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

Alliot Nagle* UT Austin
Jakhongir Saydaliev EPFL
Dhia Garbaya ENS Paris-Saclay
Michael Gastpar EPFL

Ashok Vardhan Makkuva† Télécom Paris (IP Paris)
Hyeji Kim† UT Austin

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 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 [35]. 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 [11]. 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* [27, 3]. Prior work has shown that the length of a CoT can be reduced by 50% or more on average with little drop in accuracy [18, 53, 49], 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

*Corresponding Author: `acnagle@utexas.edu`

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

function of reasoning length, gradually increases, peaks, and then decreases, suggesting the existence of an optimal reasoning length [46, 21]. Additionally, some recent works propose novel RL-training algorithms to fine-tune LRMs to produce shorter CoTs [27, 25, 9, 51, 41] and establish the Pareto frontier for those methods, showing that gaps still exist between them [9]. While these works focus on retraining an LRM, inference-time methods such as DEER [49] 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* (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. 4). 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. 4.2), resulting in more than a $2\times$ reduction in inference latency compared to the original LRM (Sec. 5.2).
- 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 (Sec. 4.1).

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. 4). 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 logically reached. Mathematically,

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

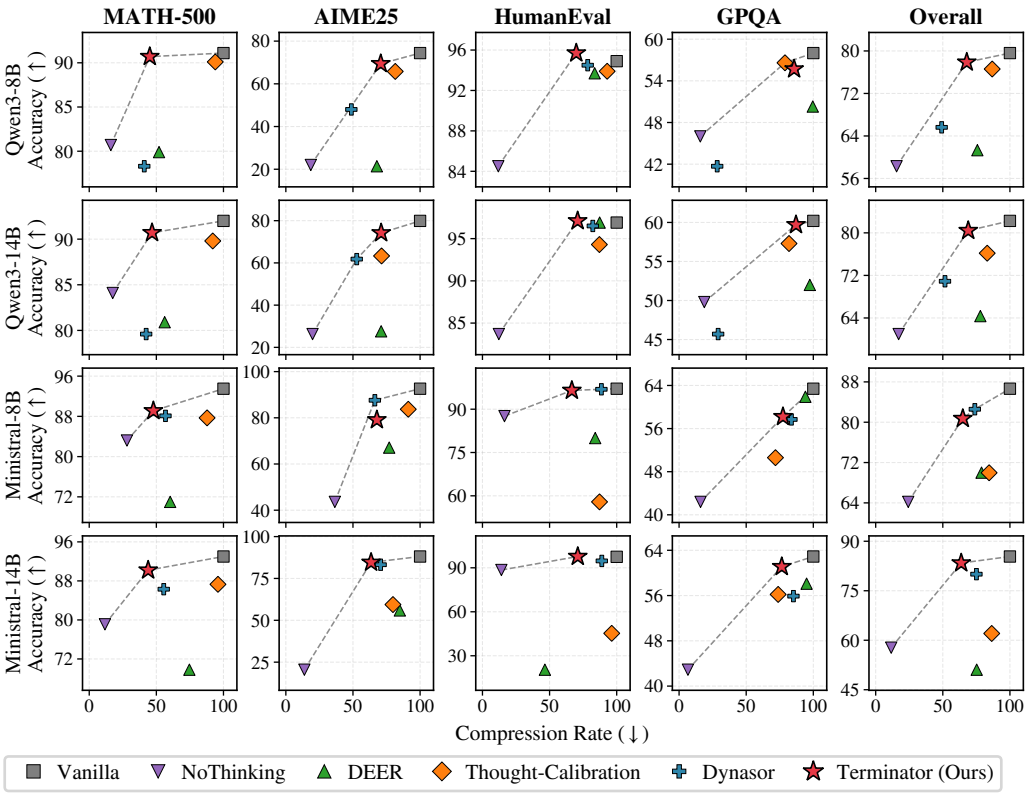


Figure 1: **Pareto Frontier.** TERMINATOR defines the Pareto frontier on 14 of the 16 (LRM, benchmark) pairings, outperforming prior work. Each point represents a method's accuracy and compression rate, with lower compression rates indicating greater token savings and hence compute. The dashed line traces the Pareto frontier connecting non-dominated solutions. We refer to Sec. 5.2 and Table 10 in App. C for more details.

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, which 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 [19], 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 [8] here, as it is principled and produces less noise. Both are used, and the same conclusions can be drawn using either.

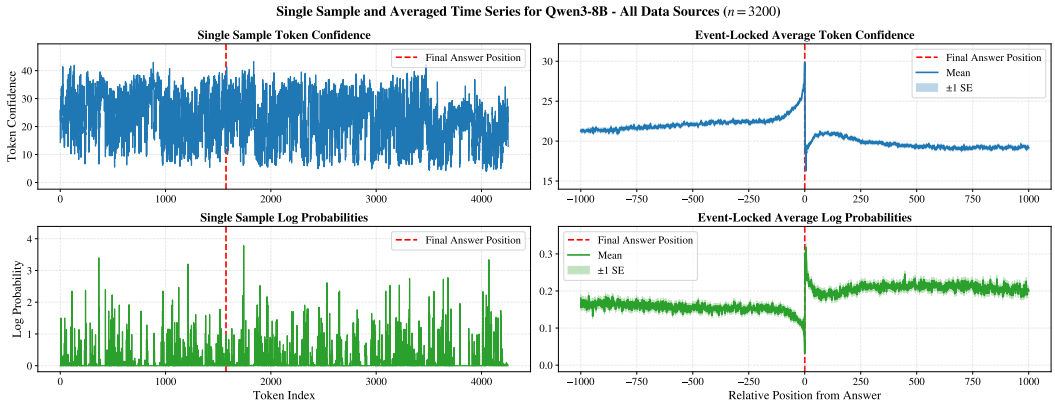


Figure 2: **Event-Locked Averaging of Token-Confidence.** Event-locked averaging shows a consistent agreement on spiking behavior at the answer position in each CoT, but disagrees elsewhere. On the other hand, this phenomenon is not readily observable in the single-sample case. Figures on the **left** show the Token-Confidence [8] and log-probability trajectories throughout reasoning for a single, randomly selected sample; figures on the **right** show 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. C show similar trends for each dataset separately. The standard error (SE) is shown as a shaded region and becomes clearer when zoomed in.

