a factor of over 2×, but does incur a slight throughput overhead. Values are reported as the mean ± 95% CI.

Method	Latency (s)	Throughput (tok/s)
Qwen3-8B		
Vanilla	32.68 ± 9.59	151.5 ± 4.4
Terminator	14.10 ± 6.27	135.2 ± 2.0
Qwen3-14B		
Vanilla	43.38 ± 13.98	98.0 ± 2.0
Terminator	18.76 ± 6.52	90.6 ± 0.8

5.2. Ablation Studies

We further run ablation studies on TERMINATOR with respect to its (1) latency and throughput over the vanilla LRM, (2) performance against the truncated early-exit baseline, (3) out-of-distribution (OOD) performance, and (4) TERMINATOR's recovery of our observed answer signal phenomena (Figs. 2 and 11). Our results in this section will be reported mostly on Qwen3-8B alone, with results on Qwen3-14B, Ministral-3-8B, and Ministral-3-14B reported in Fig. 15 and Table 7.

Latency Analysis. Table 1 shows the results of our latency analysis; TERMINATOR halves the average latency over the vanilla LRM, despite incurring a small overhead of 10.8% for Qwen3-8B and 7.5% for Qwen3-14B, respectively. Note that as the base LRM size increases, TERMINATOR incurs a proportionally smaller overhead since its architecture (a single transformer layer and an FFN) remains fixed. For our analysis, we develop a vLLM-compatible implementation of TERMINATOR and compare with running the vanilla LRM in vLLM. Both methods are evaluated on the same subset of MATH-500 questions with a batch size of 1, disabled prefix caching, and on a single GH200.

Hindsight-Optimal CoTs. Since TERMINATOR is trained on hindsight-optimal reasoning length (HORL) CoTs, it is natural to ask where TERMINATOR lies on the accuracy-compression frontier relative to the ground-truth HORL, which is shown in Fig. 4. Each dot on the curves represents the accuracy when each CoT was truncated early, and the LRM was forced to give a final solution and final answer. The diamond-shaped markers represent the position of the first occurrence of $\hat{\boldsymbol{a}}$, and therefore represent the points corresponding to hindsight-optimal reasoning. As expected, the accuracy remains constant after this point, showing that additional reasoning beyond $\hat{\boldsymbol{a}}$ does not yield better performance. We plot TERMINATOR alongside these curves to show how close it is to the hindsight-optimal CoT length and performance. Even though the HORL baseline is not achievable by any method in principle, TERMINATOR is notably close to it for all datasets.

OOD Evaluation. We train separate models on each of the four tasks in our training dataset and evaluate them on the test datasets. Whereas our main results in Fig. 15 use TERMINATOR trained on all four tasks, this experiment trains on one task at a time to assess OOD generalization. Fig. 13 reports compression rate and accuracy, with rows denoting the training dataset, columns denoting the test dataset, and cell values denoting test performance. The results show that compression is best *in-distribution*, i.e., along the diagonal, but accuracy does not always follow the same pattern. For example, training on OpenScience yields the lowest GPQA accuracy despite GPQA being in-distribution, whereas training on the OOD

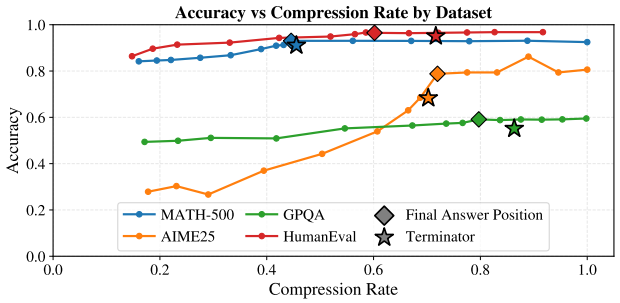

Figure 4. **Effects of Early CoT Termination.** Test set CoTs are evaluated after truncating them at various points via `</think>` and asking the LRM for a final solution and answer. Diamond-shaped points show the hindsight-optimal reasoning length, and TERMINATOR falls close to optimality for three out of the four datasets.

OpenCoder-SFT dataset improves accuracy but worsens compression to 96%. Thus, OOD evaluation can slightly improve accuracy but often delays exiting, thereby reducing token savings on data not seen during training.

TERMINATOR Recovers Early-Exit Signals. We further highlight that using TERMINATOR's predicted exit positions, we recover the *same* event-locked averaging (Fig. 2) and "thinking token" frequency (Fig. 11) phenomena. Fig. 9 mirrors Fig. 2, but uses all test samples rather than 3,200 randomly selected training samples. Its left and center panels show event-locked average Token-Confidence using ground-truth and TERMINATOR-predicted answer positions, respectively, while the right panel shows that prediction errors are concentrated near zero, with a median difference of 7. This alignment helps explain why TERMINATOR recovers most of the same signal. Similarly, Fig. 10 parallels Fig. 11 by overlaying scatter plots computed from ground-truth and predicted answer positions. The inset axes show nearly identical above-diagonal percentages, indicating that TERMINATOR preserves the same before/after "thinking token" usage biases. Together, Figs. 9 and 10 show that training TERMINATOR on the LRM's hidden states is sufficient to independently recover the early-exit signals identified above, justifying our approach.

6. Related Work

Prompt Compression. This line of work is concerned with compressing the input prompt (or context) before passing it to an LLM. Some methods use *soft-prompts* (Mu et al., 2023; Chevalier et al., 2023; Ge et al., 2024; Qin et al., 2024) to compress tokenized inputs into a sequence of embeddings. These embeddings serve as the LLM's input, allowing richer expressivity, but they are not amenable to black-box LLMs and are difficult to analyze theoretically. Other methods use *hard-prompts* (Jung & Kim, 2024; Jiang et al., 2024; Pan et al., 2024; Nagle et al., 2024), keeping the final compressed input prompt fixed to the same token vocabulary as the LLM.

Efficient Reasoning. Analogously to soft-prompt compression, *latent* or *continuous* reasoning is a technique where reasoning unfolds across latent output embeddings (or hidden states) rather than discrete tokens. Methods like Coconut (Hao et al., 2025), CCoT (Cheng & Durme, 2024), and Soft Thinking (Zhang et al., 2025d) feed the LLM's output embeddings back into the input of the LLM during the reasoning stage, which significantly decreases the number of passes through the LLM before arriving at the final answer. LightThinker (Zhang et al., 2025c) uses an idea similar to AutoCompressor (Chevalier et al., 2023), where each reasoning step is generated as discrete tokens first, compressed, and then the compressed summary of the reasoning thus far is fed back into the LLM to generate the next step. Other methods, like TokenSkip and C3oT (Xia et al., 2025; Kang et al., 2025a), are closer to hard-prompt compression, where a prompt compressor (or a summarization) model first compresses the CoTs into much shorter versions, and the LLM is retrained on these shorter CoTs. CoT-Valve (Ma et al., 2025b) extends this idea by introducing a parameter that adds explicit control over the reasoning length after fine-tuning. CALM (Schuster et al., 2022) also keeps its outputs in the token space, but saves compute by making layer-wise token-level early-exit decisions.

Early-Exit Reasoning. These methods seek to make reasoning more efficient by terminating the CoT early. All existing methods use a consistency-based approach, injecting the `</think>` token at various points to force the model to generate an answer or a useful signal. Some methods, like EAT (Wang et al., 2025b), DEER (Yang et al., 2025b), ES-CoT (Mao et al., 2025), and Dynasor (Fu et al., 2025a) are training-free; they track signals throughout the reasoning process and exit when a threshold is crossed. Other methods, like SpecExit (Yang et al., 2025c), Learn To Stop (Liu & Wang, 2025), Thought Calibration (Wu et al., 2025a), and FlashThink (Jiang et al., 2025) rely on training a separate probe classifier by using consistency as the main approach for gathering their training signals. By contrast, our work constructs a training signal to predict the immediate arrival of $\hat{a}$, thereby training on hindsight-optimal length CoTs. In addition, our work does not require threshold tuning on validation data, which is needed for Learn To Stop and Thought Calibration.

7. Conclusion

We present TERMINATOR, an early-exit method for LRM reasoning. Training TERMINATOR requires an optimal-length dataset of CoTs, which can be obtained through our robust answer extraction, identification, and verification pipeline. Furthermore, we provide novel analysis and insights into the behaviors of an LRM's (1) Token-Confidence during reasoning (Fig. 2), and (2) shift in "thinking token"

usage. While our training data curation pipeline works well, future work can explore making training more efficient as tens of thousands of CoTs are used to train TERMINATOR.

Acknowledgements

This work was partially supported by NSF CAREER Award 2443857, ARO Award W911NF2310062, ONR Award N000142412542, and the 6G@UT center within the Wireless Networking and Communications Group (WNCG) at the University of Texas at Austin.

References

Art of Problem Solving. Aime problems and solutions. https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions. Accessed: 2026-01-07.

Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. Evaluating large language models trained on code. *CoRR*, abs/2107.03374, 2021. URL https://arxiv.org/abs/2107.03374.

Chen, X., Xu, J., Liang, T., He, Z., Pang, J., Yu, D., Song, L., Liu, Q., Zhou, M., Zhang, Z., Wang, R., Tu, Z., Mi, H., and Yu, D. 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.

Cheng, J. and Durme, B. V. Compressed chain of thought: Efficient reasoning through dense representations, 2024. URL https://arxiv.org/abs/2412.13171.

Chevalier, A., Wettig, A., Ajith, A., and Chen, D. Adapting language models to compress contexts. In Bouamor, H., Pino, J., and Bali, K. (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3829–3846, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.232. URL https://aclanthology.org/2023.emnlp-main.232/.

Ding, B., Chen, Y., Wang, F., Ming, L., and Lin, T. Do thinking tokens help or trap? towards more efficient large reasoning model, 2025. URL https://arxiv.org/abs/2506.23840.

Fu, Y., Chen, J., Zhu, S., Fu, Z., Dai, Z., Zhuang, Y., Ma, Y., Qiao, A., Rosing, T., Stoica, I., and Zhang, H. Efficiently scaling llm reasoning with certaindex, 2025a. URL https://arxiv.org/abs/2412.20993.

Fu, Y., Wang, X., Tian, Y., and Zhao, J. Deep think with confidence, 2025b. URL https://arxiv.org/abs/2508.15260.

Gao, J., Yan, S., Tan, Q., Yang, L., Xu, S., Fu, W., Mei, Z., Lyu, K., and Wu, Y. How far are we from optimal reasoning efficiency?, 2025. URL https://arxiv.org/abs/2506.07104.

Ge, T., Jing, H., Wang, L., Wang, X., Chen, S.-Q., and Wei, F. In-context autoencoder for context compression in a large language model. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=uREj4ZuGJE.

Guo, D., Yang, D., Zhang, H., Song, J., Wang, P., Zhu, Q., Xu, R., Zhang, R., Ma, S., Bi, X., Zhang, X., Yu, X., Wu, Y., Wu, Z. F., Gou, Z., Shao, Z., Li, Z., Gao, Z., Liu, A., Xue, B., Wang, B., Wu, B., Feng, B., Lu, C., Zhao, C., Deng, C., Ruan, C., Dai, D., Chen, D., Ji, D., Li, E., Lin, F., Dai, F., Luo, F., Hao, G., Chen, G., Li, G., Zhang, H., Xu, H., Ding, H., Gao, H., Qu, H., Li, H., Guo, J., Li, J., Chen, J., Yuan, J., Tu, J., Qiu, J., Li, J., Cai, J. L., Ni, J., Liang, J., Chen, J., Dong, K., Hu, K., You, K., Gao, K., Guan, K., Huang, K., Yu, K., Wang, L., Zhang, L., Zhao, L., Wang, L., Zhang, L., Xu, L., Xia, L., Zhang, M., Zhang, M., Tang, M., Zhou, M., Li, M., Wang, M., Li, M., Tian, N., Huang, P., Zhang, P., Wang, Q., Chen, Q., Du, Q., Ge, R., Zhang, R., Pan, R., Wang, R., Chen, R. J., Jin, R. L., Chen, R., Lu, S., Zhou, S., Chen, S., Ye, S., Wang, S., Yu, S., Zhou, S., Pan, S., Li, S. S., Zhou, S., Wu, S., Yun, T., Pei, T., Sun, T., Wang, T., Zeng, W., Liu, W., Liang, W., Gao, W., Yu, W., Zhang, W., Xiao, W. L., An, W., Liu, X., Wang, X., Chen, X., Nie, X., Cheng, X., Liu, X., Xie, X., Liu, X., Yang, X., Li, X., Su, X., Lin, X., Li, X. Q., Jin, X., Shen, X., Chen, X., Sun, X., Wang, X., Song, X., Zhou, X., Wang, X., Shan, X., Li, Y. K., Wang, Y. Q., Wei, Y. X., Zhang, Y., Xu, Y., Li, Y., Zhao, Y., Sun, Y., Wang, Y., Yu, Y., Zhang, Y., Shi, Y., Xiong, Y., He, Y., Piao, Y., Wang, Y., Tan, Y., Ma, Y., Liu, Y., Guo, Y., Ou, Y., Wang, Y., Gong, Y., Zou, Y., He, Y., Xiong, Y., Luo, Y., You, Y., Liu, Y., Zhou, Y., Zhu, Y. X., Huang, Y., Li,

Y., Zheng, Y., Zhu, Y., Ma, Y., Tang, Y., Zha, Y., Yan, Y., Ren, Z. Z., Ren, Z., Sha, Z., Fu, Z., Xu, Z., Xie, Z., Zhang, Z., Hao, Z., Ma, Z., Yan, Z., Wu, Z., Gu, Z., Zhu, Z., Liu, Z., Li, Z., Xie, Z., Song, Z., Pan, Z., Huang, Z., Xu, Z., Zhang, Z., and Zhang, Z. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, September 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z.

Hao, S., Sukhbaatar, S., Su, D., Li, X., Hu, Z., Weston, J., and Tian, Y. Training large language models to reason in a continuous latent space, 2025. URL https://arxiv.org/abs/2412.06769.

Huang, S., Cheng, T., Liu, J. K., Hao, J., Song, L., Xu, Y., Yang, J., Liu, J. H., Zhang, C., Chai, L., Yuan, R., Zhang, Z., Fu, J., Liu, Q., Zhang, G., Wang, Z., Qi, Y., Xu, Y., and Chu, W. Opencoder: The open cookbook for top-tier code large language models. 2024. URL https://arxiv.org/pdf/2411.04905.

Hugging Face. Qwen3-8B model card. https://web.archive.org/web/20260502201803/https://huggingface.co/Qwen/Qwen3-8B, 2026. Archived May 2, 2026.

Jiang, G., Quan, G., Ding, Z., Luo, Z., Wang, D., and Hu, Z. Flashthink: An early exit method for efficient reasoning, 2025. URL https://arxiv.org/abs/2505.13949.

Jiang, H., Wu, Q., Luo, X., Li, D., Lin, C.-Y., Yang, Y., and Qiu, L. LongLLMLingua: Accelerating and enhancing LLMs in long context scenarios via prompt compression. In *ICLR 2024 Workshop on Mathematical and Empirical Understanding of Foundation Models*, 2024. URL https://openreview.net/forum?id=9YvfRrpmyw.

Jung, H. and Kim, K.-J. Discrete prompt compression with reinforcement learning. *IEEE Access*, 12:72578–72587, 2024. ISSN 2169-3536. doi: 10.1109/access.2024.3403426. URL http://dx.doi.org/10.1109/ACCESS.2024.3403426.

Kang, Y., Sun, X., Chen, L., and Zou, W. C3ot: Generating shorter chain-of-thought without compromising effectiveness. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(23):24312–24320, Apr. 2025a. doi: 10.1609/aaai.v39i23.34608. URL https://ojs.aaai.org/index.php/AAAI/article/view/34608.

Kang, Z., Zhao, X., and Song, D. Scalable best-of-n selection for large language models via self-certainty, 2025b. URL https://arxiv.org/abs/2502.18581.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Lee, C., Rush, A. M., and Vafa, K. Critical thinking: Which kinds of complexity govern optimal reasoning length?, 2025. URL https://arxiv.org/abs/2504.01935.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step. In *ICLR*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

Liu, A. H., Khandelwal, K., Subramanian, S., Jouault, V., Rastogi, A., Sadé, A., Jeffares, A., Jiang, A., Cahill, A., Gavaudan, A., Sablayrolles, A., Héliou, A., You, A., Ehrenberg, A., Lo, A., Eliseev, A., Calvi, A., Soori-yarachchi, A., Bout, B., Rozière, B., Monicault, B. D., Lanfranchi, C., Barreau, C., Courtot, C., Grattarola, D., Dabert, D., de las Casas, D., Chane-Sane, E., Ahmed, F., Berrada, G., Ecrepont, G., Guinet, G., Novikov, G., Kunsch, G., Lample, G., Martin, G., Gupta, G., Ludziejewski, J., Rute, J., Studnia, J., Amar, J., Delas, J., Roberts, J. S., Yadav, K., Chandu, K., Jain, K., Aitchison, L., Fainsin, L., Blier, L., Zhao, L., Martin, L., Saulnier, L., Gao, L., Buyl, M., Jennings, M., Pellat, M., Prins, M., Poirée, M., Guillaumin, M., Dinot, M., Futeral, M., Darrin, M., Augustin, M., Chiquier, M., Schimpf, M., Grinsztajn, N., Gupta, N., Raghuraman, N., Bousquet, O., Duchenne, O., Wang, P., von Platen, P., Jacob, P., Wambergue, P., Kurylowicz, P., Muddireddy, P. R., Chagniot, P., Stock, P., Agrawal, P., Torroba, Q., Sauvestre, R., Soletskyi, R., Menneer, R., Vaze, S., Barry, S., Gandhi, S., Waghjale, S., Gandhi, S., Ghosh, S., Mishra, S., Aithal, S., Antoniak, S., Scao, T. L., Cachet, T., Sorg, T. S., Lavril, T., Saada, T. N., Chabal, T., Foubert, T., Robert, T., Wang, T., Lawson, T., Bewley, T., Bewley, T., Edwards, T., Jamil, U., Tomasini, U., Nemychnikova, V., Phung, V., Maladière, V., Richard, V., Bouaziz, W., Li, W.-D., Marshall, W., Li, X., Yang, X., Ouahidi, Y. E., Wang, Y., Tang, Y., and Ramzi, Z. Ministral 3, 2026. URL https://arxiv.org/abs/2601.08584.

Liu, X. and Wang, L. Answer convergence as a signal for early stopping in reasoning, 2025. URL https://arxiv.org/abs/2506.02536.

Lou, C., Sun, Z., Liang, X., Qu, M., Shen, W., Wang, W., Li, Y., Yang, Q., and Wu, S. Adacot: Pareto-optimal adaptive chain-of-thought triggering via reinforcement learning, 2025. URL https://arxiv.org/abs/2505.11896.

Luck, S. J. *An introduction to the event-related potential technique / Steven J. Luck.* The MIT Press, Cambridge, Massachusetts, second edition. edition, 2014. ISBN 0-262-32406-7.

Luo, H., Shen, L., He, H., Wang, Y., Liu, S., Li, W., Tan, N., Cao, X., and Tao, D. O1-pruner: Length-harmonizing fine-tuning for o1-like reasoning pruning, 2025. URL https://arxiv.org/abs/2501.12570.

Ma, W., He, J., Snell, C., Griggs, T., Min, S., and Zaharia, M. Reasoning models can be effective without thinking, 2025a. URL https://arxiv.org/abs/2504.09858.

Ma, X., Wan, G., Yu, R., Fang, G., and Wang, X. CoT-valve: Length-compressible chain-of-thought tuning. In Che, W., Nabende, J., Shutova, E., and Pilehvar, M. T. (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/.

Mao, M., Yin, B., Zhu, Y., and Fang, X. Early stopping chain-of-thoughts in large language models, 2025. URL https://arxiv.org/abs/2509.14004.

Mu, J., Li, X. L., and Goodman, N. Learning to compress prompts with gist tokens. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=2DtxPCL3T5.

Muennighoff, N., Yang, Z., Shi, W., Li, X. L., Fei-Fei, L., Hajishirzi, H., Zettlemoyer, L., Liang, P., Candès, E., and Hashimoto, T. s1: Simple test-time scaling, 2025. URL https://arxiv.org/abs/2501.19393.

Nagle, A., Girish, A., Bondaschi, M., Gastpar, M., Makkuva, A. V., and Kim, H. Fundamental limits of prompt compression: A rate-distortion framework for black-box language models. In Globerson, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J., and Zhang, C. (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 94934–94970. Curran Associates, Inc., 2024. doi: 10.52202/079017-3009. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/ac8fbba029dadca99d6b8c3f913d3ed6-Paper-Conference.pdf.

NVIDIA. OpenScience dataset (v1). https://huggingface.co/datasets/nvidia/OpenScience, 2025. Last updated: June 18 (per repository history). Accessed: 2026-01-07.

OpenAI. Learning to reason with llms. https://openai.com/index/learning-to-reason-with-llms/, September 12 2024. Accessed: 2025-01-19.

Pan, Z., Wu, Q., Jiang, H., Xia, M., Luo, X., Zhang, J., Lin, Q., Ruhle, V., Yang, Y., Lin, C.-Y., Zhao, H. V., Qiu, L., and Zhang, D. LLMLingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. In Ku, L.-W., Martins, A., and Srikumar, V. (eds.), *Findings of the Association for Computational Linguistics ACL 2024*, pp. 963–981, Bangkok, Thailand and virtual meeting, August 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.findings-acl.57.

Qian, C., Liu, D., Wen, H., Bai, Z., Liu, Y., and Shao, J. Demystifying reasoning dynamics with mutual information: Thinking tokens are information peaks in llm reasoning. *arXiv preprint arXiv:2506.02867*, 2025.

Qin, G., Rosset, C., Chau, E., Rao, N., and Van Durme, B. Dodo: Dynamic contextual compression for decoder-only LMs. In Ku, L.-W., Martins, A., and Srikumar, V. (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9961–9975, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.536. URL https://aclanthology.org/2024.acl-long.536/.

Rein, D., Hou, B. L., Stickland, A. C., Petty, J., Pang, R. Y., Dirani, J., Michael, J., and Bowman, S. R. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

Schuster, T., Fisch, A., Gupta, J., Dehghani, M., Bahri, D., Tran, V., Tay, Y., and Metzler, D. Confident adaptive language modeling. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 17456–17472. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/6fac9e316a4ae75ea244ddcef1982c71-Paper-Conference.pdf.

Shrivastava, V., Awadallah, A., Balachandran, V., Garg, S., Behl, H., and Papailiopoulos, D. Sample more to think less: Group filtered policy optimization for concise reasoning, 2025. URL https://arxiv.org/abs/2508.09726.

Wang, C., Feng, Y., Chen, D., Chu, Z., Krishna, R., and Zhou, T. Wait, we don't need to "wait"! removing thinking tokens improves reasoning efficiency.

In Christodoulopoulos, C., Chakraborty, T., Rose, C., and Peng, V. (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/.

Wang, X., McInerney, J., Wang, L., and Kallus, N. Entropy after ⟨/Think⟩ for reasoning model early exiting, 2025b. URL https://arxiv.org/abs/2509.26522.

Wu, C., Li, B., Gao, M., Tian, Y., and Wang, Z. From efficiency to adaptivity: A deeper look at adaptive reasoning in large language models, 2026. URL https://arxiv.org/abs/2511.10788.

Wu, M., Zhou, C., Bates, S., and Jaakkola, T. Thought calibration: Efficient and confident test-time scaling. In Christodoulopoulos, C., Chakraborty, T., Rose, C., and Peng, V. (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 14302–14316, Suzhou, China, November 2025a. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.722. URL https://aclanthology.org/2025.emnlp-main.722/.

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

Xia, H., Leong, C. T., Wang, W., Li, Y., and Li, W. TokenSkip: Controllable chain-of-thought compression in LLMs. In Christodoulopoulos, C., Chakraborty, T., Rose, C., and Peng, V. (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 3351–3363, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.165. URL https://aclanthology.org/2025.emnlp-main.165/.

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., Zheng, C., Liu, D., Zhou, F., Huang, F., Hu, F., Ge, H., Wei, H., Lin, H., Tang, J., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Zhou, J., Lin, J., Dang, K., Bao, K., Yang, K., Yu, L., Deng, L., Li, M., Xue, M., Li, M., Zhang, P., Wang, P., Zhu, Q., Men, R., Gao, R., Liu, S., Luo, S., Li, T., Tang, T., Yin, W., Ren, X., Wang, X., Zhang, X., Ren, X., Fan, Y., Su, Y., Zhang, Y., Zhang, Y., Wan, Y., Liu, Y., Wang, Z., Cui, Z., Zhang, Z., Zhou, Z., and Qiu, Z. Qwen3 technical report, 2025a. URL https://arxiv.org/abs/2505.09388.

Yang, C., Si, Q., Duan, Y., Zhu, Z., Zhu, C., Li, Q., Chen, M., Lin, Z., and Wang, W. Dynamic early exit in reasoning models, 2025b. URL https://arxiv.org/abs/2504.15895.

Yang, R., Bai, H., Liu, S., Yu, G., Fan, R., Dang, Y., Zhang, J., Liu, K., Zhu, J., and Chen, P. Specexit: Accelerating large reasoning model via speculative exit, 2025c. URL https://arxiv.org/abs/2509.24248.

Yi, J., Wang, J., and Li, S. Shorterbetter: Guiding reasoning models to find optimal inference length for efficient reasoning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=MJvwM5dBZM.

Zhang, A., Chen, Y., Pan, J., Zhao, C., Panda, A., Li, J., and He, H. 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.

Zhang, J., Lin, N., Hou, L., Feng, L., and Li, J. AdaptThink: Reasoning models can learn when to think. In Christodoulopoulos, C., Chakraborty, T., Rose, C., and Peng, V. (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 3716–3730, Suzhou, China, November 2025b. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.184. URL https://aclanthology.org/2025.emnlp-main.184/.

Zhang, J., Zhu, Y., Sun, M., Luo, Y., Qiao, S., Du, L., Zheng, D., Chen, H., and Zhang, N. LightThinker: Thinking step-by-step compression. In Christodoulopoulos, C., Chakraborty, T., Rose, C., and Peng, V. (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 13307–13328, Suzhou, China, November 2025c. 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/.

Zhang, Z., He, X., Yan, W., Shen, A., Zhao, C., Wang, S., Shen, Y., and Wang, X. E. Soft thinking: Unlocking the reasoning potential of llms in continuous concept space, 2025d. URL https://arxiv.org/abs/2505.15778.

Algorithm 1 Answer Span Extraction, Identification, and Verification With Feedback

Input: CoT $\boldsymbol{r}$, final solution $\boldsymbol{s}$, `LRM`, `tokenizer`, max retries K
Output: answer position $i^\star$ (token index where answer is reached)

// Extract final answer value from model output
$\hat{\boldsymbol{a}} \leftarrow$ `LRM`("Extract the final answer from: " + $\boldsymbol{s}$)

// Iteratively extract and verify span with feedback
$\boldsymbol{z} \leftarrow \varnothing$ *// feedback provided to the LRM*
for $k = 1, \ldots, K$ **do**
  // Ask LRM to identify a string span containing the first occurrence of $\hat{\boldsymbol{a}}$ in $\boldsymbol{r}$
  $\boldsymbol{d} \leftarrow$ `LRM`("Find first occurrence of " + $\hat{\boldsymbol{a}}$ + " in: " + $\boldsymbol{r}$ + $\boldsymbol{z}$)

  // Verify the identified span contains the answer
  $v \leftarrow$ `LRM`("Does " + $\boldsymbol{d}$ + " contain " + $\hat{\boldsymbol{a}}$ + "?")

  if $v ==$ true **then**
    break *// span verified, proceed*
  end if

  $\boldsymbol{z} \leftarrow \boldsymbol{z}$ + "\n Previous span " **+** $\boldsymbol{d}$ **+** " was incorrect, try again"
end for

// Pattern match span text to get character-wise positioning of the span
$c \leftarrow$ `FuzzyMatch`($\boldsymbol{d}, \boldsymbol{r}$) *// c is an integer-based character index of $\boldsymbol{d}$ in $\boldsymbol{r}$*

// Convert to token position where answer is reached
$i^\star \leftarrow$ `CharToTokenPos`($c +$ `len`($\boldsymbol{d}$), $\boldsymbol{r}$, `tokenizer`)

return $i^\star$

A. Additional Details on Our Methods

A.1. Early Answer Extraction, Identification, and Verification

Our early answer extraction, identification, and verification pipeline (Fig. 5) is a critical component of our data curation process; at its core is an LRM that (1) extracts the final answer $\hat{\boldsymbol{a}}$ from final solution $\boldsymbol{s}$ (answer extraction), (2) identifies the earliest logical arrival to $\hat{\boldsymbol{a}}$ in $\boldsymbol{r}$ (answer identification), (3) verifies that the extraction step was successful (answer verification). And finally (4) we extract the exact position of $\hat{\boldsymbol{a}}$ from the CoT (token-index extraction).

Rationale. Extracting the position of $\hat{\boldsymbol{a}}$ is not trivial. Human inspection and annotation of CoTs is one route, but it is expensive and not scalable. Our early attempts at answer extraction, identification, and verification relied solely on fuzzy pattern matching, resulting in many false positives despite our best efforts to accommodate as many edge cases as possible. The primary challenge is that identifying the answer position within a CoT is a semantic search problem that cannot be reliably solved with fuzzy or regex pattern matching for numerical answers, mathematical expressions, and code. Examples where pattern matching fails for these three is given in, which we illustrate by three failure modes:

1. **Numerical answers.** A numerical value may appear frequently throughout the CoT in intermediate calculations, problem restatements, or discarded solution attempts, making it impossible to distinguish these occurrences from the true final answer by pattern alone. For example, if the final answer is `x = 42`, the value `42` may appear dozens of times in prior reasoning steps without having any connection to a logical arrival to that answer. The CoT could simply consider `42` without steps for arriving to it as an answer, or `42` could appear out of pure coincidence as an intermediate value during calculation.

2. **Mathematical expressions.** The same mathematical object can be represented in many syntactically distinct forms.

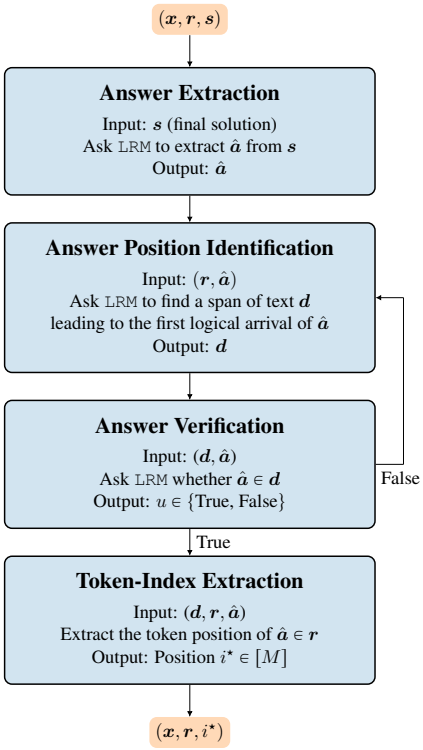

Figure 5. **Training-Dataset Curation Process.** We use an LRM to (1) extract final answer $\hat{\boldsymbol{a}}$ from final solution $\boldsymbol{s}$, (2) identify the earliest position of $\hat{\boldsymbol{a}}$ in the CoT $\boldsymbol{r}$, and (3) verify that the position was correct. If it was, then we can extract the exact position of $\hat{\boldsymbol{a}}$ from the CoT at the final token-index extraction step; otherwise, we retry the identification step with feedback.

For instance, `x`^{2}, `x**2`, `pow(x,2)`, and `x` $\cdot$ `x` are semantically equivalent but would not be matched by any single pattern. Differences in LaTeX formatting, Unicode symbols, and whitespace further compound this.

3. **Python functions.** A Python function may not appear as a contiguous block anywhere in the CoT; instead, it may be generated line by line, interspersed with commentary. The final reconstructed answer, therefore, does not exist verbatim in the text, making positional matching fundamentally ill-posed.

Using an LRM for all three cases confirmed that the earliest answer positions can be reliably extracted.

Our Extract-Identify-Verify Pipeline. We first extract the final answer $\hat{\boldsymbol{a}}$ from $\boldsymbol{s}$, where it is explicitly marked, e.g. with `\boxed{}`, making extraction straightforward for the LRM. Next, the LRM identifies a span $\boldsymbol{d}$ that both precedes and includes $\hat{\boldsymbol{a}}$. This ensures that $\boldsymbol{d}$ is a unique substring of $\boldsymbol{r}$, allowing us to recover the exact token position of the earliest occurrence of $\hat{\boldsymbol{a}}$. The LRM then verifies that $\boldsymbol{d}$ contains $\hat{\boldsymbol{a}}$. If verification fails, the LRM repeats the identification step, providing textual feedback listing all previously selected spans that did not contain $\hat{\boldsymbol{a}}$, reducing the chance of selecting the same span again. If no valid span is found within the retry limit, the corresponding CoT is excluded from the training set. Otherwise, we locate $\boldsymbol{d}$ in $\boldsymbol{r}$ and retrieve the earliest answer token position $i^\star$. Algorithm 1 in App. A.1 gives pseudocode for this procedure. These steps scale the construction of TERMINATOR, which we use to train our probe classifier. Our cost-benefit analysis in App. B.4 shows that the inference-time benefits of TERMINATOR substantially outweigh the cost of running this pipeline.

Algorithm 1 contains pseudocode for our pipeline. $i^\star$, the index of the earliest token position containing $\hat{\boldsymbol{a}}$, is used to construct the label set of our training data by setting all positions prior to $i^\star$ to 0 and setting all positions after $i^\star$ to 1. Each of the three steps (extraction, identification, and verification) requires separate calls to an LRM; please refer to our codebase for details on the exact system prompts that we used for each step.

A.2. Implementation Details

A.2.1. MODELS

We train and evaluate our method on LRMs from two different model families: Qwen3-8B and Qwen3-14B (Yang et al., 2025a), and Ministral-3-8B-Reasoning-2512 and Ministral-3-14B-Reasoning-2512 (Liu et al., 2026). We use

Qwen3-30B-A3B-Thinking-2507 for our answer extraction, identification, and verification pipeline. Our trained models consist of a single transformer layer initialized from the final layer of the LRM and a binary prediction head. We compare TERMINATOR against (1) prompt-based approaches, including Vanilla, NoThinking (Ma et al., 2025a), DEER (Yang et al., 2025b), and Dynasor (Fu et al., 2025a), and (2) a probe-based approach, Thought Calibration (Wu et al., 2025a). Vanilla is a direct evaluation of the LRM without any intervention. NoThinking prompts the model to skip the reasoning phase and generate the final solution $\boldsymbol{s}$ directly. DEER splits the reasoning into chunks, checks the average token probability after every chunk, and exits if it exceeds a threshold. Dynasor periodically prompts the model to produce intermediate answers at fixed token intervals and triggers early exit when 8 consecutive answers are consistent. Thought Calibration trains linear probes on the hidden representations of reasoning steps to automatically decide when to stop generation. We retrain these probes for our 4 models using their *Supervised* method. For further details of the baselines' implementations, we refer to App. A.3.

A.2.2. DATASETS

We form a training data mix with AIME (1983–2024) (Art of Problem Solving), MATH (Lightman et al., 2024), OpenCoder-SFT (Huang et al., 2024), and OpenScience (NVIDIA, 2025). We form our training datasets by sampling three CoTs from each dataset, identifying the answer positions (see App. A.1), and assigning the corresponding training labels. We evaluate our method and all baselines on AIME 2025 (Art of Problem Solving), MATH-500 (Lightman et al., 2024), HumanEval (Chen et al., 2021), and GPQA (Rein et al., 2024).

Our training dataset consists of CoTs from AIME (1983–2024) (Art of Problem Solving), MATH (Lightman et al., 2024), OpenCoder-SFT (Huang et al., 2024), and OpenScience (NVIDIA, 2025). All 933 samples and all 12,000 samples from the AIME (1983–2024) and MATH datasets, respectively, are used. We randomly select 12,000 samples from the `educational_instruct` subset of the OpenCoder-SFT-Stage2 dataset, which we refer to as "OpenCoder-SFT" in the main paper. This subset consists of generated and validated Python coding examples. Our sampling procedure for this dataset was not uniform, unlike the others. Instead, problems are grouped by their `entry_point` field, and sampling is split into rounds, with each round randomly sampling one problem from that group without replacement. Finally, we randomly sample an additional 12,000 samples from the `OS-Q3-235B-4` subset of the OpenScience dataset. This subset consists of multiple-choice STEM question-answer pairs that were synthetically generated from Qwen3-235B-A22B (Yang et al., 2025a).

Three CoTs are sampled per problem by the target LRM (we used Qwen and Mistral models), yielding a set of approximately 110,799 CoTs per LRM. The respective answer positions for each set of CoTs is obtained with our extraction method outlined in App. A.1. However, this procedure is not perfect, as even with our retry logic, the answer extractor, identifier, and verifier LRM (Qwen3-30B-A3B-Thinking-2507) cannot always identify the earliest final answer position. Thus, all three of these steps are successful for roughly 70%–80% of CoTs. Finally, a training-ready dataset for each LRM is formed by preparing label vectors (based on the answer positions), loss masks (based on the positions of `<think>` and `</think>`), and tokenizing the CoTs.

A.2.3. TRAINING DETAILS

During training, we optimize for high performance on a holdout validation set for our prediction task; we choose our model based solely on how well it performs on the binary predictive task, without peeking at the evaluation dataset performance. Our validation metric of choice is the Macro-F1 score.

All models are trained using a consistent set of hyperparameters: batch size of 2, gradient accumulation over 8 steps, no dropout, and a weight decay of 0.01. We initialize the learning rate at 2×10^{-4}. Training follows a cosine annealing schedule with 100 steps of linear warmup, decaying to a minimum learning rate of 1×10^{-6}.

Optimization is performed with the AdamW optimizer ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1 \times 10^{-8}$), and all runs use a fixed random seed of 1337. Each model consists of a single transformer layer and a single-layer binary classification head. Training proceeds for 2 epochs.

Fig. 6 shows the convergence curves of the training loss and the Macro-F1 scores on a small validation set during training. The validation set is a held-out set of examples from our curated dataset, but is not seen during training. Macro-F1 is the unweighted average of per-class F1 scores, treating all classes equally regardless of their frequency, making it a robust and fair metric for evaluating performance on class-imbalanced datasets. The following architectures are studied:

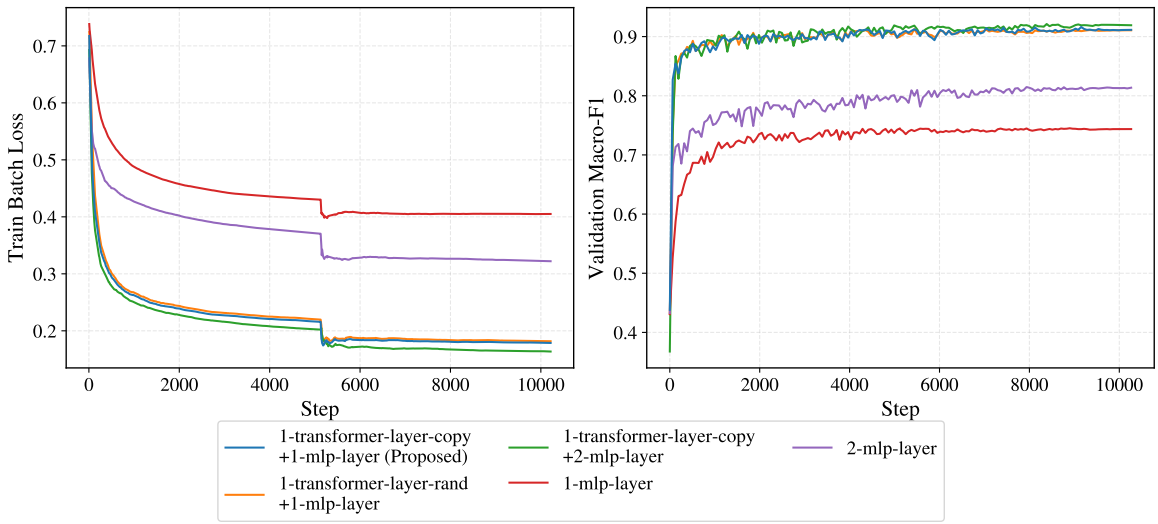

Figure 6. **Training Convergence.** During training, our proposed TERMINATOR architecture achieves the best balance between the highest Macro-F1 validation scores and the lowest compute overhead.