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 [3, 27]. 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 [11]. Many follow-up works have observed the same overthinking phenomenon and developed methods to mitigate wasteful token expenditure [53, 24, 45, 52].

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 usually 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 3. 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.

Fig. 3 complements this 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 [42, 37, 6]. 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,

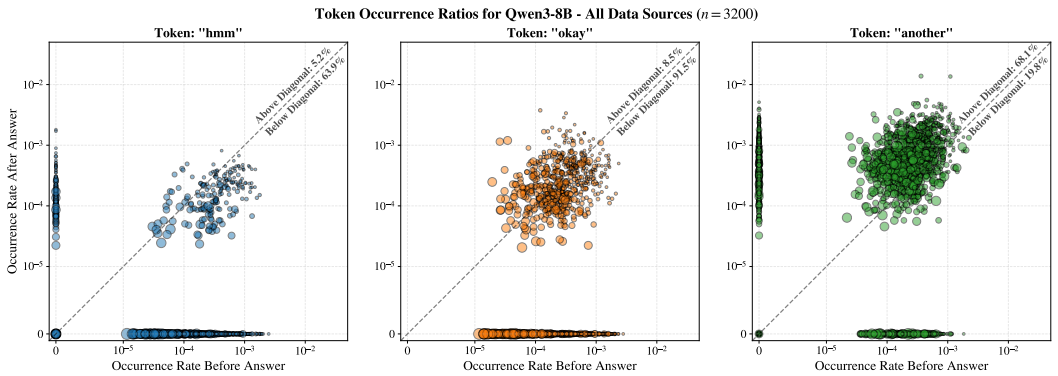


Figure 3: **Token Usage Frequency Shift.** "Thinking token" usage changes depending on whether the final answer has been generated in the CoT. Rates are computed by counting the raw number of occurrences of the token before and after the answer, and then normalizing each count by the respective number 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. C demonstrates similar results for other "thinking tokens" in Fig. 20 and for each data source in Figs. 21 to 24.

computed as raw counts divided by the corresponding number of tokens, and point diameter indicates relative CoT length.

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. 3 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. Under the hood of the LRM, there are clearly meaningful signals to indicate the earliest arrival of $\hat{\boldsymbol{a}}$. However, as outlined above there are unique challenges in leveraging them in a hand-guided way to design an early-exit online inference algorithm. To address this, we approach this through the lens of *prediction*: i.e. predicting whether an LRM's final answer $\hat{\boldsymbol{a}}$ has been generated or not. To this end, the core idea behind our method is to train a probe classifier—the TERMINATOR—on the hidden states of the final layer, thereby utilizing as many of the LRM's underlying signals as possible (Fig. 4). Prior work has examined hidden states to assess whether LRMs know when their intermediate CoT answers are correct [52], a finding primarily aimed at understanding model internals rather than designing practical early-exit methods at inference. Our work adopts a fundamentally different, deployable approach by probing for the *final answer* $\hat{\boldsymbol{a}}$, a signal that is fully self-contained within the LRM's reasoning process and requires no ground-truth labels at inference time, thereby enabling a principled early-exit strategy.

Advantages of TERMINATOR over Prior Methods. To train the probe classifier, we process inputs at the token level, offering much finer-grained predictions than prior work. That is, our dataset is curated for a token classification task. To the best of our knowledge, all previous methods use much coarser granularity in their training dataset where they chunk each CoT $\boldsymbol{r}$ according to some heuristic, such as "thinking tokens" or paragraph delimiters such as `\n\n`. Then, at inference time, they exit once the predicted probability crosses a data-calibrated threshold [24, 45, 52]. In contrast, our approach offers two-fold advantages at inference-time: (1) a probe classifier trained with our dataset has the ability to exit immediately after $\hat{\boldsymbol{a}}$ is generated, and (2) while our approach is amenable to using a data-calibrated threshold, it is not necessary. The main drawback of data-calibrated thresholding is that it requires additional samples from the evaluation data distribution, and the resulting threshold is therefore specific to that distribution and may not transfer well to other distributions.

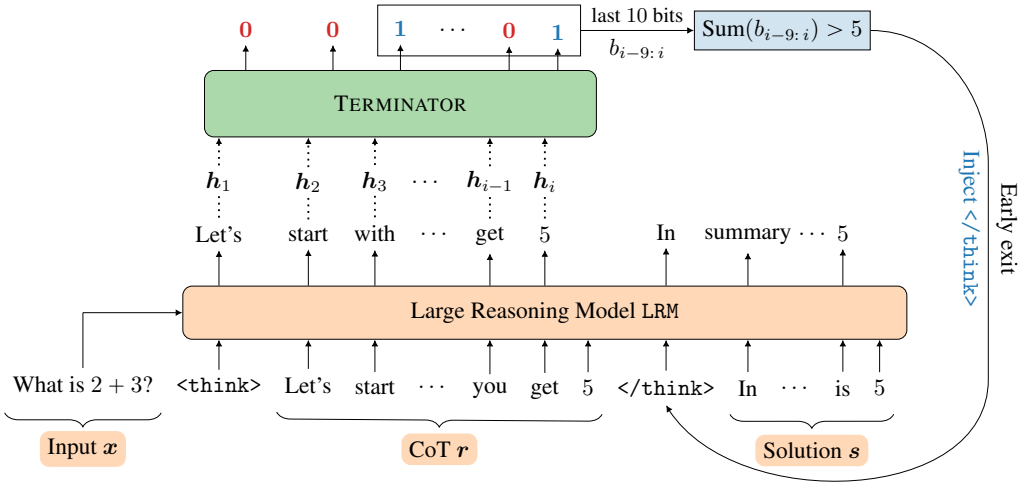


Figure 4: **Early stopping via TERMINATOR.** TERMINATOR is a binary probe classifier that predicts whether to exit or not at every CoT token. Once the majority of prediction bits within a window (10 here) are 1, `</think>` is injected into the LRM's token stream to stop thinking.