1. **1-transformer-layer-copy+1-mlp-layer:** The proposed design, consisting of a single transformer layer with copied initialization followed by a one-layer MLP
2. **1-transformer-layer-rand+1-mlp-layer:** A variant with random initialization of the transformer layer
3. **1-transformer-layer-copy+2-mlp-layer:** A variant with a two-layer MLP head
4. **1-mlp-layer, 2-mlp-layer:** Two MLP-only baselines with one and two layers, respectively.

The transformer block is critical: MLP-only architectures incur a substantial 10–17% drop in Macro-F1. When the transformer layer is included, increasing the MLP head from one to two layers yields only a marginal improvement of $\sim$0.9%, despite additional computational cost. The choice of initialization for the transformer layer (copied vs. random) yields a tiny improvement ($\sim$0.1%); we therefore adopt copied initialization by default, as it incurs no additional cost.

A.2.4. TERMINATOR INFERENCE DETAILS

We use vLLM (Kwon et al., 2023) with asynchronous requests to sample CoTs when curating our training datasets. During inference with our trained model, a sliding window of the 10 most recent predictions is used, and the `</think>` token is injected when more than 50% of the labels are 1 (majority voting). Our main results reported in this paper use a threshold of 0.7 to predict 1.

Window Size and Majority Voting. Our proposed strategy for early-exiting based on TERMINATOR's predictions uses majority voting over a window of the 10 most recent predictions. The window and majority vote serve as a consistency check, preventing a single spurious prediction from triggering an early exit. With Terminator scoring $\sim$91% Macro-F1 and $\sim$93% accuracy on the validation set, a window of 10 yields fewer than one incorrect prediction on average, making the strict majority (at least 6 of 10) a principled default. Fig. 7 explores the effect of varying the window size while fixing the threshold to 0.7. Performance is stable across all window sizes, confirming that the method is robust to this hyperparameter. As shown in Figures 17–21, once the predicted exit probability exceeds the threshold, it rarely drops below, so the window size has minimal impact on the majority vote.

Thresholding. Fig. 8 shows how the compression rate and the accuracy change as a result of varying the threshold used by TERMINATOR to predict 0 or 1. For example, a threshold $\tau = 0.7$ requires TERMINATOR to have a predictive confidence of at least 0.7 to predict 1. Interestingly, varying the threshold has a greater impact on compression rate than on accuracy; setting $\tau = 0.1$ yields 8% and 17% better compression rates for MATH-500 and HumanEval, respectively, compared to $\tau = 0.9$, with little change in accuracy. For example, from $\tau = 0.9$ to $\tau = 0.1$ yields a 4% drop in accuracy but a 27% improvement in the compression rate for GPQA. Overall, this suggests that TERMINATOR learns a stable confidence signal

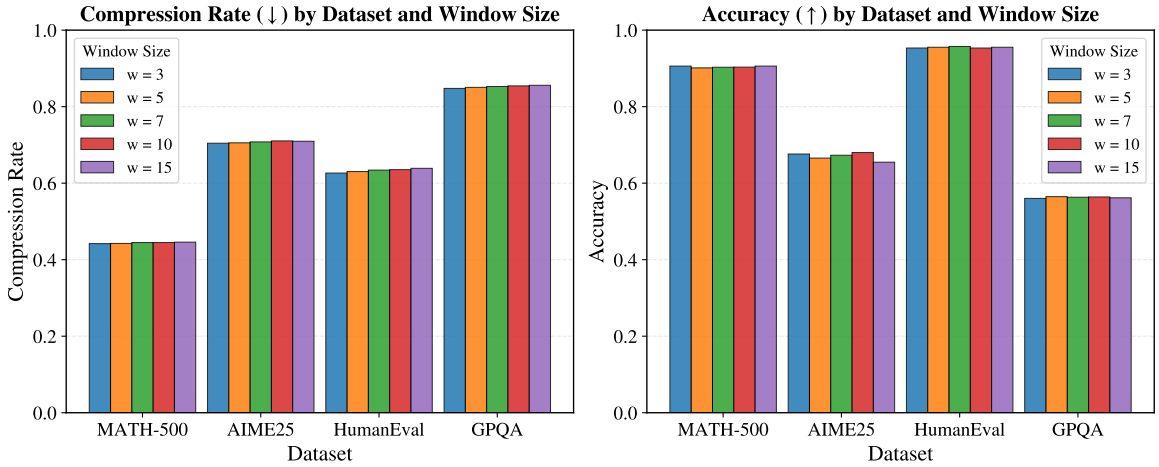

Figure 7. **Effect of Varying Window Size.** We fix the threshold to 0.7 and vary the size of the window used by TERMINATOR at inference-time. TERMINATOR is robust to changes in the window size; Figures 17-21 in our manuscript show that the predicted probability of TERMINATOR during CoT generation tends to only increase. This means that once the predicted probability crosses the threshold, it tends not to fall back below it, and the choice of window size has little or no effect.

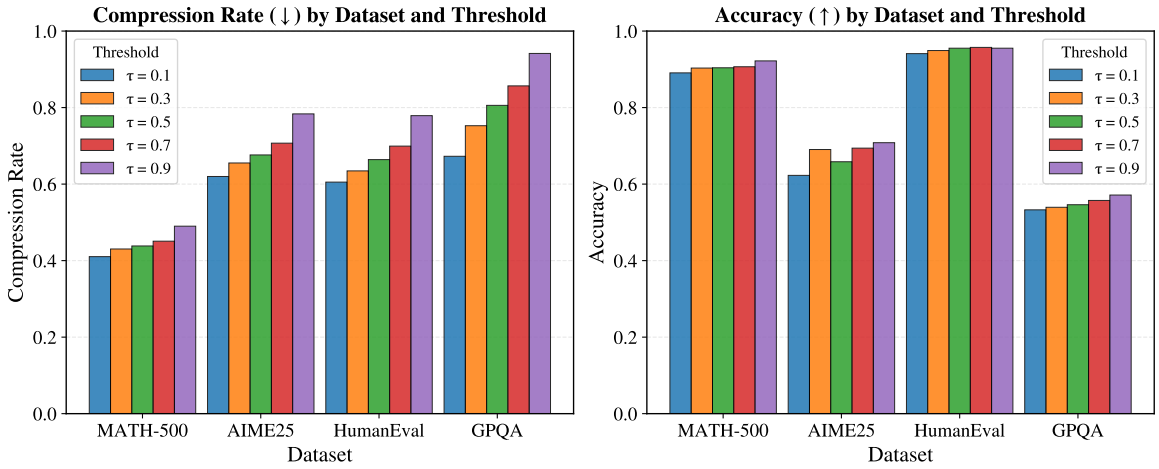

Figure 8. **Effect of Varying Predictive Threshold.** We fix the window size to 10 and vary the threshold τ used by TERMINATOR to predict whether to exit early. TERMINATOR is robust to changes in the predictive threshold: lowering τ makes the method more aggressive, leading to substantially higher compression rates, while accuracy changes only modestly across datasets. In particular, datasets such as MATH-500 and HumanEval exhibit little performance degradation across thresholds from $\tau = 0.1$ to $\tau = 0.9$, suggesting that TERMINATOR learns a stable confidence signal for identifying viable early-exit points.

for identifying viable early-exit points: lowering the threshold makes the method more aggressive, substantially improving compression while only modestly affecting accuracy.

TERMINATOR's Context Length. In general, we stick to training and evaluating TERMINATOR with context lengths up to $32,768$, since those are the lengths of the training CoTs anyway. However, Mistral models have unusually long CoTs only for AIME problems, so we train and evaluate TERMINATOR for a max length of $65,536$ tokens on AIME for Mistral models.

A.3. Implementation of the Baseline Methods

Dynasor. Dynasor (Fu et al., 2025a) works by interrupting reasoning at regular token intervals (e.g. every 32, or 64 tokens) by injecting the prompt "Oh, I suddenly got the answer to the whole problem, Final Answer: boxed{" to extract the model's current answer. The method decides to exit early when consistent answers appear across multiple probing intervals for at least w times. In our experiments, we set w=8 and use a token interval of 64 tokens, following their so-called *mild*

configuration setup.

Thought Calibration. (Wu et al., 2025a) propose training a linear probe to predict optimal stopping points during reasoning generation. The method first segments reasoning trajectories (CoTs) into individual steps, delimited by "\n\n" and containing words like "wait" or "but". Three probe variants are introduced: *Supervised* predicts whether the LRM is correct based on current thoughts; *Consistent* predicts whether the current answer ($\hat{\boldsymbol{a}}$) matches the final answer ($\boldsymbol{a}$); and *Novel Leaf* predicts whether the current step is a leaf node but not novel.

The stopping decision is controlled by two hyperparameters: *tolerance* δ (the maximum acceptable risk of stopping incorrectly) that implies a threshold λ (the calibrated probe score cutoff that triggers stopping), and *window size* (the number of consecutive reasoning steps averaged to smooth predictions before threshold comparison). We retrain the *Supervised* and *Consistent* probes on the S1-K dataset (Muennighoff et al., 2025) for all four models in our experiments: Qwen3-8B, Qwen3-14B, Ministral-3-8B-Reasoning-2512, and Ministral-3-8B-Reasoning-2512. For inference, we use the hyperparameters specified in Table 2. As shown in Table 2, the Qwen3 models require lower thresholds compared to the Ministral models, as their probe outputs yield systematically lower confidence scores. Similar model-specific calibration requirements have been observed in prior work (Yang et al., 2025b).

	Supervised			**Consistent**		
Model	tolerance (δ)	threshold (λ)	window size	tolerance (δ)	threshold (λ)	window size
Qwen3-8B	0.25	0.6526	10	0.25	0.8790	10
Qwen3-14B	0.25	0.6377	10	0.25	0.8679	10
Ministral-3-8B-Reasoning-2512	0.10	0.8173	10	0.025	0.9973	10
Ministral-3-14B-Reasoning-2512	0.10	0.8306	10	0.025	0.9973	10

Table 2. Hyperparameters used for *Supervised* and *Consistent* linear probs for Thought Calibration.

We reported the results for the *Supervised* probe in Table 7, as it performed better across test datasets than the *Consistent* probe. For comparison, we report the results for both probes in Table 3. Note that for the *Consistent* probe with Ministral models, tolerance 0.025 represents the smallest feasible setting among values suggested in (Wu et al., 2025a). The smallest possible setting, being tolerance 0.01, yields threshold=1.0, resulting in no compression.

	MATH-500		**AIME 2025**		**GPQA**		**HumanEval**	
Model	Acc (↑)	CR (↓)	Acc (↑)	CR (↓)	Acc (↑)	CR (↓)	Acc (↑)	CR (↓)
Supervised								
Qwen3-8B	90.1%	93.89%	65.8%	81.54%	52.6%	78.87%	71.8%	92.92%
Qwen3-14B	89.8%	91.92%	63.3%	71.29%	54.6%	81.90%	74.8%	87.12%
Ministral-3-8B-Reasoning-2512	87.7%	87.80%	83.7%	91.16%	47.3%	71.78%	47.2%	87.16%
Ministral-3-14B-Reasoning-2512	87.3%	95.86%	59.4%	79.77%	49.5%	73.81%	27.6%	96.25%
Consistent								
Qwen3-8B	88.2%	72.09%	43.1%	54.64%	44.7%	45.17%	70.7%	73.90%
Qwen3-14B	85.9%	64.40%	41.7%	45.57%	48.3%	53.83%	70.9%	39.60%
Ministral-3-8B-Reasoning-2512	75.8%	80.53%	23.3%	60.50%	42.4%	62.69%	34.2%	75.24%
Ministral-3-14B-Reasoning-2512	68.7%	77.40%	9.4%	34.42%	40.6%	59.97%	9.6%	95.90%

Table 3. Performance comparison of *Supervised* and *Consistent* probes for Thought Calibration across models and tasks.

B. Additional Experimental Results

B.1. Early-Exit Signal Analysis

TERMINATOR Recovers Early-Exit Signals. As discussed in Sec. 5.2, TERMINATOR's early-exit predictions can be used to recover similar early-exit signals as when the ground-truth earliest answer position is used. Figs. 9 and 10 establish this claim. Discussion on our findings is provided in Sec. 5.2

Event-Locked Averaging Token-Confidence. The results shown in Figs. 2 and 11 motivate our approach by confirming that the first arrival of $\hat{\boldsymbol{a}}$ is (1) marked by spiking behavior in the Token-Confidence, which is most easily seen in the event-locked average case, and (2) by a shift in the "thinking token" usage distribution. Focusing on the averaged plots in Fig. 2, we observe that the confidence of the LRM grows up to the point when $\hat{\boldsymbol{a}}$ is first generated, where the confidence finally peaks. The confidence immediately drops after $\hat{\boldsymbol{a}}$ is generated, which is intuitive given that the LRM immediately

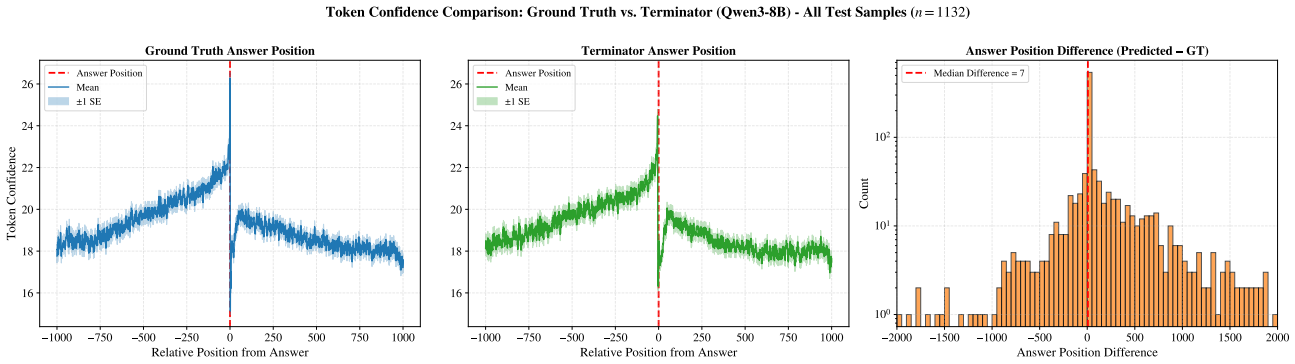

Figure 9. **TERMINATOR Recovers Event-Locked Average Spiking.** The exit positions predicted by TERMINATOR **(center)** recovers similar spiking behavior in the event-locked averaged Token-Confidence as the ground-truth answer positions **(left)**. The histogram of differences between the exit positions **(right)** shows that TERMINATOR's predicted exit positions are close to the ground-truth. Note that the y-axis on the histogram is log-scaled.

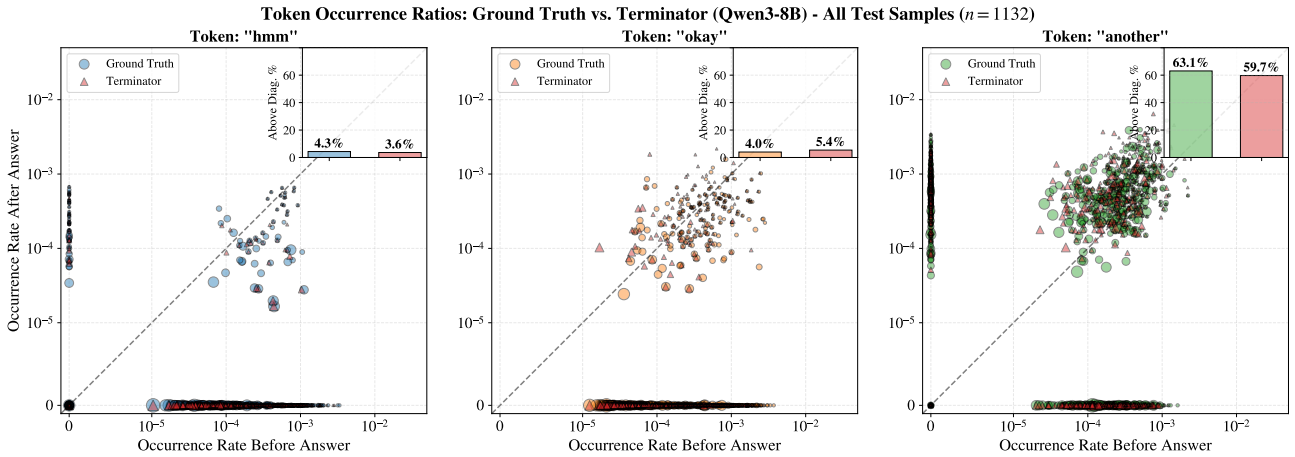

Figure 10. **TERMINATOR Token Usage Biases.** The exit positions predicted by TERMINATOR recover the same biases in the "thinking token" occurrence rates as the ground-truth answer positions. The inset axes on each panel show the percentage of dots that lie above the diagonal when the ground-truth and TERMINATOR answer positions are used.

begins to doubt itself, often producing "thinking tokens" like `wait` or `but`, signaling uncertainty about the answer that was just generated. The LRM's confidence improves slightly as it continues to rethink the problem.

Figs. 16 to 19 show the Token-Confidence and log-probabilities for the single-sample and event-locked averaging case, similar to what's shown in Fig. 2, separately for each data source. While the exact contours of these two signal types vary for different data sources, the same idea applies to all: the LRM's Token-Confidence has a sharp spike at the position of the first occurrence of $\hat{\boldsymbol{a}}$, followed by a sharp decrease. In all cases, the Token-Confidence then has a quick recovery before plateauing or decaying.

Finally, we liken the averaged result in Fig. 2 to the field of *event-related potential* (ERP) research. An ERP is a measurable brain response elicited by a sensory, cognitive, or motor event, captured by electroencephalogram (EEG) recordings (Luck, 2014). However, EEG recordings are often noisy, so ERPs are estimated using time-locked statistical estimators (e.g. averaging) across multiple EEG trials. While we do not claim that our findings will align exactly with ERP research, it is quite interesting that meaningful and observable signals can be extracted from LRMs using similar approaches, and we believe this warrants further exploration in future work.