However, obtaining the $\hat{a}$ 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, reliable detection for tens of thousands of CoTs is a unique challenge, which we address through our pipeline in Sec. 4.1. We then present our method for training TERMINATOR, which is a probe classifier, in Sec. 4.2.

4.1 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 $\hat{a}$ from final solution s (answer extraction), (2) identifies the earliest logical arrival to $\hat{a}$ in r (answer identification), (3) verifies that the extraction step was successful (answer verification). Finally, (4) we extract the exact position of $\hat{a}$ from the CoT (token-index extraction).

Rationale. Extracting the position of $\hat{a}$ 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 App. B.2. 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 $\hat{a}$ from s, where it is explicitly marked, e.g. with `\boxed{}`, making extraction straightforward for the LRM. Next, the LRM identifies a span d that both precedes and includes $\hat{a}$. This ensures that d is a unique substring of r, allowing us to recover the exact token position of the earliest occurrence of $\hat{a}$. The LRM then verifies that d contains $\hat{a}$. If verification fails, the LRM repeats the identification step, providing textual feedback listing all previously selected spans that did not contain $\hat{a}$, 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 d in r and retrieve the earliest answer token position $i^\star$. Algorithm 1 in App. B.1 gives pseudocode for this procedure. These steps scale the construction of TERMINATOR, which we use to train our probe classifier.

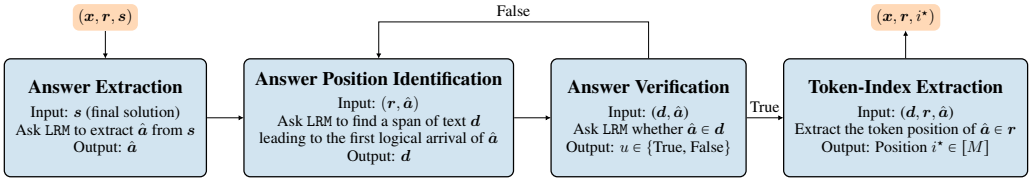


Figure 5: **Training-Dataset Curation Process.** We use an LRM to (1) extract final answer $\hat{\boldsymbol{a}}$ from final solution $\boldsymbol{s}$, (2) identify the earliest position of $\hat{\boldsymbol{a}}$ in the CoT $\boldsymbol{r}$, and (3) verify that the position was correct. If it was, then 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.

Computational Cost. Our cost-benefit analysis in App. C.4 shows that the inference-time benefits of TERMINATOR substantially outweigh the cost of running this pipeline.

4.2 TERMINATOR: 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. 4). 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 $y_i = \mathbb{1}(i \geqslant i^\star) \in \{0, 1\}$ 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.

Here we note that 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

5.1 Implementation Details

Models. We train and evaluate our method on LRMs from two different model families: Qwen3-8B and Qwen3-14B [48], and Ministral-3-8B-Reasoning-2512 and Ministral-3-14B-Reasoning-2512 [23]. 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 [28], DEER [49], and Dynasor [7], and (2) a probe-based approach, Thought Calibration [45]. 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. B.6.

Datasets. We form a training data mix with AIME (1983–2024) [1], MATH [22], OpenCoder-SFT [13], and OpenScience [34]. We form our training datasets by sampling three CoTs from each dataset, identifying the answer positions (see Sec. 4.1), and assigning the corresponding training labels. We evaluate our method and all baselines on AIME 2025 [1], MATH-500 [22], HumanEval [2], and GPQA [39]. Additional details on our training datasets are available in App. B.3.

Training and Inference. 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. Additional details on training TERMINATOR are provided in App. B.4.

We use vLLM [20] 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. Additional details on TERMINATOR's selected inference parameters are provided in App. B.5.

5.2 TERMINATOR: Main Results

Fig. 1 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. 1 shows that TERMINATOR achieves a better overall compression-performance trade-off over prior methods, the same results—presented in table format—in Table 10 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.

5.3 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 3). 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. 1 and Table 10.

Table 1: **Latency Analysis.** Latency and throughput benchmarks on MATH-500 problems for Qwen3-based vanilla and TERMINATOR models. Values are mean $\pm$ 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

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. 6. 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

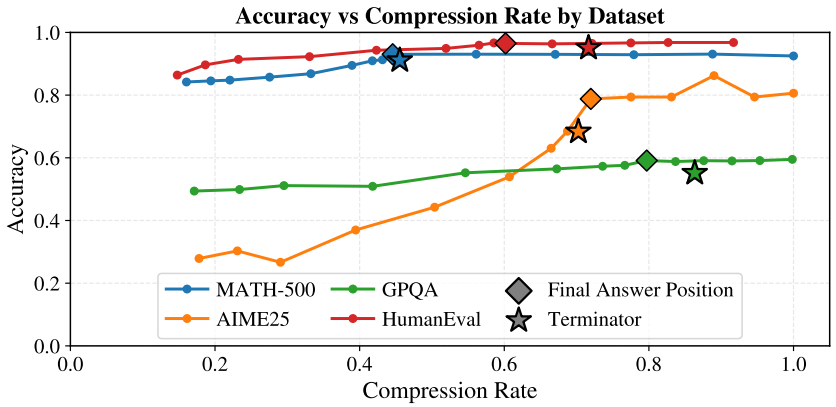


Figure 6: **Effects of Early CoT Termination.** Test set CoTs are evaluated after truncating them at various points via `</think>` and asking the LRM for a final solution and answer. Diamond-shaped markers show the hindsight-optimal answer position (not achievable). TERMINATOR is close to optimality on all datasets.

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. 1 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 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. 3) phenomena. Fig. 10 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. 11 parallels Fig. 3 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. 10 and 11 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* [31, 5, 10, 38] 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* [17, 16, 36, 33], 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 hidden states rather than discrete tokens. Methods

like Coconut [12], CCoT [4], and Soft Thinking [55] 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 [54] uses an idea similar to Auto-Compressor [5], 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 [47, 18], 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 [29] extends this idea by introducing a parameter that adds explicit control over the reasoning length after fine-tuning. CALM [40] 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 [43], DEER [49], ES-CoT [30], and Dynasor [7] are training-free; they track signals throughout the reasoning process and exit when a threshold is crossed. Other methods, like SpecExit [50], Learn To Stop [24], Thought Calibration [45], and FlashThink [15] 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.

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 [26]. 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.

Acknowledgments and Disclosure of Funding

This work was partly supported by ARO Award W911NF2310062, ONR Award N000142412542, NSF 2443857 and the 6G@UT center within the Wireless Networking and Communications Group (WNCG) at the University of Texas at Austin. This research was supported by the Swiss National Science Foundation Grant No. 10004381. Our work used the Vista cluster at the Texas Advanced Computing Center (TACC) at The University of Texas at Austin. We thank the Center for Generative AI and TACC for providing the high-performance computing resources that supported this research.

References

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

[2] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex

Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *CoRR*, abs/2107.03374, 2021. URL https://arxiv.org/abs/2107.03374.

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

[4] Jeffrey Cheng and Benjamin Van Durme. Compressed chain of thought: Efficient reasoning through dense representations, 2024. URL https://arxiv.org/abs/2412.13171.

[5] Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 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/.

[6] Bowen Ding, Yuhan Chen, Futing Wang, Lingfeng Ming, and Tao Lin. Do thinking tokens help or trap? towards more efficient large reasoning model, 2025. URL https://arxiv.org/abs/2506.23840.

[7] Yichao Fu, Junda Chen, Siqi Zhu, Zheyu Fu, Zhongdongming Dai, Yonghao Zhuang, Yian Ma, Aurick Qiao, Tajana Rosing, Ion Stoica, and Hao Zhang. Efficiently scaling llm reasoning with certaindex, 2025. URL https://arxiv.org/abs/2412.20993.

[8] Yichao Fu, Xuewei Wang, Yuandong Tian, and Jiawei Zhao. Deep think with confidence, 2025. URL https://arxiv.org/abs/2508.15260.

[9] Jiaxuan Gao, Shu Yan, Qixin Tan, Lu Yang, Shusheng Xu, Wei Fu, Zhiyu Mei, Kaifeng Lyu, and Yi Wu. How far are we from optimal reasoning efficiency?, 2025. URL https://arxiv.org/abs/2506.07104.

[10] Tao Ge, Hu Jing, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. 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.

[11] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan

Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, September 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z.

[12] Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space, 2025. URL https://arxiv.org/abs/2412.06769.

[13] Siming Huang, Tianhao Cheng, Jason Klein Liu, Jiaran Hao, Liuyihan Song, Yang Xu, J. Yang, J. H. Liu, Chenchen Zhang, Linzheng Chai, Ruifeng Yuan, Zhaoxiang Zhang, Jie Fu, Qian Liu, Ge Zhang, Zili Wang, Yuan Qi, Yinghui Xu, and Wei Chu. Opencoder: The open cookbook for top-tier code large language models. 2024. URL https://arxiv.org/pdf/2411.04905.

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

[15] Guochao Jiang, Guofeng Quan, Zepeng Ding, Ziqin Luo, Dixuan Wang, and Zheng Hu. Flashthink: An early exit method for efficient reasoning, 2025. URL https://arxiv.org/abs/2505.13949.

[16] Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 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.

[17] Hoyoun Jung and Kyung-Joong Kim. 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.

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

[19] Zhewei Kang, Xuandong Zhao, and Dawn Song. Scalable best-of-n selection for large language models via self-certainty, 2025. URL https://arxiv.org/abs/2502.18581.

[20] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

[21] Celine Lee, Alexander M. Rush, and Keyon Vafa. Critical thinking: Which kinds of complexity govern optimal reasoning length?, 2025. URL https://arxiv.org/abs/2504.01935.

[22] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *ICLR*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

[23] Alexander H. Liu, Kartik Khandelwal, Sandeep Subramanian, Victor Jouault, Abhinav Rastogi, Adrien Sadé, Alan Jeffares, Albert Jiang, Alexandre Cahill, Alexandre Gavaudan, Alexandre Sablayrolles, Amélie Héliou, Amos You, Andy Ehrenberg, Andy Lo, Anton Eliseev, Antonia Calvi, Avinash Sooriyarachchi, Baptiste Bout, Baptiste Rozière, Baudouin De Monicault, Clémence Lanfranchi, Corentin Barreau, Cyprien Courtot, Daniele Grattarola, Darius Dabert, Diego de las Casas, Elliot Chane-Sane, Faruk Ahmed, Gabrielle Berrada, Gaëtan Ecrepont, Gauthier Guinet, Georgii Novikov, Guillaume Kunsch, Guillaume Lample, Guillaume Martin,