Token Usage Frequency Shift. Fig. 11 complements Fig. 2 by comparing token frequencies before and after the first $\hat{\boldsymbol{a}}$ occurrence for three "thinking tokens": `hmm`, `okay`, and `another`. These tokens are associated with ongoing reasoning, e.g., `wait`, `so`, `alternatively`, `therefore`, and we hypothesize that their usage changes once $\hat{\boldsymbol{a}}$ has been generated (Wang et al., 2025a; Qian et al., 2025; Ding et al., 2025). Indeed, `hmm` and `okay` occur more often before $\hat{\boldsymbol{a}}$, while `another` occurs more often after it. Although not every thinking token shows a clear before/after bias, these shifts indicate that token-frequency distributions can signal when $\hat{\boldsymbol{a}}$ has appeared; additional examples are shown in Fig. 20. In each scatter plot, the axes show before- and after-$\hat{\boldsymbol{a}}$ token rates, computed as raw counts divided by the corresponding number of tokens,

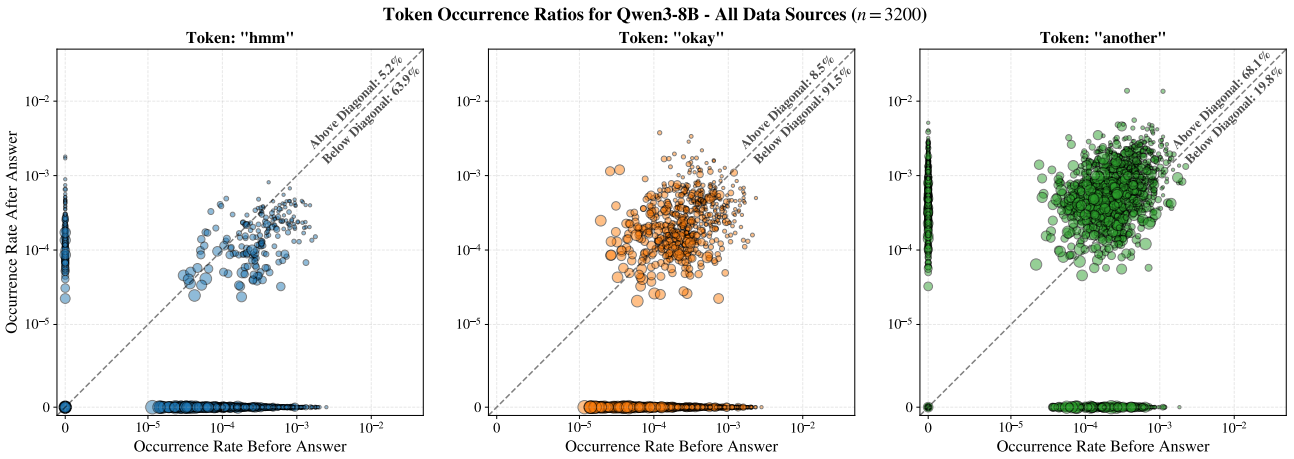

Figure 11. **Token Usage Frequency Shift.** "Thinking token" usage changes depending on whether the final answer has been generated in the CoT. Rates are computed by counting the raw number of occurrences of the token before and after the answer, and then normalizing each count by the respective number of tokens in the before and after bins. The arrival of the final answer is hinted at by changes in the rates for these tokens. The relative length of a CoT is captured by its dot size, where a longer CoT has a larger dot. App. B demonstrates similar results for other "thinking tokens" in Fig. 20 and for each data source in Figs. 21 to 24.

and point diameter indicates relative CoT length.

Beyond providing motivation for our method, the results of Fig. 11 offer some interesting insights on the usage of "thinking tokens." These plots show the strong usage bias that can occur with respect to the first occurrence of $\hat{\boldsymbol{a}}$. For example, 63.9% and 91.5% of CoTs contain the tokens `hmm` and `okay` more often before $\hat{\boldsymbol{a}}$ than after it, respectively. Other tokens, like `another` are more frequently used after. Moreover, Figs. 20 to 24 show that the occurrence rates can differ drastically between data sources. For example, the token `alternatively` has an above-diagonal rate of 80.4% for MATH, but only 19.2% for OpenCoder-SFT.

Fig. 11 also indicates the length of each CoT by its dot size. Upon closer inspection, it appears that there is some correlation between the dot size and the occurrence rates. We plot the before and after occurrence rates with respect to CoT length for these three tokens in Fig. 25. Interestingly, shorter CoTs do in fact correlate with higher token occurrences for these three "thinking tokens."

Fig. 20 shows an expanded view of Fig. 11, including six additional "thinking tokens." Figs. 21 to 24 reproduces this expanded plot, but with the data sources separated. Interestingly, the occurrence rates can vary substantially across different data sources. For example, for the token `alternatively`, only 19.2% of points lie above the diagonal (only 19.2% of CoTs have the token `alternatively` occurring more frequently after the answer than before), and 45.8% (nearly half!) of the points are at the origin (the token `alternatively` never occurs in 45.8% of CoTs) for OpenCoder-SFT. However, for MATH, 80.4% of points lie above the diagonal and only 4.1% lie at the origin for the same token. Other tokens, like `therefore`, are strongly biased toward occurring after the answer.

Fig. 25 shows that the average occurrence rate across CoTs from all data sources changes depending on how long the CoT is. We normalize by the number of tokens occurring before and after the first occurrence of $\hat{\boldsymbol{a}}$ for the before and after rates, respectively, so these plots suggest that the LRM uses `hmm`, `okay`, and `another` more frequently for shorter CoTs than longer ones.

B.2. Adaptive Strategies for TERMINATOR

In principle, adaptive early-exiting strategies offer a promising way to tailor inference to the target domain (Wu et al., 2026). There are two design choices for TERMINATOR that are amenable to inference-time adaptation: (1) domain-based model selection, and (2) threshold selection. Domain-based model selection is concerned with training four separate TERMINATOR models on each of the four training data sets separately and choosing the appropriate one at inference time. This contrasts with our proposed approach, which trains TERMINATOR on all domains simultaneously. Similarly, threshold selection considers adaptivity at inference time by setting a separate threshold for each domain, rather than using a single threshold for all domains. We seek to understand how much performance TERMINATOR can gain, if any, by enabling domain adaptation

Table 4. **Cross-Domain vs. Single-Domain Training.** Comparison of accuracy and compression ratio when TERMINATOR is trained on a single-domain (based on Fig. 13) and all domains (based on Table 7). Cross-domain training (the proposed, unified model) and single-domain training are both valid approaches; cross-domain achieves a 1% increase in accuracy over single-domain, but at a 1.5% worse compression rate.

Method	Math				Coding		Science		Overall	
	MATH-500		AIME25		HumanEval		GPQA			
	Acc↑	CR↓	Acc↑	CR↓	Acc↑	CR↓	Acc↑	CR↓	Acc↑	CR↓
Single-domain training (Fig. 13)	**91%**	47%	66%	**70%**	**96%**	**67%**	55%	**82%**	77%	**66.5%**
Cross-domain training (Table 7)	**91%**	**45%**	**69%**	71%	**96%**	70%	**56%**	86%	**78%**	68%

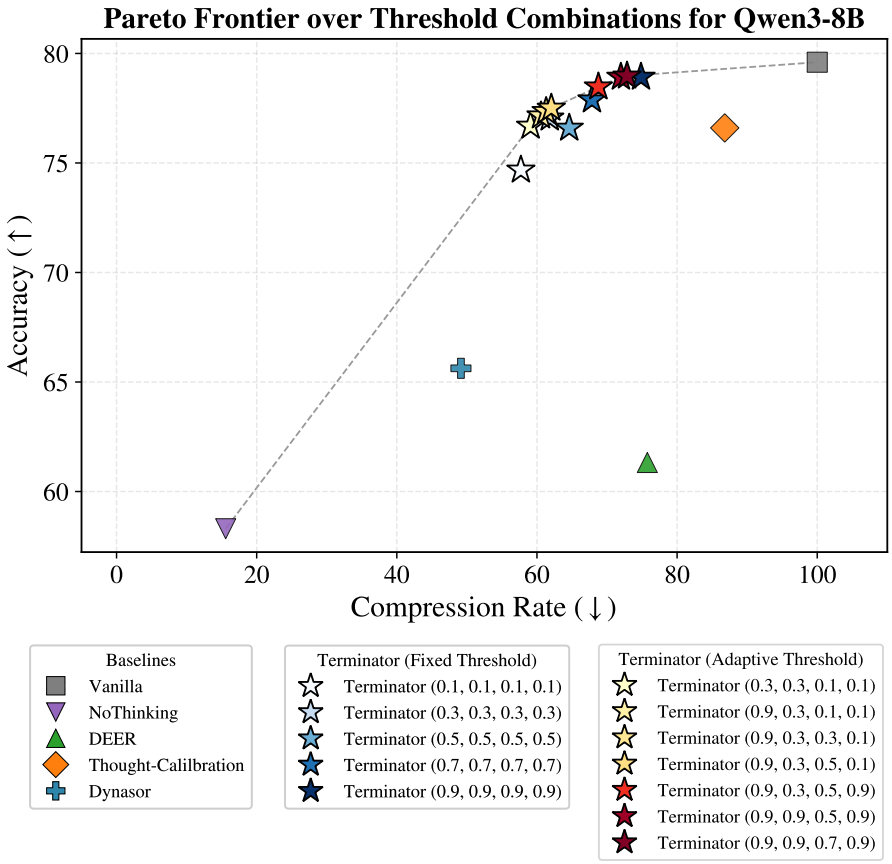

Figure 12. **Adaptive Thresholding for TERMINATOR.** We plot TERMINATOR at various thresholds (blue stars, fixed for all domains) and compare with a domain-adaptive threshold selection (red stars). The tuples correspond to the thresholds chosen for (MATH-500, AIME 25, HumanEval, GPQA) accordingly. For example, "(0.9, 0.3, 0.1, 0.1)" corresponds to choosing thresholds 0.9, 0.3, 0.1, 0.1 for MATH-500, AIME 25, HumanEval, and GPQA, respectively. We enumerate all possible threshold combinations and retain those that define the Pareto frontier. Domain adaptation provides incremental gains but requires domain knowledge at inference time.

on these two design choices.

Table 4 shows the outcome of comparing domain-based model selection with our proposed approach. The domain-based model selection results ("single-domain training") come from the diagonals of Fig. 13, and the results of our proposed method ("cross-domain training") for TERMINATOR come from Table 7. Overall, both approaches yield good performance, without a clear winner: cross-domain training achieves 1% higher accuracy than single-domain training, but at a 1.5% worse compression rate.

Fig. 12 shows the best possible performance with a domain-adaptive thresholding approach for TERMINATOR for five possible thresholds: $0.1, 0.3, 0.5, 0.7, and 0.9$. In the figure, the tuples correspond to the thresholds chosen for (MATH-

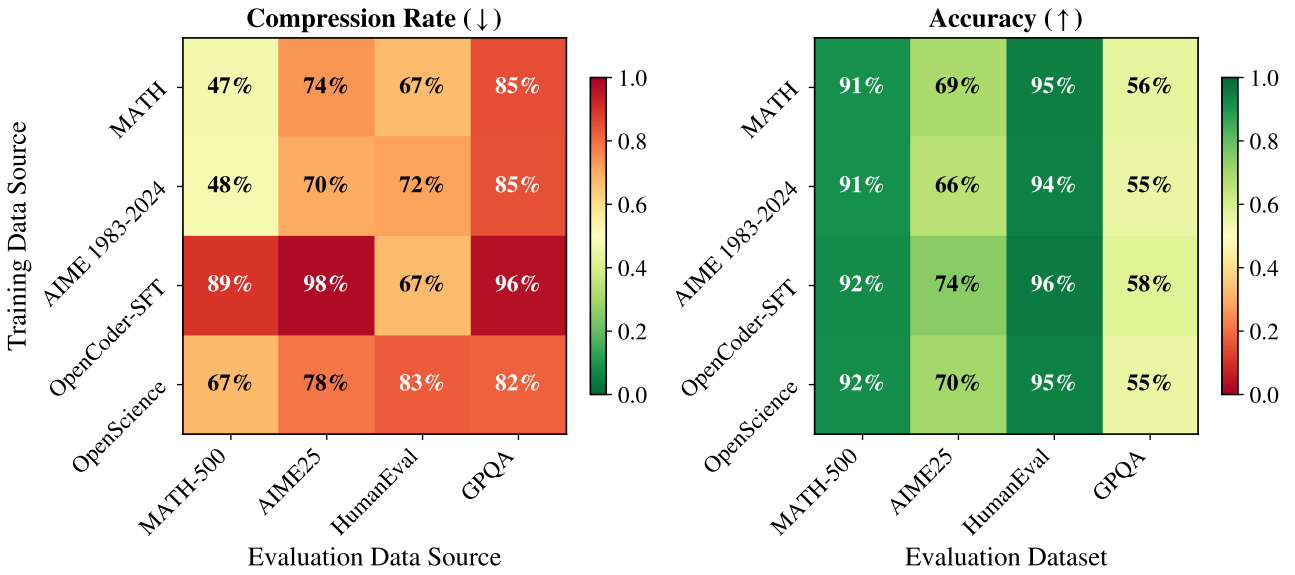

Figure 13. **OOD Performance of TERMINATOR.** The best trade-off between accuracy and compression rate is achieved when the evaluation set is in-distribution with the training dataset. Here the out-of-distribution performance of TERMINATOR with respect to the compression rate **(left)** and the accuracy **(right)** for Qwen3-8B is shown. Training datasets are listed along the row axis, and the evaluation sets are listed across the column axis. For example, training TERMINATOR on MATH and evaluating on HumanEval yields a compression rate of 67% and an accuracy of 95%. Every training dataset has an in-domain evaluation dataset, i.e. MATH → MATH-500, AIME 1983–2024 → AIME25, OpenCoder-SFT → HumanEval, and OpenScience → GPQA.

500, AIME 25, HumanEval, GPQA) accordingly. For example, "$(0.9, 0.3, 0.1, 0.1)$" corresponds to choosing thresholds $0.9, 0.3, 0.1, 0.1$ for MATH-500, AIME 25, HumanEval, and GPQA, respectively. Five thresholds across four datasets yield $5^4 = 625$ possible combinations, so we show only those that define the Pareto frontier. We use a threshold of 0.7 (represented with "$(0.7, 0.7, 0.7, 0.7)$" in Fig. 12) for TERMINATOR. While domain-adaptive thresholding (red stars) generally lies above fixed thresholding (blue stars), the improvement in performance is marginal.

However, we note that the results for these domain-adaptive are an upper bound to the performance that can be realized in practice, since perfect knowledge of which domain a given problem belongs to is assumed. In practice, an inference-time strategy is required to appropriately choose the model or threshold, which may introduce error and additional computational overhead. Furthermore, the domain-based model selection approach requires four separately trained models rather than a single unified model, thereby incurring additional compute and memory overhead for model routing.

B.3. TERMINATOR Prediction Analysis

TERMINATOR Predictions During Reasoning. Fig. 26 shows the event-locked average of the predicted probabilities from TERMINATOR for each data source, and Fig. 27, Fig. 28, Fig. 29, and Fig. 30 show the predicted probabilities from four randomly chosen examples for AIME25, MATHH-500, HumanEval, and GPQA, respectively. The event-locked average and the individual examples from MATH-500 and HumanEval show sharp transitions in predicted confidence at the exiting threshold, with good separation (dotted gray line). However, AIME25 and GPQA examples do not show such a sharp transition, suggesting that it is challenging for TERMINATOR to identify a good exit position for very hard tasks.

Answer Prediction Histograms. Fig. 31 gives an overview of the position of the first occurrence of $\hat{\boldsymbol{a}}$ in the CoTs for each data source we used for training. Fig. 32 shows the compression rate statistics of TERMINATOR for each test dataset.

Case Study of TERMINATOR. By manual inspection, we have seen that easy problems have a clearer transition to overthinking, which is well detected by TERMINATOR. However, TERMINATOR may not always detect the appropriate answer position cleanly on harder questions. Figs. 33 to 36 demonstrate this behavior on Qwen3-14B CoTs.

B.4. Pipeline Cost-Benefit Analysis

Gathering $110,799$ CoTs for training takes approximately 3 days with a single GH200. The pipeline for obtaining the position of $\hat{\boldsymbol{a}}$ on those CoTs takes roughly one week on 8 GH200s, totaling an estimated $1,416$ GPU-hours. Fig. 14 provides

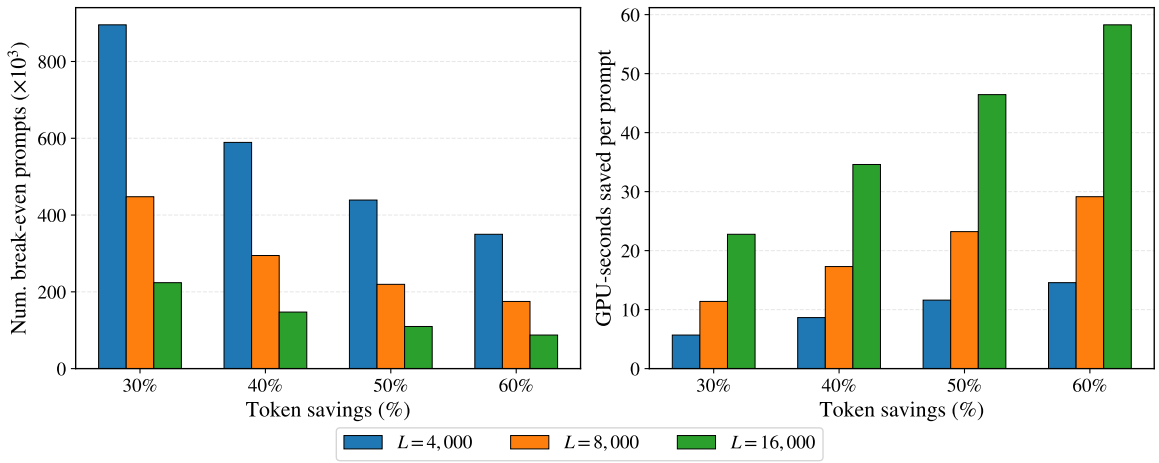

Figure 14. **Pipeline Cost-Benefit Analysis.** We report an estimate on the number of prompts to TERMINATOR needed to break even with the compute cost associated with running our data curation pipeline **(left)** and on the number of GPU-seconds saved per prompt **(right)**. We use the Qwen3-8B throughput data from Table 1 and the cost across fixed token savings (i.e. a reduction in the number of tokens by $30\%, 40\%, 50\%$, and 60%) and fixed prompt lengths of $4,000, 8,000$ and $16,000$ tokens. At the time of writing, the number of *monthly* Qwen3-8B downloads from Hugging Face is $\sim$9.9 million (Hugging Face, 2026). If only a single prompt were run per download on Qwen3-8B with TERMINATOR, the compute offset would be compensated for many times over.

a rough estimate of the number of prompts needed to offset the one-time data curation compute cost to train Terminator with Qwen3-8B (left), along with the number of seconds saved per prompt. The x-axis corresponds to the fixed token savings across all prompts, and L represents the length of the CoTs. For example, for a 30% savings, about $\sim 200,000$ prompts with CoT lengths of $16,000$ are sufficient to break even with the data curation step. The number of prompts required to break even is small compared to the number of monthly downloads for Qwen3-8B on Hugging Face ($\sim$9.9 million at the time of writing (Hugging Face, 2026)). If just a single prompt were run per download on Qwen3-8B with TERMINATOR, the compute offset for the training data curation would be compensated for many times over.

B.5. Disentangling Hindsight-Optimal Labels and Token-Level Exit Prediction

We analyze the relative roles of two design choices in TERMINATOR: (1) hindsight-optimal exit labels and (2) token-level exit prediction. These two ingredients are closely coupled. Hindsight-optimal labels provide the supervision signal that makes token-level predictions meaningful, while token-level predictions are needed to fully exploit the fine-grained structure of those labels. To isolate their effects, we consider all four combinations of hindsight-optimal versus consistency-based labels and token-level versus chunk-level prediction.

We first describe the two alternatives considered in this analysis.

Consistency-based Labels. Consistency-based early-exit methods inject `</think>` at intermediate points in the CoT and force the model to produce a final answer. One common labeling strategy is to assign a positive exit label to the first intermediate point whose forced answer matches the model's final answer $\hat{a}$. In contrast, our hindsight-optimal labeling assigns a positive label at the first point where $\hat{a}$ itself appears in the CoT. Thus, rather than repeatedly probing whether the model can already reproduce the final answer, hindsight-optimal labeling directly identifies when the final answer has emerged in the reasoning trajectory.

Chunk-level Prediction. Prior early-exit approaches typically make exit decisions only at predefined checkpoints during generation. These checkpoints may occur at fixed intervals, after structural delimiters such as `\n\n`, or after heuristic "thinking tokens" such as `wait`, `alternatively`, or `therefore`. In contrast, TERMINATOR predicts whether to exit at every generated token.

The four possible combinations are summarized in Table 5.

Hindsight-optimal Labels with Token-level Prediction. This is our proposed method, TERMINATOR. It uses hindsight-optimal labels to identify the earliest point at which the final answer appears in the CoT, and trains a token-level predictor to decide whether generation can safely terminate at each position.