Gunshi Gupta, Jan Ludziejewski, Jason Rute, Joachim Studnia, Jonas Amar, Joséphine Delas, Josselin Somerville Roberts, Karmesh Yadav, Khyathi Chandu, Kush Jain, Laurence Aitchison, Laurent Fainsin, Léonard Blier, Lingxiao Zhao, Louis Martin, Lucile Saulnier, Luyu Gao, Maarten Buyl, Margaret Jennings, Marie Pellat, Mark Prins, Mathieu Poirée, Mathilde Guillaumin, Matthieu Dinot, Matthieu Futeral, Maxime Darrin, Maximilian Augustin, Mia Chiquier, Michel Schimpf, Nathan Grinsztajn, Neha Gupta, Nikhil Raghuraman, Olivier Bousquet, Olivier Duchenne, Patricia Wang, Patrick von Platen, Paul Jacob, Paul Wambergue, Paula Kurylowicz, Pavankumar Reddy Muddireddy, Philomène Chagniot, Pierre Stock, Pravesh Agrawal, Quentin Torroba, Romain Sauvestre, Roman Soletskyi, Rupert Menneer, Sagar Vaze, Samuel Barry, Sanchit Gandhi, Siddhant Waghjale, Siddharth Gandhi, Soham Ghosh, Srijan Mishra, Sumukh Aithal, Szymon Antoniak, Teven Le Scao, Théo Cachet, Theo Simon Sorg, Thibaut Lavril, Thiziri Nait Saada, Thomas Chabal, Thomas Foubert, Thomas Robert, Thomas Wang, Tim Lawson, Tom Bewley, Tom Bewley, Tom Edwards, Umar Jamil, Umberto Tomasini, Valeriia Nemychnikova, Van Phung, Vincent Maladière, Virgile Richard, Wassim Bouaziz, Wen-Ding Li, William Marshall, Xinghui Li, Xinyu Yang, Yassine El Ouahidi, Yihan Wang, Yunhao Tang, and Zaccharie Ramzi. Ministral 3, 2026. URL https://arxiv.org/abs/2601.08584.

[24] Xin Liu and Lu Wang. Answer convergence as a signal for early stopping in reasoning, 2025. URL https://arxiv.org/abs/2506.02536.

[25] Chenwei Lou, Zewei Sun, Xinnian Liang, Meng Qu, Wei Shen, Wenqi Wang, Yuntao Li, Qingping Yang, and Shuangzhi Wu. Adacot: Pareto-optimal adaptive chain-of-thought triggering via reinforcement learning, 2025. URL https://arxiv.org/abs/2505.11896.

[26] Steven J. Luck. *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.

[27] Haotian Luo, Li Shen, Haiying He, Yibo Wang, Shiwei Liu, Wei Li, Naiqiang Tan, Xiaochun Cao, and Dacheng Tao. O1-pruner: Length-harmonizing fine-tuning for o1-like reasoning pruning, 2025. URL https://arxiv.org/abs/2501.12570.

[28] Wenjie Ma, Jingxuan He, Charlie Snell, Tyler Griggs, Sewon Min, and Matei Zaharia. Reasoning models can be effective without thinking, 2025. URL https://arxiv.org/abs/2504.09858.

[29] Xinyin Ma, Guangnian Wan, Runpeng Yu, Gongfan Fang, and Xinchao Wang. CoT-valve: Length-compressible chain-of-thought tuning. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6025–6035, Vienna, Austria, July 2025. 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/.

[30] Minjia Mao, Bowen Yin, Yu Zhu, and Xiao Fang. Early stopping chain-of-thoughts in large language models, 2025. URL https://arxiv.org/abs/2509.14004.

[31] Jesse Mu, Xiang Lisa Li, and Noah Goodman. Learning to compress prompts with gist tokens. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=2DtxPCL3T5.

[32] Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling, 2025. URL https://arxiv.org/abs/2501.19393.

[33] Alliot Nagle, Adway Girish, Marco Bondaschi, Michael Gastpar, Ashok Vardhan Makkuva, and Hyeji Kim. Fundamental limits of prompt compression: A rate-distortion framework for black-box language models. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 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.

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

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

[36] Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Ruhle, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Dongmei Zhang. LLMLingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics ACL 2024*, pages 963–981, Bangkok, Thailand and virtual meeting, August 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.findings-acl.57.

[37] Chen Qian, Dongrui Liu, Haochen Wen, Zhen Bai, Yong Liu, and Jing Shao. Demystifying reasoning dynamics with mutual information: Thinking tokens are information peaks in llm reasoning. *arXiv preprint arXiv:2506.02867*, 2025.

[38] Guanghui Qin, Corby Rosset, Ethan Chau, Nikhil Rao, and Benjamin Van Durme. Dodo: Dynamic contextual compression for decoder-only LMs. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 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/.

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

[40] Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 17456–17472. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/6fac9e316a4ae75ea244ddcef1982c71-Paper-Conference.pdf.

[41] Vaishnavi Shrivastava, Ahmed Awadallah, Vidhisha Balachandran, Shivam Garg, Harkirat Behl, and Dimitris Papailiopoulos. Sample more to think less: Group filtered policy optimization for concise reasoning, 2025. URL https://arxiv.org/abs/2508.09726.

[42] Chenlong Wang, Yuanning Feng, Dongping Chen, Zhaoyang Chu, Ranjay Krishna, and Tianyi Zhou. Wait, we don't need to "wait"! removing thinking tokens improves reasoning efficiency. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, editors, *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 7459–7482, Suzhou, China, November 2025. 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/.

[43] Xi Wang, James McInerney, Lequn Wang, and Nathan Kallus. Entropy after ⟨/Think⟩ for reasoning model early exiting, 2025. URL https://arxiv.org/abs/2509.26522.

[44] Chao Wu, Baoheng Li, Mingchen Gao, Yu Tian, and Zhenyi Wang. From efficiency to adaptivity: A deeper look at adaptive reasoning in large language models, 2026. URL https://arxiv.org/abs/2511.10788.

[45] Menghua Wu, Cai Zhou, Stephen Bates, and Tommi Jaakkola. Thought calibration: Efficient and confident test-time scaling. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, editors, *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 14302–14316, Suzhou, China, November 2025. 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/.

[46] Yuyang Wu, Yifei Wang, Tianqi Du, Stefanie Jegelka, and Yisen Wang. When more is less: Understanding chain-of-thought length in LLMs. In *Workshop on Reasoning and Planning for Large Language Models*, 2025. URL https://openreview.net/forum?id=W8dxn7hBk0.

[47] Heming Xia, Chak Tou Leong, Wenjie Wang, Yongqi Li, and Wenjie Li. TokenSkip: Controllable chain-of-thought compression in LLMs. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, editors, *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 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/.

[48] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

[49] Chenxu Yang, Qingyi Si, Yongjie Duan, Zheliang Zhu, Chenyu Zhu, Qiaowei Li, Minghui Chen, Zheng Lin, and Weiping Wang. Dynamic early exit in reasoning models, 2025. URL https://arxiv.org/abs/2504.15895.

[50] Rubing Yang, Huajun Bai, Song Liu, Guanghua Yu, Runzhi Fan, Yanbin Dang, Jiejing Zhang, Kai Liu, Jianchen Zhu, and Peng Chen. Specexit: Accelerating large reasoning model via speculative exit, 2025. URL https://arxiv.org/abs/2509.24248.

[51] Jingyang Yi, Justin Wang, and Sida Li. 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.

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

[53] Jiajie Zhang, Nianyi Lin, Lei Hou, Ling Feng, and Juanzi Li. AdaptThink: Reasoning models can learn when to think. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, editors, *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 3716–3730, Suzhou, China, November 2025. 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/.

[54] Jintian Zhang, Yuqi Zhu, Mengshu Sun, Yujie Luo, Shuofei Qiao, Lun Du, Da Zheng, Huajun Chen, and Ningyu Zhang. LightThinker: Thinking step-by-step compression. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, editors, *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 13307–13328, Suzhou, China, November 2025. 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/.

[55] Zhen Zhang, Xuehai He, Weixiang Yan, Ao Shen, Chenyang Zhao, Shuohang Wang, Yelong Shen, and Xin Eric Wang. Soft thinking: Unlocking the reasoning potential of llms in continuous concept space, 2025. URL https://arxiv.org/abs/2505.15778.

A Limitations and Broader Impacts

A.1 Limitations

Our approach relies on the assumption that exiting at the earliest occurrence of the final answer is sufficient for the LRM to use that same answer in the final solution. Although rare, the LRM does not always commit to the exited answer in the CoT, which can cause the exited response to be marked incorrect even when the vanilla LRM, using full reasoning, eventually obtains the correct answer. Furthermore, the practicality of our approach depends on the final answer being present within the CoT, which might not always be the case, especially for open-ended generation tasks.

Each base model also requires its own probe. TERMINATOR is initialized as an exact copy of the final layer of the LRM it attaches to, including its architecture and weights, and hidden-state geometry differs substantially across model families, so we do not expect probes to transfer across families without fine-tuning. This is a light requirement in practice, since TERMINATOR is a one-layer transformer with a linear head, and cross-family transfer is an interesting direction for follow-up work. Finally, our evaluation covers 8B and 14B models, which enable controlled comparisons across families at a fixed scale and across scales within a family. Prior work has not found early-exit effectiveness to differ meaningfully between small- and large-scale LRMs [49, 45, 7].

A.2 Broader Impacts

This work aims to advance machine learning research by improving the efficiency of reasoning in large language models through early-exit mechanisms based on model confidence. Potential benefits include reduced energy consumption, faster inference, and lower deployment costs, thereby improving the scalability and environmental sustainability of language model systems.

At the same time, confidence-based early termination introduces the risk of over-confidence and premature conclusions, particularly in settings where incorrect outputs may have significant consequences. These risks underscore the importance of careful calibration, evaluation, and responsible deployment practices. Our work does not introduce new classes of societal harm beyond those already associated with large language models, but it highlights the need for continued research into reliable confidence estimation and safe adaptive inference.

B Additional Details on Our Methods

B.1 Early Answer Extraction, Identification, and Verification

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 at or 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.

We label every position at or after $i^\star$ as 1 rather than supervising only near the stopping boundary. Supervising only near the stopping boundary would make the training label a one-hot vector over a sequence of up to 32k tokens, so a trivial all-zero predictor achieves near-perfect accuracy and the probe fails to learn any exit point. Our formulation instead asks "Has the final answer arrived?" rather than "Is this the exact stopping point?", which keeps the label dense enough to learn from and lets TERMINATOR still find a valid exit even when the earliest arrival has already passed.

B.2 Failure Modes of Pattern Matching for Answer Position Identification

The following are examples of failure modes for pattern matching while extracting the earliest arrival of $\hat{\boldsymbol{a}}$ from an CoT, thus motivating our LRM-based extraction pipeline:

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

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
$\hat{\boldsymbol{a}} \leftarrow \texttt{LRM}(\text{“Extract the final answer from: ”} + \boldsymbol{s})$

// 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 $\boldsymbol{r}$
  $\boldsymbol{d} \leftarrow \texttt{LRM}(\text{“Find first occurrence of ”} + \hat{\boldsymbol{a}} + \text{“ in: ”} + \boldsymbol{r} + \boldsymbol{z})$

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

  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
$c \leftarrow \texttt{FuzzyMatch}(\boldsymbol{d}, \boldsymbol{r})$ *// c is an integer-based character index of $\boldsymbol{d}$ in $\boldsymbol{r}$*

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

return $i^\star$

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. For instance, $\texttt{x}^2$, `x**2`, `pow(x,2)`, and $\texttt{x} \cdot \texttt{x}$ 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.