Table 5. **Combinations of exit supervision and prediction granularity.** TERMINATOR combines hindsight-optimal labels with token-level prediction. The remaining combinations either correspond to prior work, underperform our method, or are computationally infeasible.

	Token-level prediction	**Chunk-level prediction**
Hindsight-optimal labels	TERMINATOR (proposed)	Underperforms TERMINATOR
Consistency-based labels	Computationally infeasible	Prior work (underperforms)

Table 6. **Effect of chunk-level exit prediction.** We train a chunk-level TERMINATOR variant using the `\n\n` delimiter so that, for each CoT, the positive exit label is assigned to the end of the chunk containing the hindsight-optimal exit position. At inference time, exit predictions are made only at chunk boundaries. All other hyperparameters, including the exit threshold, are kept the same as in TERMINATOR.

Method	**Math**				**Coding**		**Science**			
	MATH-500		**AIME25**		**HumanEval**		**GPQA**		**Overall**	
	Acc↑	CR↓	Acc↑	CR↓	Acc↑	CR↓	Acc↑	CR↓	Acc↑	CR↓
TERMINATOR (Proposed)	90.7%	45.1%	69.4%	70.7%	95.7%	69.9%	55.7%	85.7%	77.9%	67.8%
Chunk Variant	80.8%	18.1%	34.9%	43.0%	86.6%	18.4%	45.4%	16.9%	61.9%	24.1%

Hindsight-optimal Labels with Chunk-level Prediction. To test whether hindsight-optimal labels alone are sufficient, we also train a variant that only makes exit predictions at chunk boundaries, following the prediction granularity used in prior early-exit work. Specifically, we use the `\n\n` delimiter to define chunks and assign the target exit to the end of the chunk containing the hindsight-optimal exit position. At inference time, predictions are made only at chunk boundaries.

Table 6 shows that this variant substantially underperforms token-level TERMINATOR. Although the chunk-level variant exits much less aggressively, achieving substantially lower compression rates, it also suffers large accuracy drops across evaluation datasets. For example, the chunk-level variant achieves an overall accuracy of 56.7% and compression rate of 24.1%, compared to 73.7% accuracy and 67.8% compression for TERMINATOR. The gap is particularly large on AIME25 and GPQA, where coarse exit checkpoints appear unable to reliably capture the precise point at which the model has completed the necessary reasoning. These results indicate that token-level granularity is necessary to fully leverage hindsight-optimal supervision.

Consistency-based Labels with Token-level Prediction. A token-level consistency-based method would require injecting `</think>` after every generated token and forcing the model to produce an intermediate final answer at each position. This is computationally prohibitive. For example, assuming a conservative overhead of 1 second per forced-answer generation, labeling a single CoT with 8,000 tokens would require roughly 2.22 hours of compute. Using the same compute budget as for our training data ($1,416$ GPU hours, see App. B.4), this would label only a small fraction (about 700) of the available CoTs ($110,799$ in total), yielding too little data for effective training. Thus, while token-level consistency labels are conceptually possible, they are not practical at scale.

Consistency-based labels with chunk-level prediction. This setting corresponds to the dominant design used in prior early-exit work, where the model is probed at coarse checkpoints and exits when an intermediate forced answer is consistent with the final answer. Our main results show that TERMINATOR outperforms these methods. Moreover, the hindsight-optimal chunk-level variant in Table 6 performs worse than token-level TERMINATOR, despite using our stronger labeling signal. This demonstrates that hindsight-optimal labels alone are not sufficient: they define a better target, but token-level prediction is needed to realize their benefit.

Overall, these results suggest that hindsight-optimal labels set the achievable performance ceiling by identifying meaningful early-exit positions, while token-level prediction provides the granularity needed to reach that ceiling.

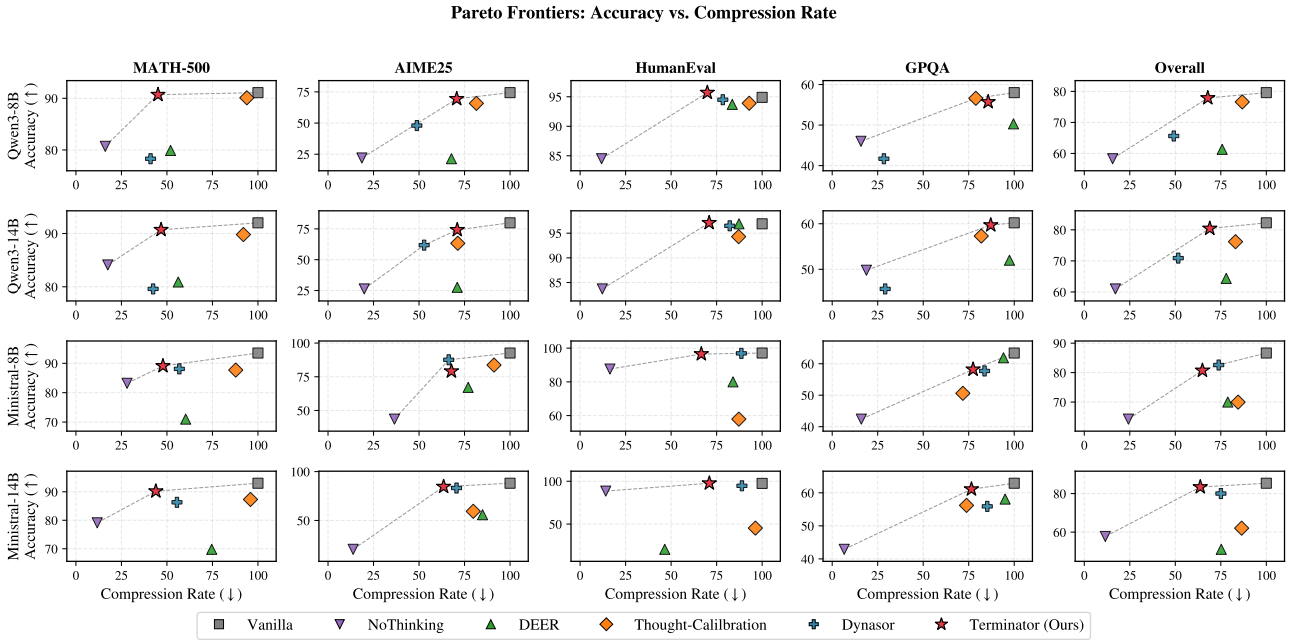

Figure 15. **Pareto Frontier.** TERMINATOR defines the Pareto frontier on 14 of the 16 (LRM, benchmark) pairings, outperforming prior work. Each point represents a method's accuracy and compression rate, with lower compression rates indicating greater token savings and hence compute. The dashed line traces the Pareto frontier connecting non-dominated solutions. We refer to Sec. 5.1 and Table 7 in App. B for more details.

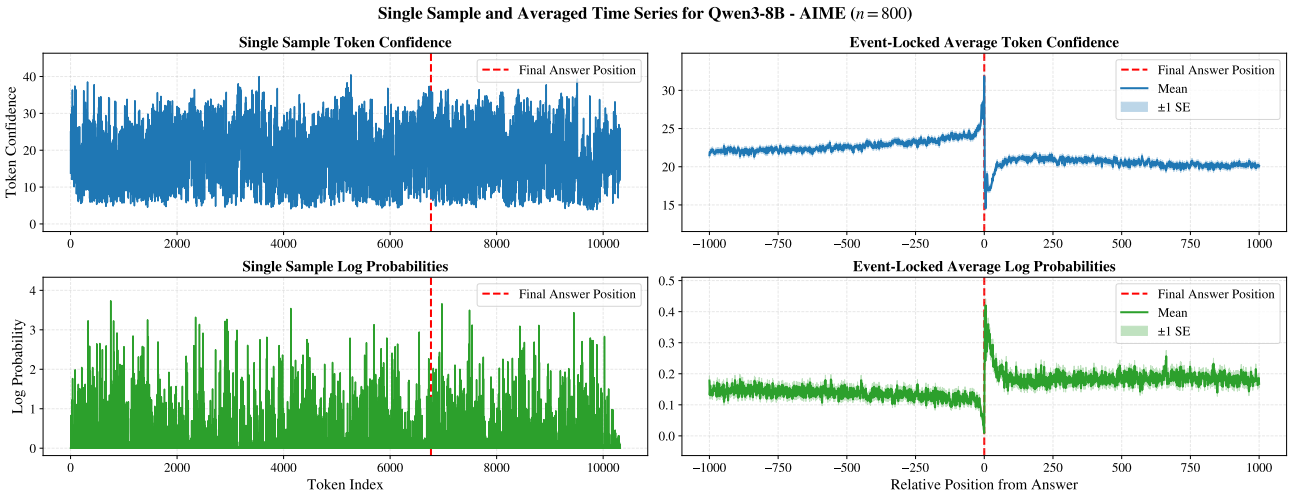

Figure 16. **Event-Locked Averaging of Token-Confidence for AIME (1983–2024).** A reproduction of Fig. 2, but only using CoTs from 800 randomly selected AIME (1983–2024) problems.

Table 7. **Performance of TERMINATOR and Baselines.** ↑ Indicates that higher values are better, while ↓ indicates that lower values are better. CR is the compression rate, reported here as the mean per-sample compression rate. Tok is the mean number of tokens per sample. **Bold** and Underlined values highlight the best and second-best performing early exit methods, respectively. TERMINATOR demonstrates superior accuracy-efficiency trade-offs (best or second-best performance across 28 out of 32 metrics). Fig. 15 in Sec. 5 shows the results of this table on the Pareto frontier.

Method	Math						Coding			Science			Overall	
	MATH-500			AIME25			HumanEval			GPQA				
	Acc↑	Tok↓	CR↓	Acc↑	Tok↓	CR↓	Acc↑	Tok↓	CR↓	Acc↑	Tok↓	CR↓	Acc↑	CR↓
Qwen3-8B														
Vanilla	91.1%	5,037	100%	74.4%	14,499	100%	94.9%	3,792	100%	58.0%	8,594	100%	79.6%	100%
NoThinking	80.7%	809	16.1%	22.0%	2,355	18.6%	84.6%	353	11.8%	46.0%	1,204	15.8%	58.3%	15.6%
DEER	79.9%	2,602	52.0%	21.4%	10,349	67.8%	93.7%	3,275	83.6%	50.3%	8,553	99.6%	61.3%	75.8%
Thought-Calib	90.1%	4,372	93.9%	65.8%	11,014	81.5%	93.9%	3,267	92.9%	**56.6%**	6,240	78.9%	76.6%	86.8%
Dynasor	78.3%	1,850	**41.0%**	48.0%	7,479	**48.8%**	94.5%	2,883	78.4%	41.7%	2,455	**28.4%**	65.6%	**49.2%**
TERMINATOR	**90.7%**	2,425	45.1%	**69.4%**	10,970	70.7%	**95.7%**	2,716	**69.9%**	55.7%	7,543	85.7%	**77.9%**	67.8%
Qwen3-14B														
Vanilla	92.0%	4,598	100%	79.9%	14,255	100%	96.9%	3,296	100%	60.2%	7,628	100%	82.3%	100%
NoThinking	84.1%	786	17.5%	26.3%	2,472	19.9%	83.7%	317	12.2%	49.8%	1,265	18.8%	61.0%	17.1%
DEER	80.9%	2,501	56.2%	27.6%	10,497	71.0%	96.9%	2,961	87.3%	52.0%	7,451	97.4%	64.5%	78.0%
Thought-Calib	89.8%	3,778	92.0%	63.3%	9,429	71.3%	94.3%	2,582	87.1%	**57.3%**	5,757	81.9%	76.2%	83.1%
Dynasor	79.6%	1,702	**42.4%**	61.8%	7,937	**52.8%**	96.5%	2,611	82.2%	45.7%	2,101	**29.1%**	70.9%	**51.6%**
TERMINATOR	**90.7%**	2,261	46.8%	**74.2%**	10,787	71.0%	**97.1%**	2,358	**70.9%**	59.6%	6,798	87.1%	**80.4%**	68.9%
Ministral-3-8B-Reasoning-2512														
Vanilla	93.5%	6,212	100%	92.6%	22,124	100%	97.1%	4,367	100%	63.4%	11,765	100%	86.6%	100%
NoThinking	83.2%	1,908	28.1%	43.6%	7,711	36.5%	87.6%	727	16.4%	42.4%	2,106	16.0%	64.2%	24.3%
DEER	71.0%	3,791	60.3%	67.1%	17,481	77.0%	80.1%	3,606	84.0%	**61.9%**	11,312	94.1%	70.0%	78.9%
Thought-Calib	87.7%	5,695	87.8%	83.7%	20,358	91.2%	57.9%	3,536	87.2%	50.6%	7,406	**71.8%**	70.0%	84.5%
Dynasor	88.1%	2,967	56.8%	**87.6%**	15,407	**66.3%**	**96.9%**	3,931	88.6%	57.7%	9,766	83.7%	**82.6%**	73.9%
TERMINATOR	**89.1%**	2,863	**47.8%**	57.4%	15,239	67.1%	96.5%	2,960	**66.6%**	58.2%	9,588	77.4%	75.3%	**64.7%**
Ministral-3-14B-Reasoning-2512														
Vanilla	93.0%	6,385	100%	88.1%	23,694	100%	97.5%	3,918	100%	62.9%	9,539	100%	83.4%	100%
NoThinking	79.1%	535	11.7%	20.5%	2,413	13.8%	88.5%	528	14.1%	42.8%	570	6.6%	57.75%	11.5%
DEER	69.8%	4,279	74.6%	55.9%	20,049	84.9%	20.5%	1,684	**46.5%**	58.1%	9,185	95.0%	51.1%	75.3%
Thought-Calib	87.3%	5,860	95.9%	59.4%	17,763	79.8%	45.3%	3,465	96.3%	56.2%	6,028	**73.8%**	62.1%	86.5%
Dynasor	86.3%	3,240	55.5%	**83.2%**	17,920	70.6%	94.7%	3,538	88.9%	55.9%	7,917	85.2%	**80.0%**	75.1%
TERMINATOR	**90.2%**	2,946	**43.9%**	65.8%	15,898	**68.7%**	**97.8%**	2,903	71.0%	**61.2%**	7,727	76.5%	78.7%	**65.0%**

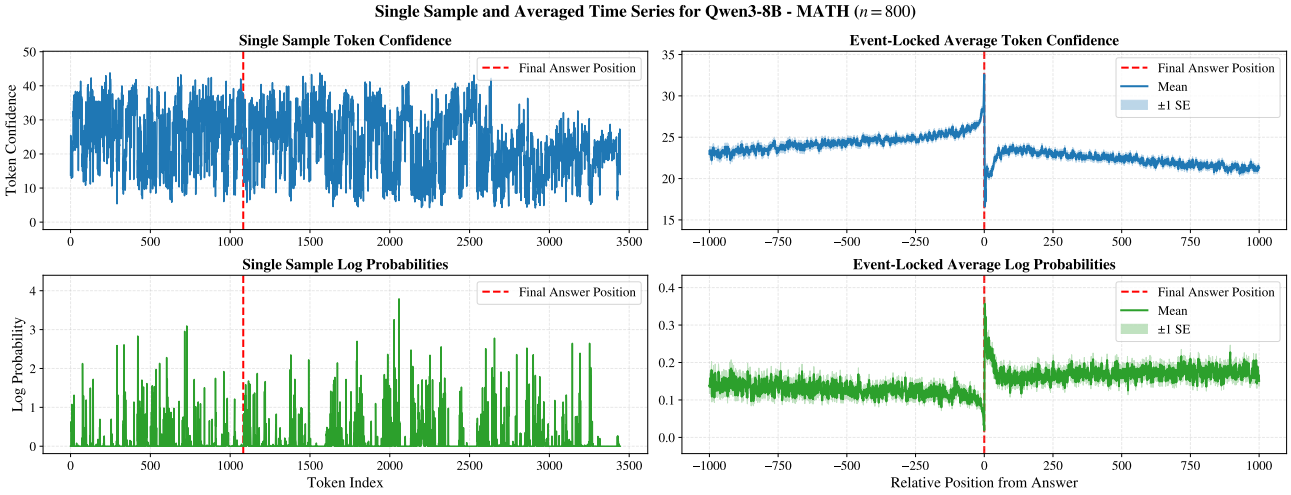

Figure 17. **Event-Locked Averaging of Token-Confidence for MATH.** A reproduction of Fig. 2, but only using CoTs from 800 randomly selected MATH problems.

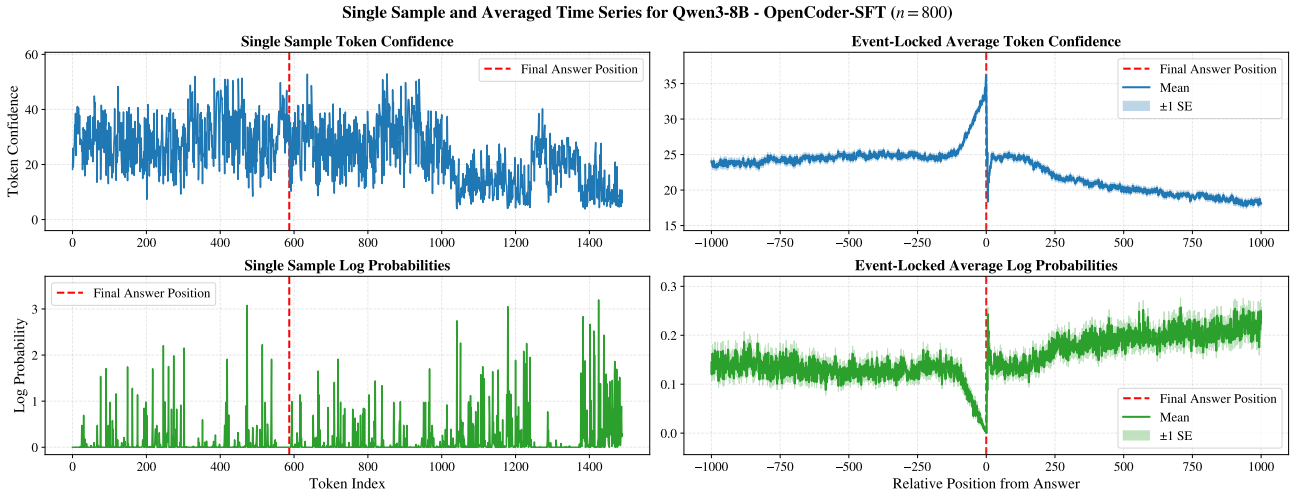

Figure 18. **Event-Locked Averaging of Token-Confidence for OpenCoder-SFT.** A reproduction of Fig. 2, but only using CoTs from 800 randomly selected OpenCoder-SFT problems.

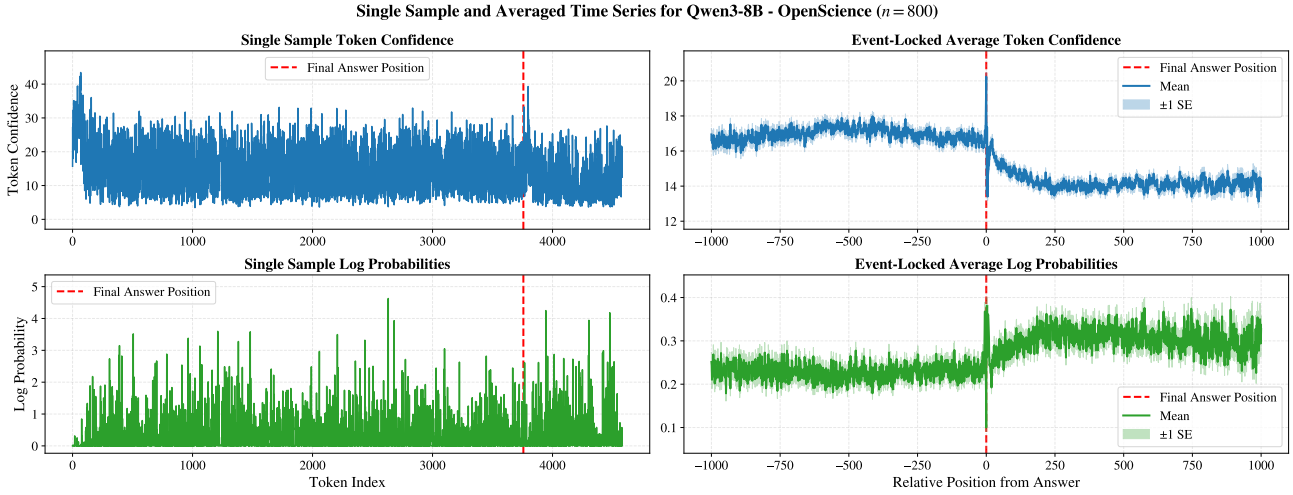

Figure 19. **Event-Locked Averaging of Token-Confidence for OpenScience.** A reproduction of Fig. 2, but only using CoTs from 800 randomly selected OpenScience problems.