B.3 Training Dataset Details

Our training dataset consists of CoTs from AIME (1983–2024) [1], MATH [22], OpenCoder-SFT [13], and OpenScience [34]. 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 [48].

Table 2: **Pipeline Retention by Data Source.** Number of Qwen3-8B CoTs processed by our answer extraction, identification, and verification pipeline, the percentage retained for training, and the median length (in generated tokens) of retained and dropped CoTs.

Source	CoTs	Retained	Median Len. Retained	Median Len. Dropped
MATH	35,693	77.9%	3,398	5,427
OpenScience	35,996	83.5%	2,494	6,274
OpenCoder-SFT	35,958	63.9%	3,015	5,190
AIME (1983–2024)	2,698	64.3%	8,817	13,834
All	**110,345**	**74.8%**	**3,037**	**5,691**

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 Sec. 4.1. 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.

Retained and Dropped CoTs. Table 2 breaks down pipeline retention by data source. Dropped CoTs are longer than retained ones in every source: $1.6\times$ on MATH and AIME (1983–2024), $1.7\times$ on OpenCoder-SFT, and $2.5\times$ on OpenScience. Retention also varies by source, from 83.5% (OpenScience) down to 63.9% (OpenCoder-SFT) and 64.3% (AIME). We attribute most of this to a single cause: the pipeline drops a CoT when it cannot localize a unique earliest arrival, and longer CoTs contain more restatements, abandoned branches, and verification passes. AIME CoTs are the longest and among the least retained, consistent with this account. OpenCoder-SFT is an exception that we attribute to the answer format, since a code answer is rarely a short extractable span, and so a first occurrence is often ill-posed. Filtering applies only to label construction; evaluation uses the full unfiltered benchmarks, so no reported numbers benefit from it.

Label Quality. We offer two forms of validation that the pipeline's exit positions are accurate. First, Fig. 6 shows, for each benchmark, where the chosen exit position falls on the accuracy-compression trade-off (diamond markers), alongside accuracy from truncating CoTs before and after that point. The curve is flat after the exit position, so additional reasoning past it yields no accuracy benefit. If the pipeline were latching onto superficial early answers rather than the first logical arrival, consistent accuracy gains would remain available past the labeled positions.

Second, we manually inspected 30 randomly sampled CoTs from our two hardest, longest-context benchmarks, AIME25 ($n = 20$) and GPQA ($n = 10$), checking whether the exit position was preceded by a derivation showing how the final answer was reached. In 29 of 30 cases, it was; the single failure was a false positive, disjoint from any earliest logical arrival. We note that precise false-positive and false-negative rates are challenging to quantify, since these CoTs are lengthy and often contain several sections where the final answer is deliberated without concrete steps constituting a logical arrival.

B.4 TERMINATOR Training Details

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. 7 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

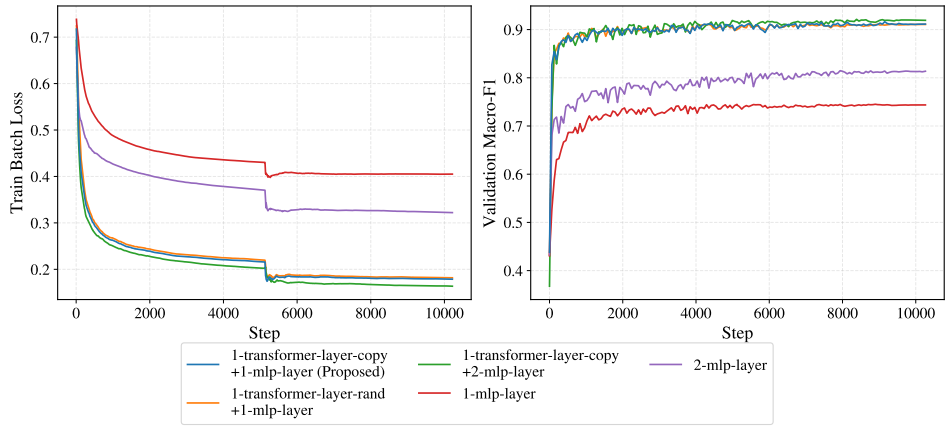


Figure 7: **Training Convergence.** During training, our proposed TERMINATOR architecture achieves the best balance between the highest Macro-F1 validation scores and the lowest compute overhead.

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:

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.

B.5 TERMINATOR Inference Details

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. 8 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 Figs. 26 to 30, once the predicted exit probability exceeds the threshold, it rarely drops below, so the window size has minimal impact on the majority vote.

Probability-Based Aggregation. Figs. 27 to 30 plot the predicted exit probability over time for randomly sampled problems. The signal is step-like, staying low then rising sharply and plateauing at the exit position; where it is not a sharp step (some GPQA problems), it remains close to monotonic, and once it crosses the threshold, it rarely falls back. Averaging or EMA smoothing over a near-monotonic step function therefore adds no signal over thresholding directly; its main effect is to delay the crossing point, which is already reachable by adjusting our window size and threshold. We did not pursue calibrated sequential rules because they require held-out data representative of the evaluation distribution, which is not always available, and separate calibration parameters per benchmark, which assumes the problem domain is known at inference time.