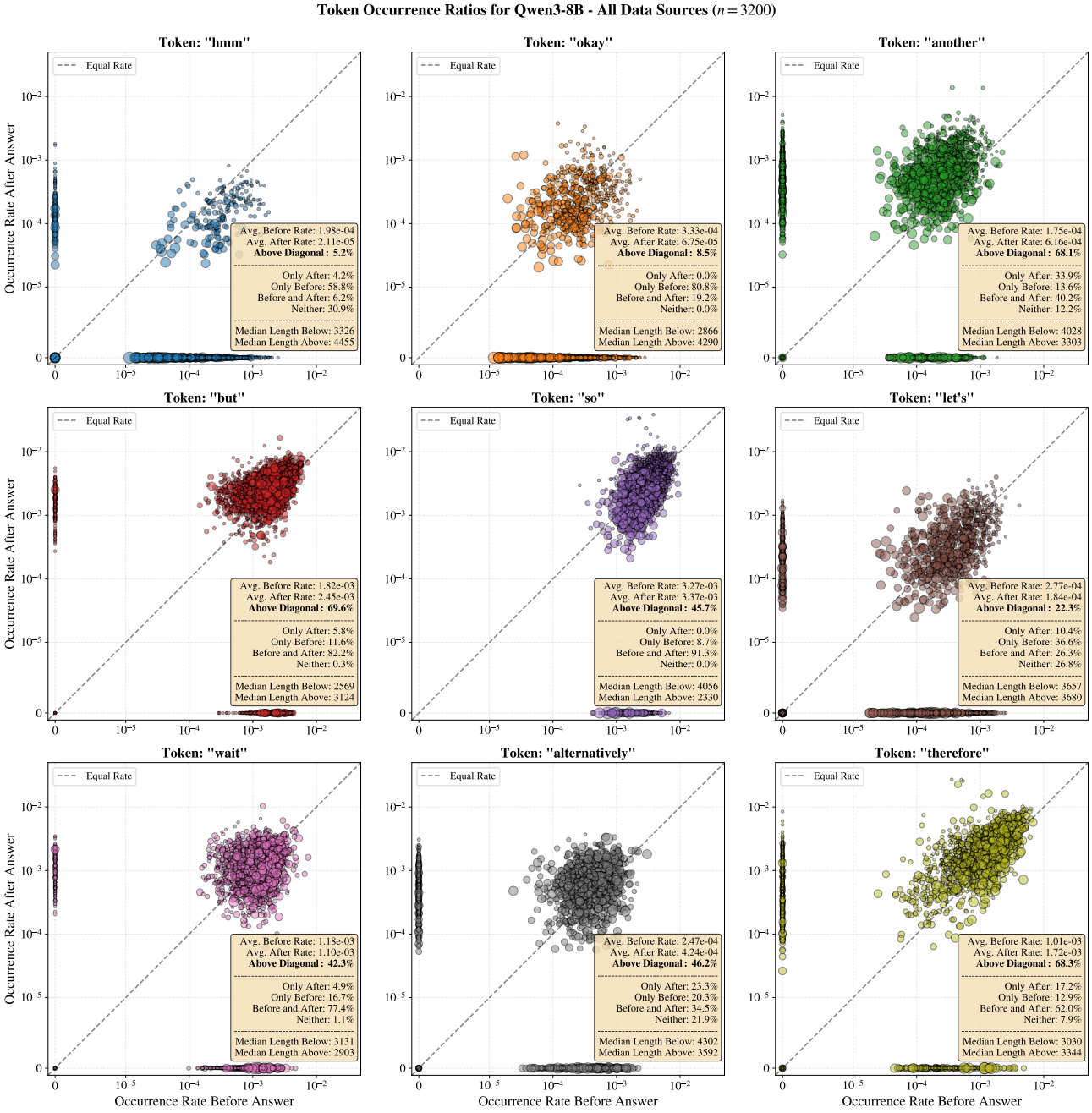

Figure 20. **Token Usage Frequency Shift.** An extension of the results shown in Fig. 11, highlighting additional "thinking tokens." While most "thinking tokens" shown here have some bias, often occurring before and after the first occurrence of the final answer as indicated by the **Above Diagonal** statistic, some tokens, like "so," are close to an equal rate on average.

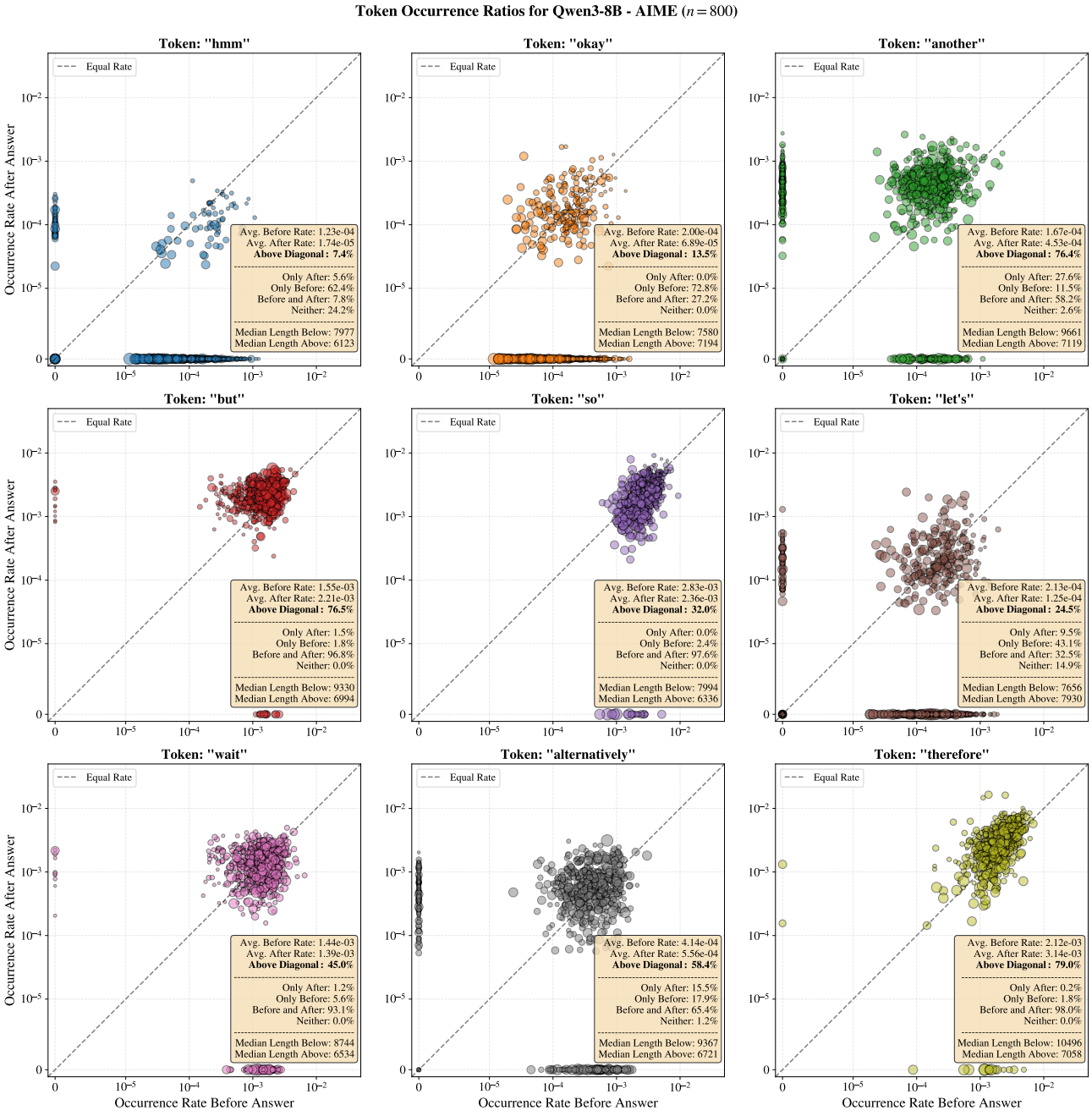

Figure 21. **Token Usage Frequency Shift for AIME (1983–2024.** A reproduction of Fig. 20, but only using CoTs from 800 randomly selected AIME (1983–2024) problems.

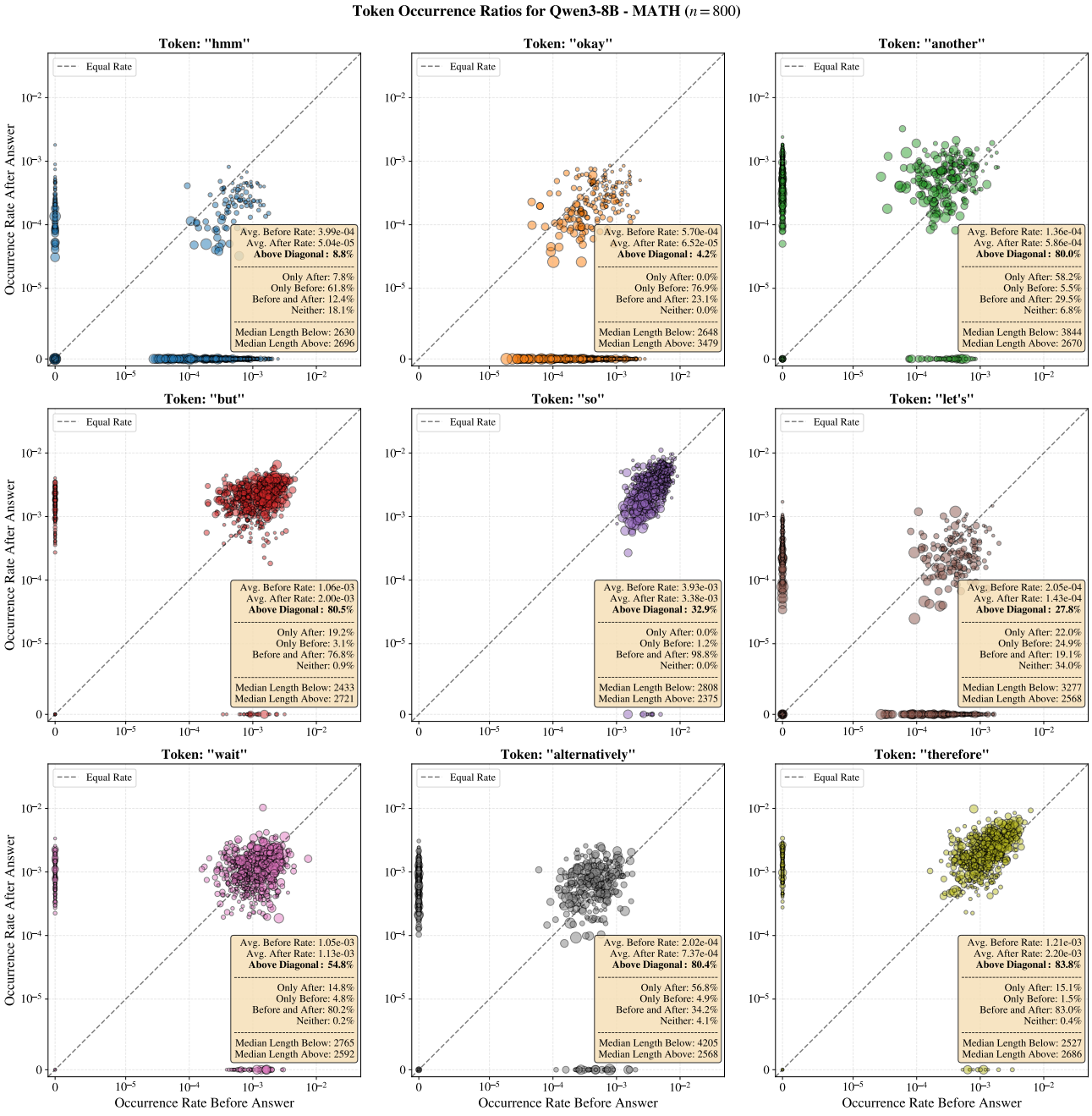

Figure 22. **Token Usage Frequency Shift for MATH.** A reproduction of Fig. 20, but only using CoTs from 800 randomly selected MATH problems.

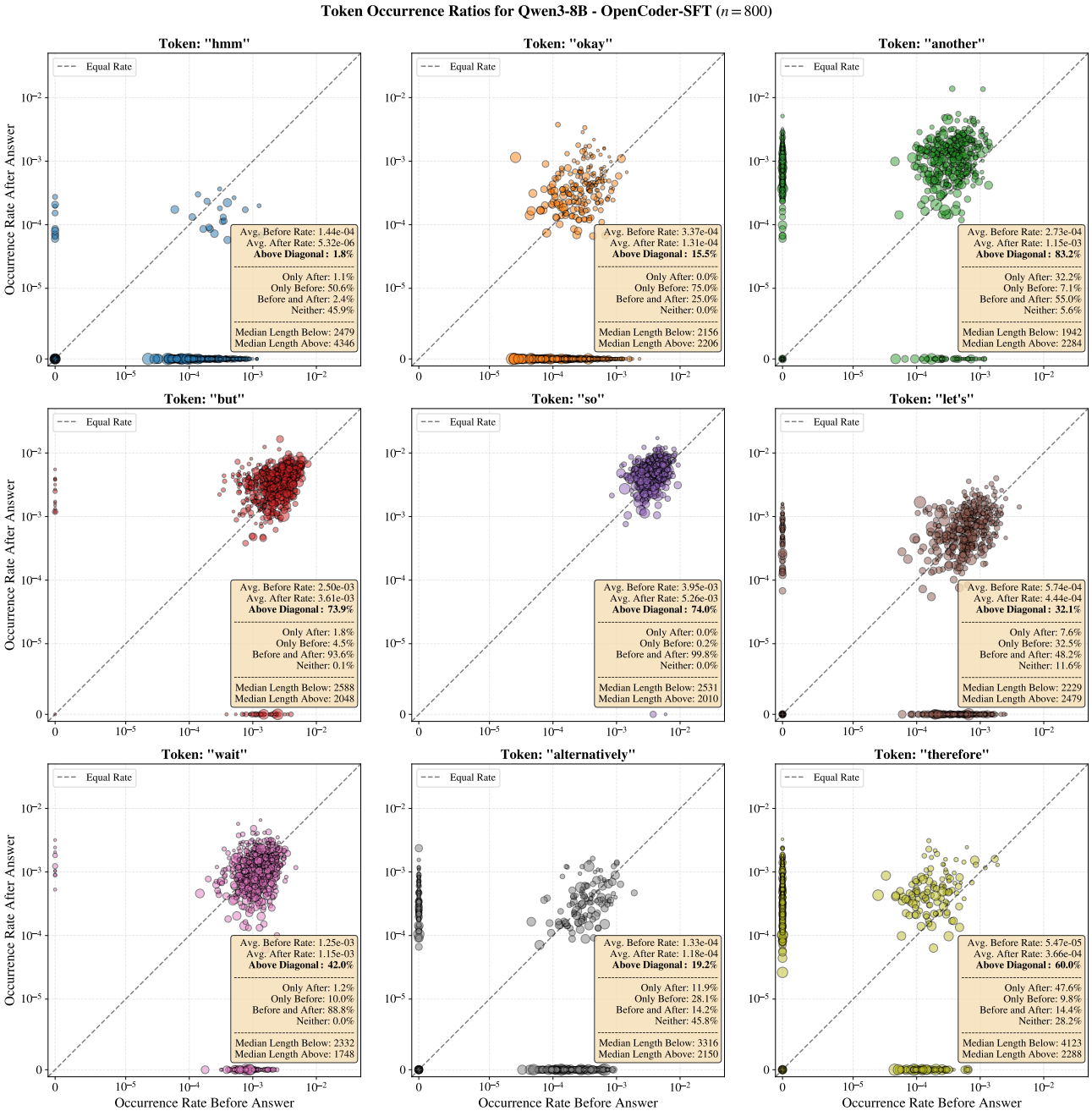

Figure 23. **Token Usage Frequency Shift for OpenCoder-SFT.** A reproduction of Fig. 20, but only using CoTs from 800 randomly selected OpenCoder-SFT problems.

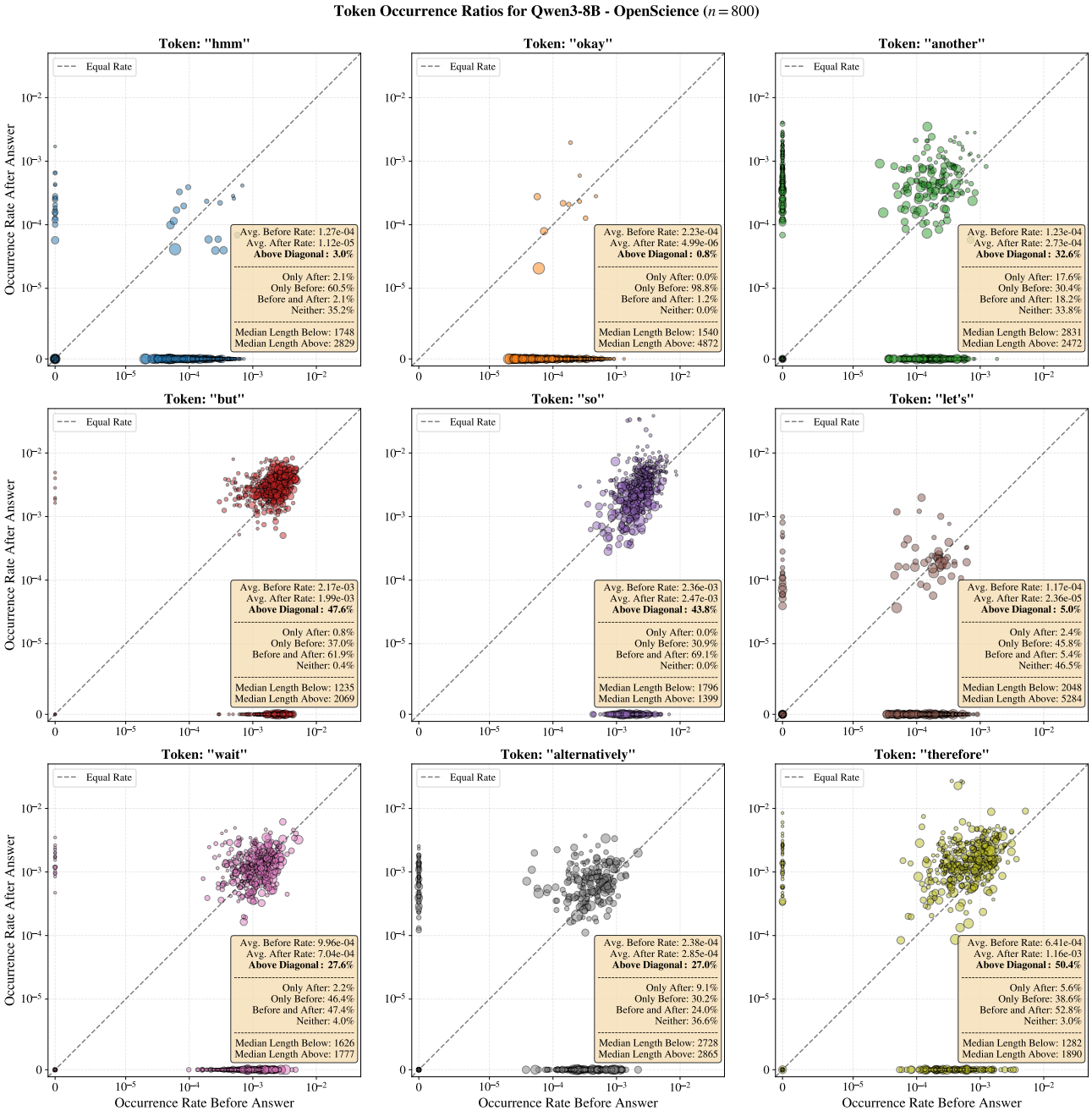

Figure 24. **Token Usage Frequency Shift for OpenScience.** A reproduction of Fig. 20, but only using CoTs from 800 randomly selected OpenScience problems.

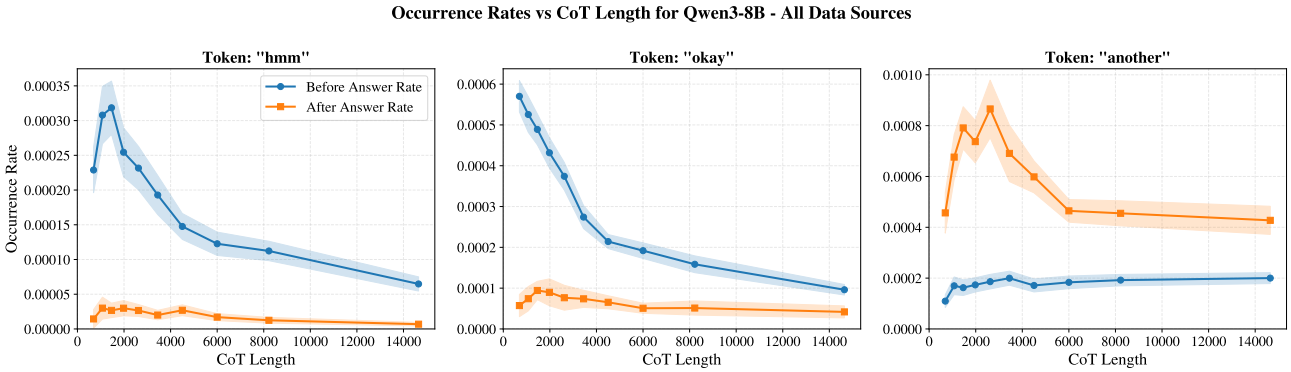

Figure 25. **Token Occurrence Rate vs CoT Length.** For some tokens, such as these three "thinking tokens," occurrence rates decrease rapidly as CoT length increases. Interestingly, the side of the answer (either "before" or "after") with the highest rate is the one that decays the most (the "before" rate for `hmm` and `okay` and the "after" rate for `another`), while the side with the lower rate sees only a slight decrease or increase. For each plot, the lengths are placed into ten bins with percentile-based bin edges. In other words, each bin contains approximately 10% of the samples. Shaded regions indicate the 95% confidence interval.

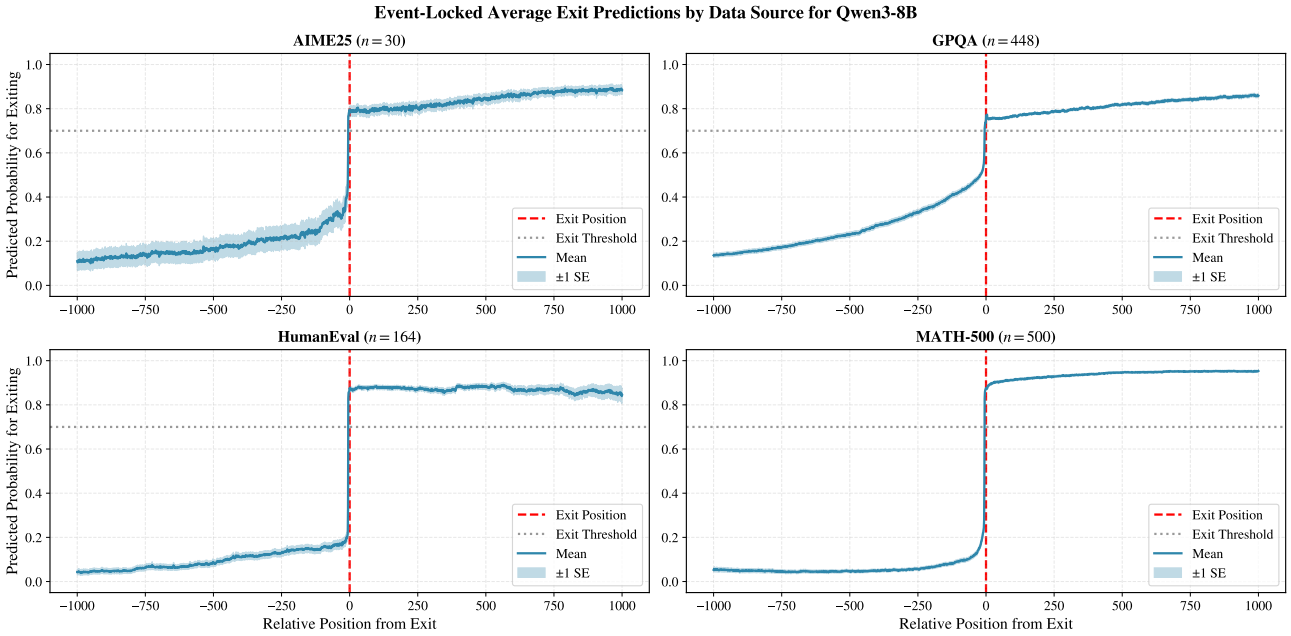

Figure 26. **Predicted Probabilities Event-Locked Averaging.** The dashed vertical line shows where TERMINATOR terminates the CoT with a sliding window of 10 and an exit threshold of 0.7, as indicated by the horizontal dotted line. We show the average predicted probability stream across all test problems from MATH-500, AIME25, HumanEval, and GPQA. Figs. 27 to 30 show predictions streams from individual, randomly drawn samples from each data source.

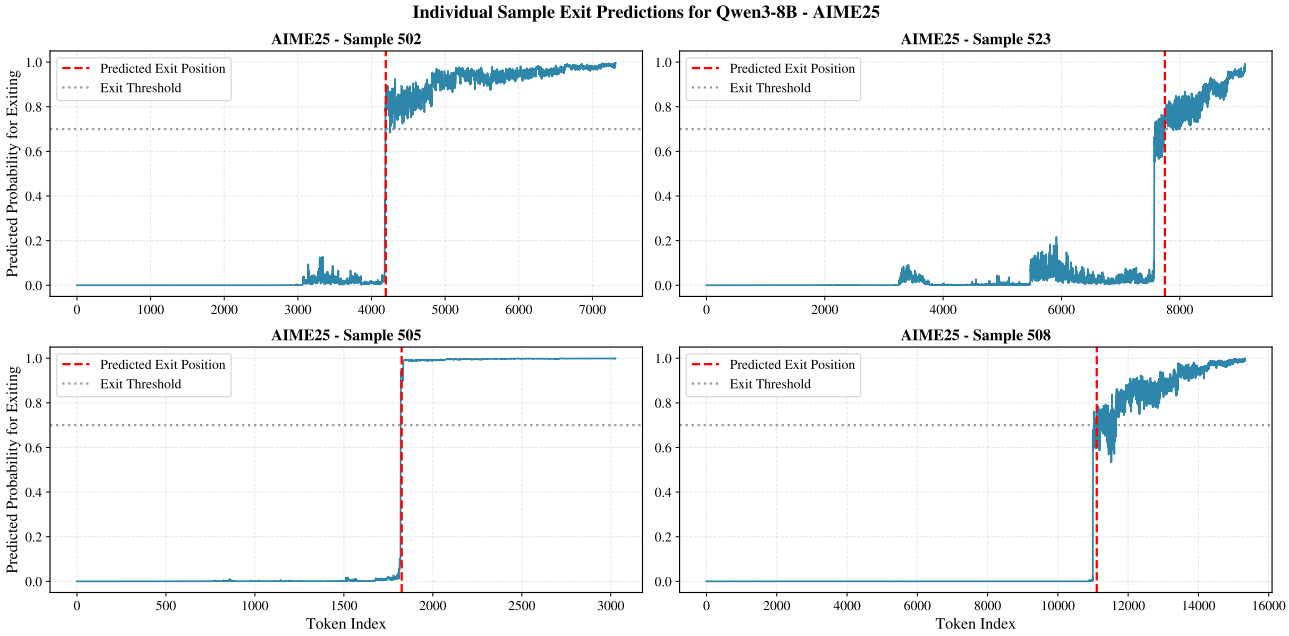

Figure 27. **Predicted Probabilities for AIME25.** TERMINATOR's predicted probability stream for early-exiting on four randomly chosen samples from AIME25.

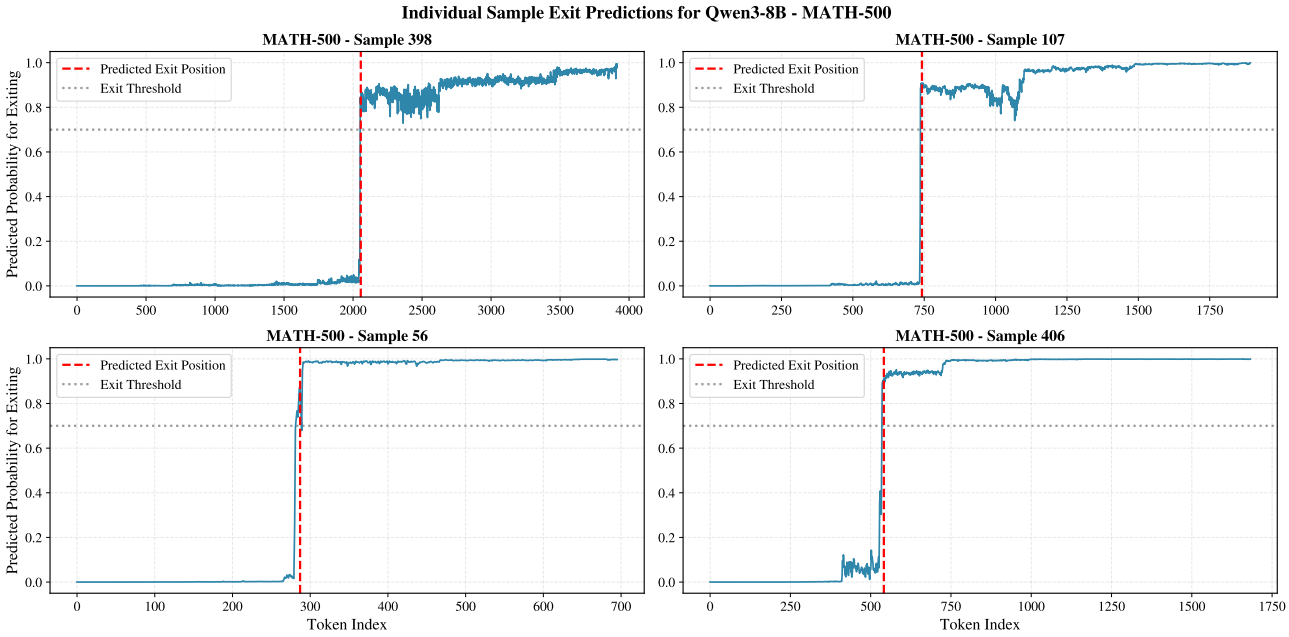

Figure 28. **Predicted Probabilities for MATH-500.** TERMINATOR's predicted probability stream for early-exiting on four randomly chosen samples from MATH-500.

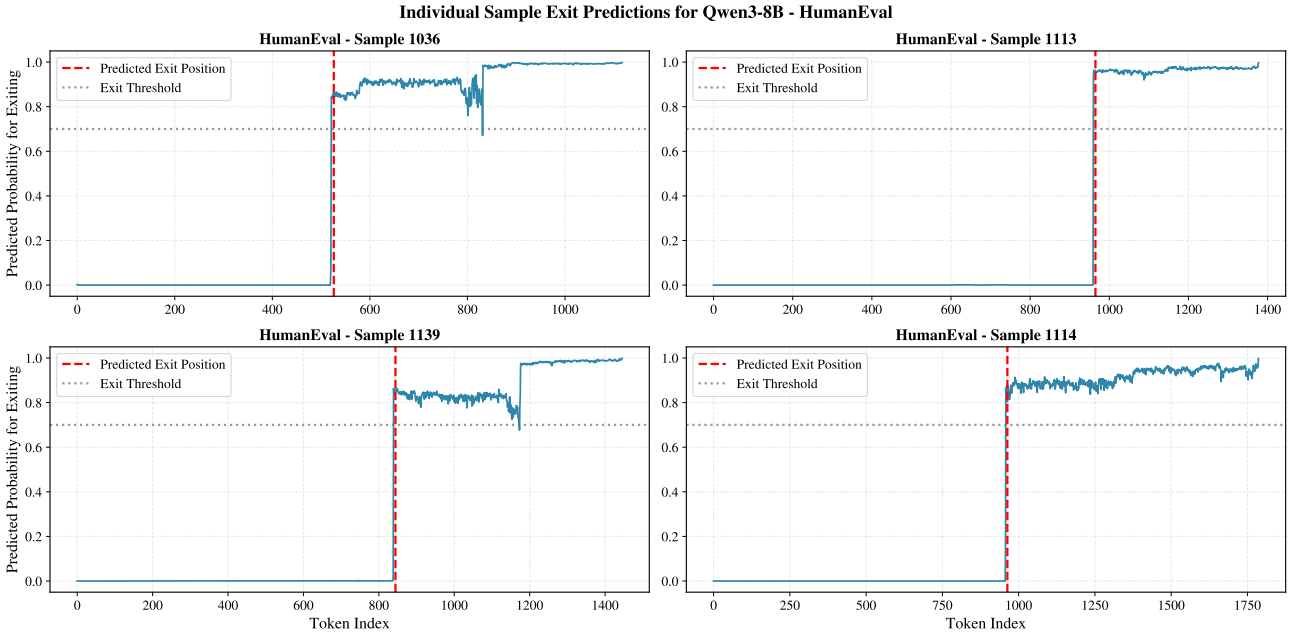

Figure 29. **Predicted Probabilities for HumanEval.** TERMINATOR's predicted probability stream for early-exiting on four randomly chosen samples from HumanEval.

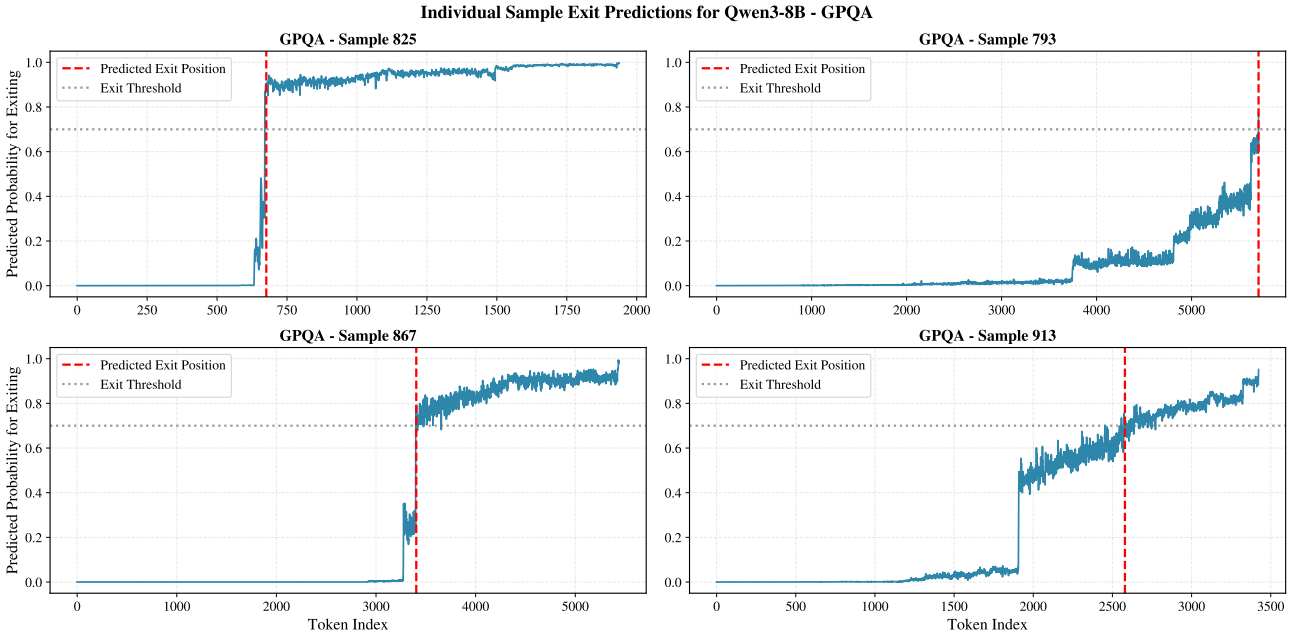

Figure 30. **Predicted Probabilities for GPQA.** TERMINATOR's predicted probability stream for early-exiting on four randomly chosen samples from GPQA.

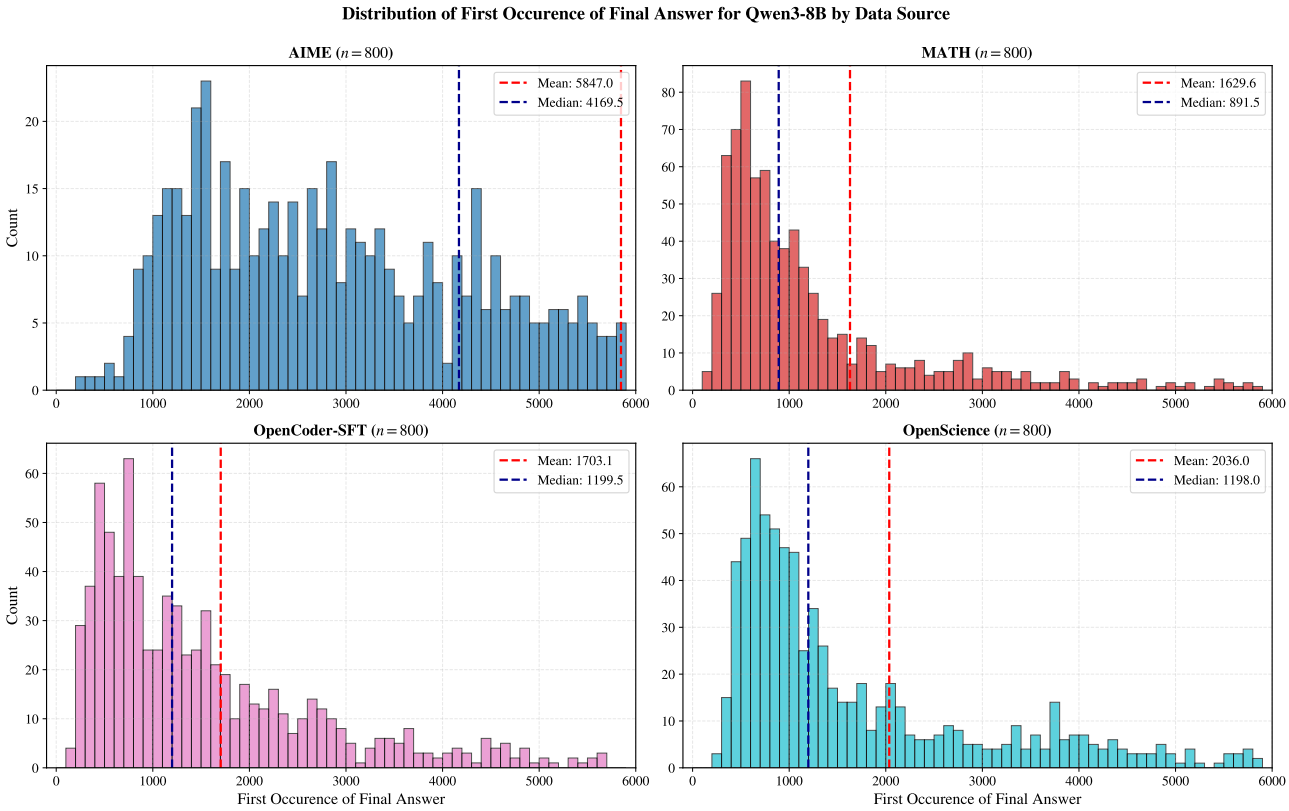

Figure 31. **First Answer Occurrence Histogram.** A histogram of the first occurrence of the final answer for each data source used in our training dataset is shown.

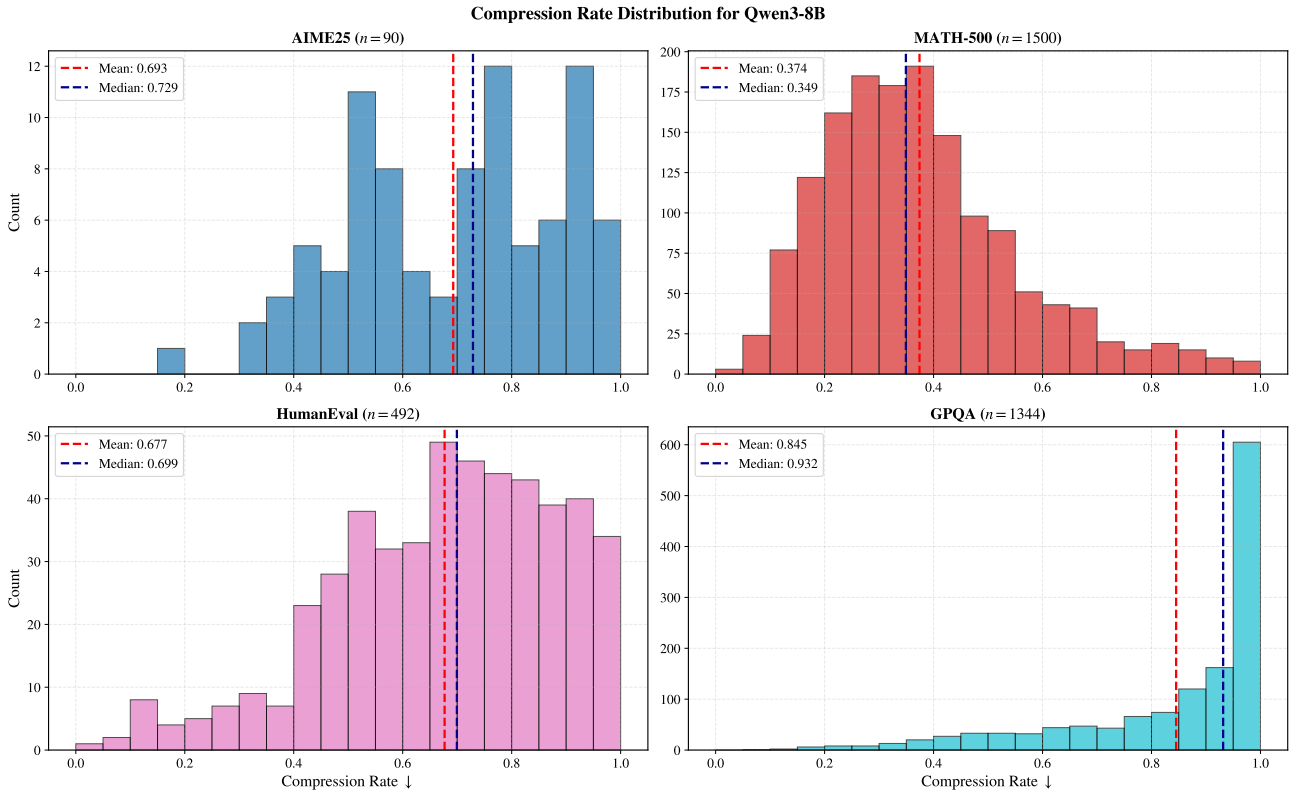

Figure 32. **Compression Ratio Histograms for TERMINATOR.** Each histogram shows the frequency of an achieved compression rate when early-exiting with TERMINATOR. ↓ indicates that a lower compression rate is better, as it results in more tokens saved with our method. Three CoTs are sampled per data source.

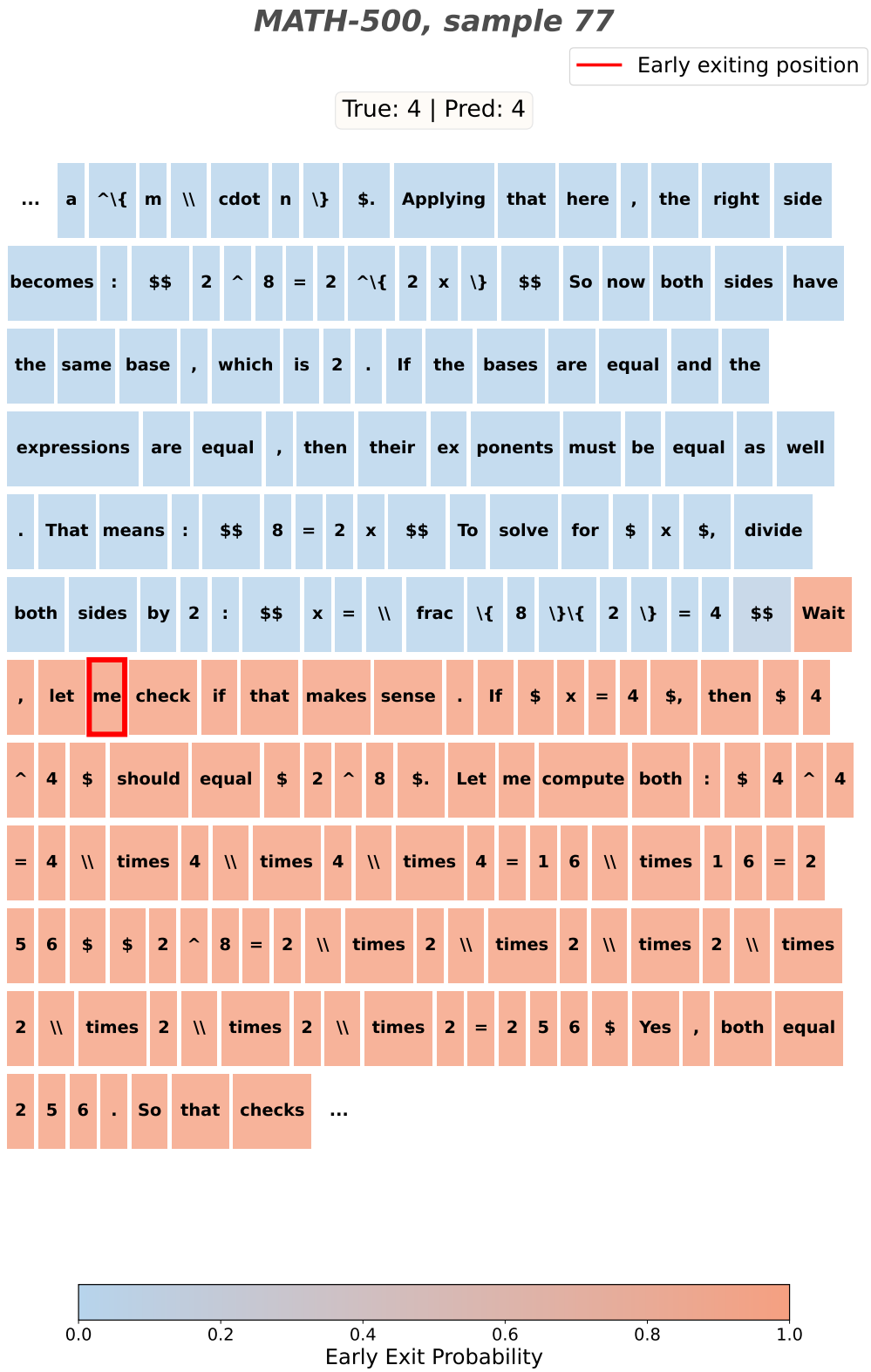

Figure 33. **Predicted Probabilities for MATH-500.** TERMINATOR's predicted probabilities for early-exit on a randomly chosen sample from MATH-500. The beginning and the end are truncated for better visibility.

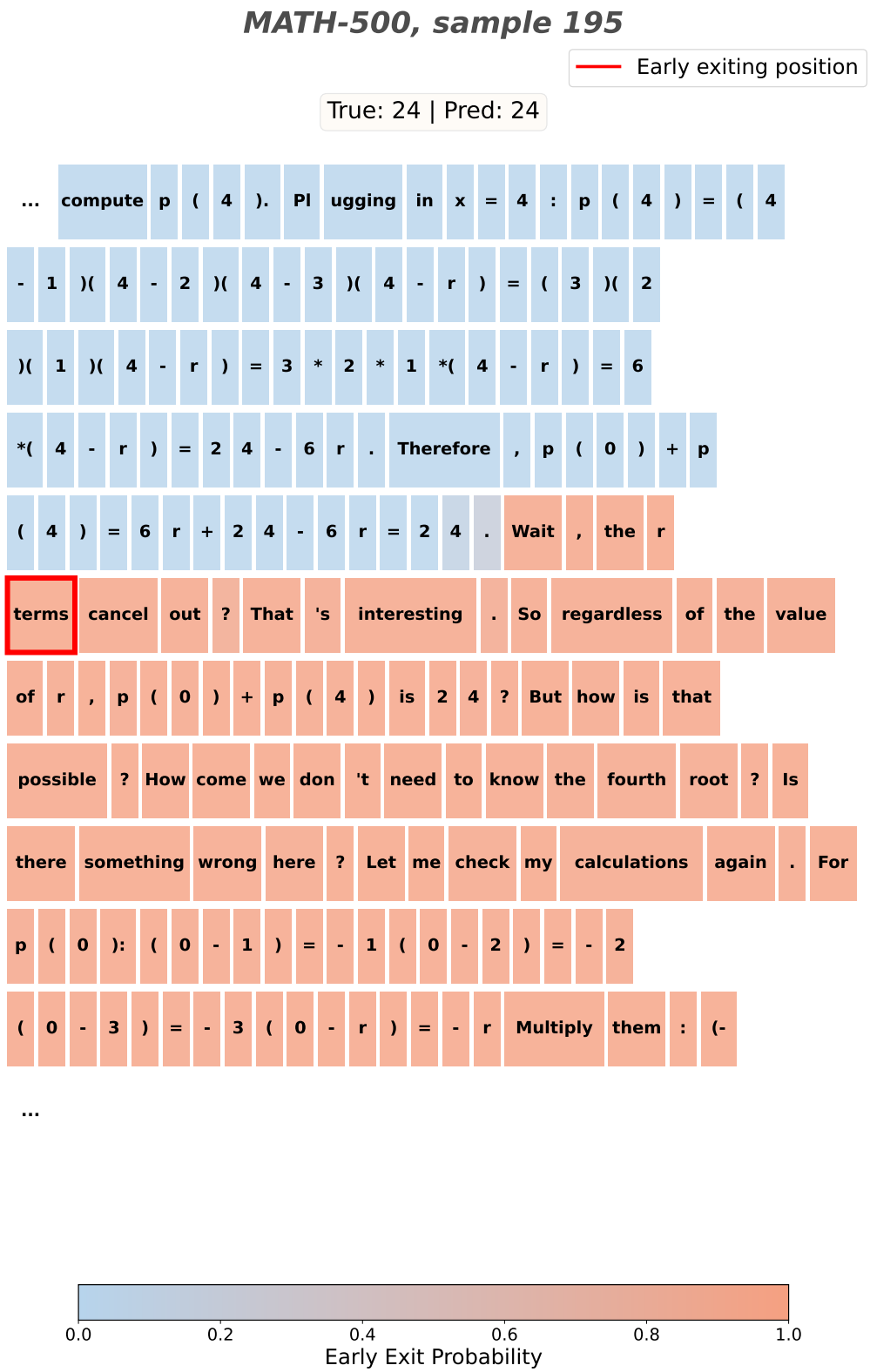

Figure 34. **Predicted Probabilities for MATH-500.** TERMINATOR's predicted probabilities for early-exit on a randomly chosen sample from MATH-500. The beginning and the end are truncated for better visibility.

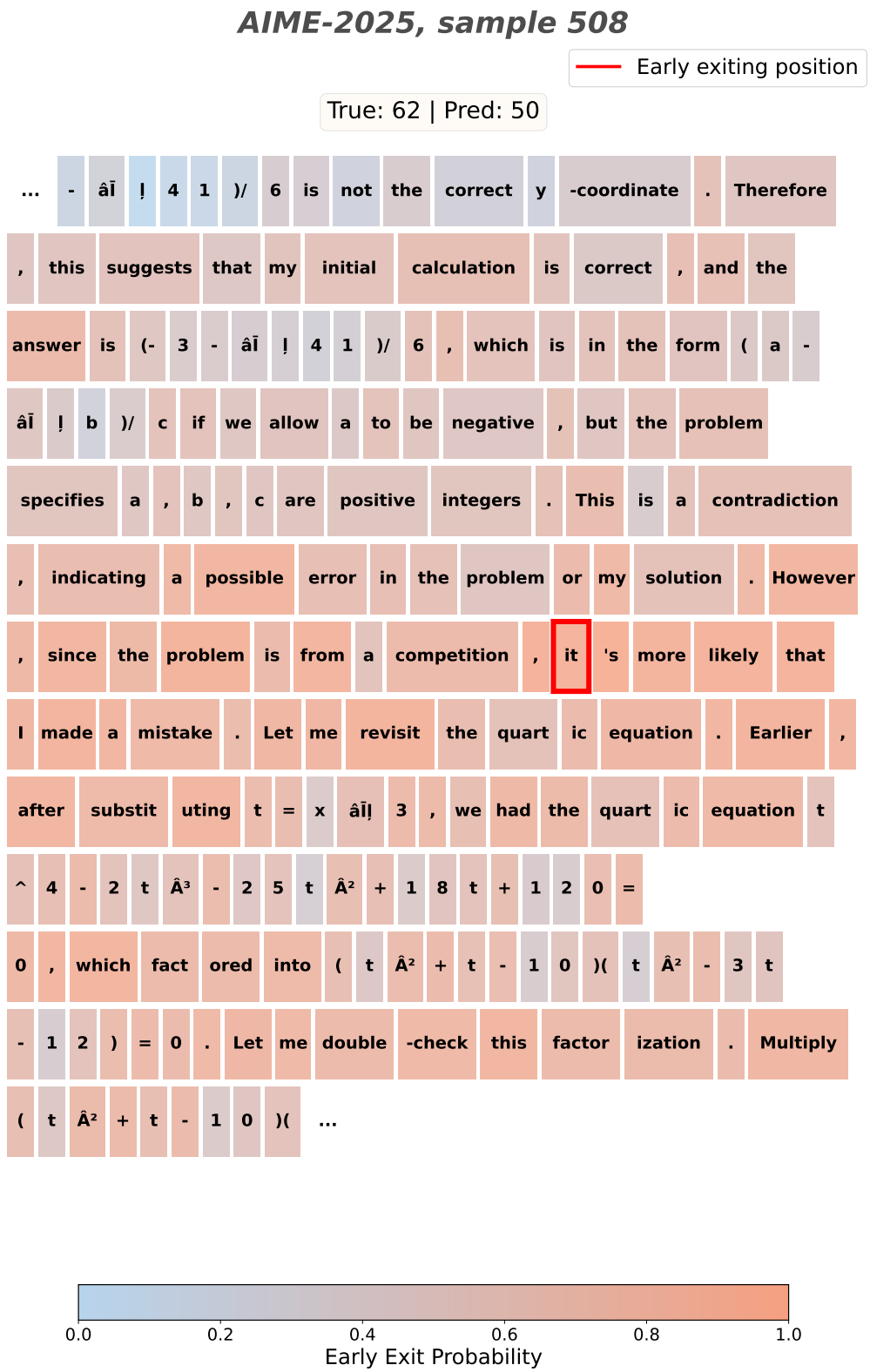

Figure 35. **Predicted Probabilities for AIME25.** TERMINATOR's predicted probabilities for early-exit on a randomly chosen sample from AIME-2025. The beginning and the end are truncated for better visibility.

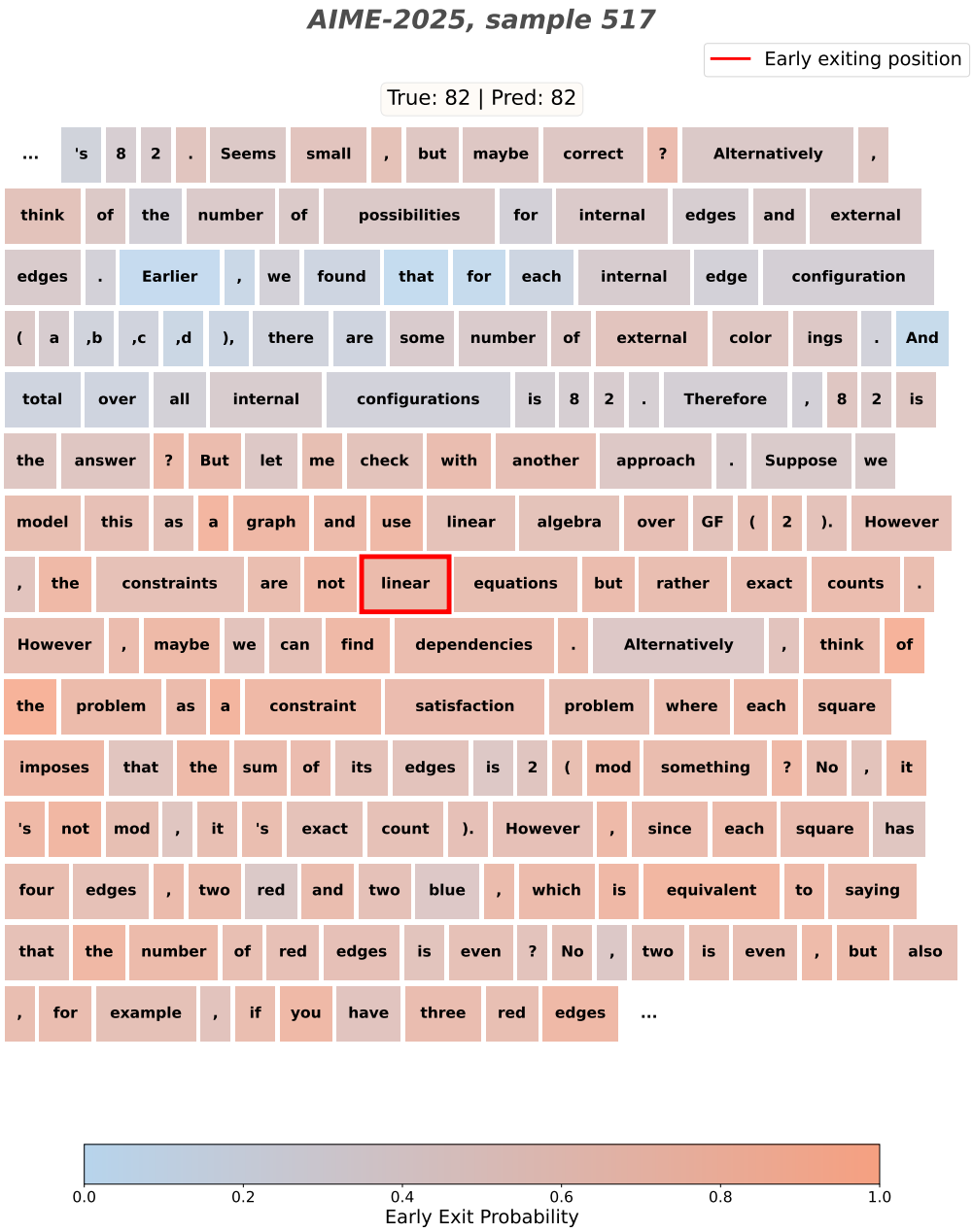

Figure 36. **Predicted Probabilities for AIME25.** TERMINATOR's predicted probabilities for early-exit on a randomly chosen sample from AIME-2025. The beginning and the end are truncated for better visibility.