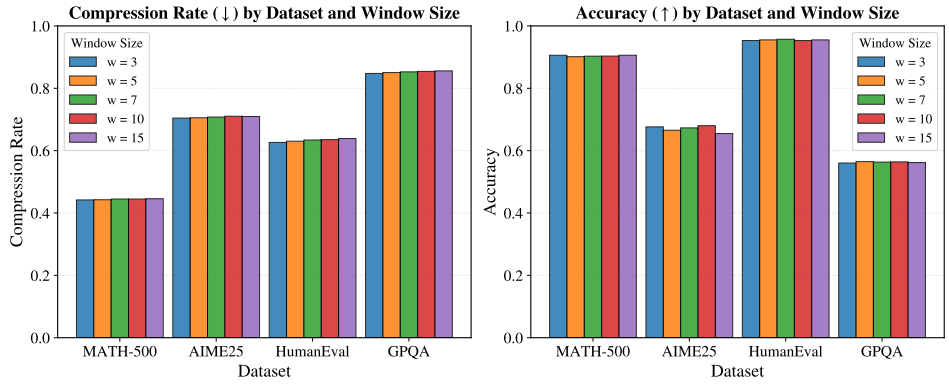


Figure 8: **Effect of Varying Window Size.** We fix the threshold to 0.7 and vary the size of the window used by TERMINATOR at inference-time. TERMINATOR is robust to changes in the window size; Figs. 26 to 30 show that the predicted probability of TERMINATOR during CoT generation tends to only increase. 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.

Thresholding. Fig. 9 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 for identifying viable early-exit points: lowering the threshold makes the method more aggressive, substantially improving compression while only modestly affecting accuracy.

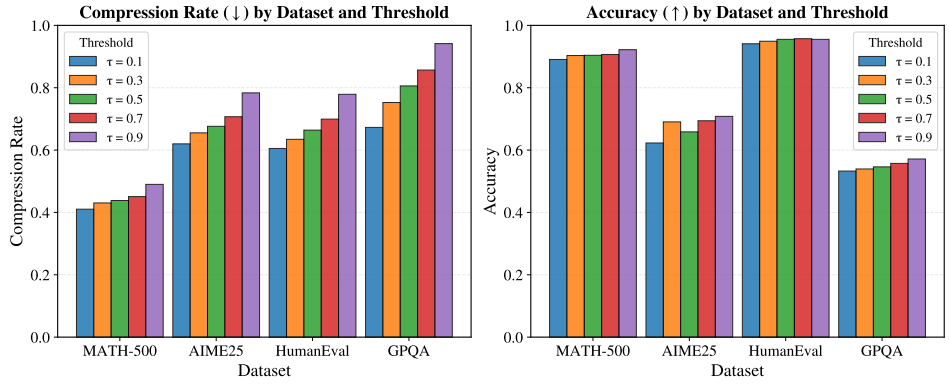


Figure 9: **Effect of Varying Predictive Threshold.** We fix the window size to 10 and vary the threshold τ used by TERMINATOR to predict whether to exit early. TERMINATOR is robust to changes in the predictive threshold: lowering τ makes the method more aggressive, leading to substantially higher compression 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.

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.

B.6 Implementation of the Baseline Methods

Dynasor. Dynasor [7] 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. [45] 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 [32] for all four models in our experiments: Qwen3-8B, Qwen3-14B, Ministral-3-8B-Reasoning-2512, and Ministral-3-14B-Reasoning-2512. For inference, we use the hyperparameters specified in Table 3. As shown in Table 3, 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 [49].

Table 3: Hyperparameters used for *Supervised* and *Consistent* linear probes for Thought Calibration. Ministral models' full names are Ministral-3-xxB-Reasoning-2512. Tolerance and threshold refer to δ and λ, respectively.

	Supervised			**Consistent**		
Model	tolerance	threshold	window size	tolerance	threshold	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	0.10	0.8173	10	0.025	0.9973	10
Ministral-3-14B	0.10	0.8306	10	0.025	0.9973	10

We reported the results for the *Supervised* probe in Table 10, as it performed better across test datasets than the *Consistent* probe. For comparison, we report the results for both probes in Table 4. Note that for the *Consistent* probe with Ministral models, tolerance 0.025 represents the smallest feasible setting among values suggested by Wu et al. [45]. The smallest possible setting, being tolerance 0.01, yields threshold=1.0, resulting in no compression.

Table 4: Performance comparison of *Supervised* and *Consistent* probes for Thought Calibration across models and tasks. Ministral models' full names are Ministral-3-xxB-Reasoning-2512.

	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	87.7%	87.80%	83.7%	91.16%	47.3%	71.78%	47.2%	87.16%
Ministral-3-14B	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	75.8%	80.53%	23.3%	60.50%	42.4%	62.69%	34.2%	75.24%
Ministral-3-14B	68.7%	77.40%	9.4%	34.42%	40.6%	59.97%	9.6%	95.90%

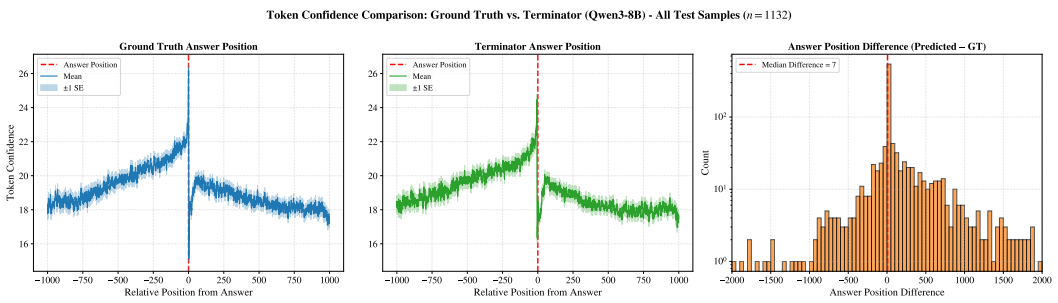


Figure 10: **TERMINATOR Recovers Event-Locked Average Spiking.** The exit positions predicted by TERMINATOR **(center)** recovers similar spiking behavior in the event-locked averaged Token-Confidence as the ground-truth answer positions **(left)**. The histogram of differences between the exit positions **(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.

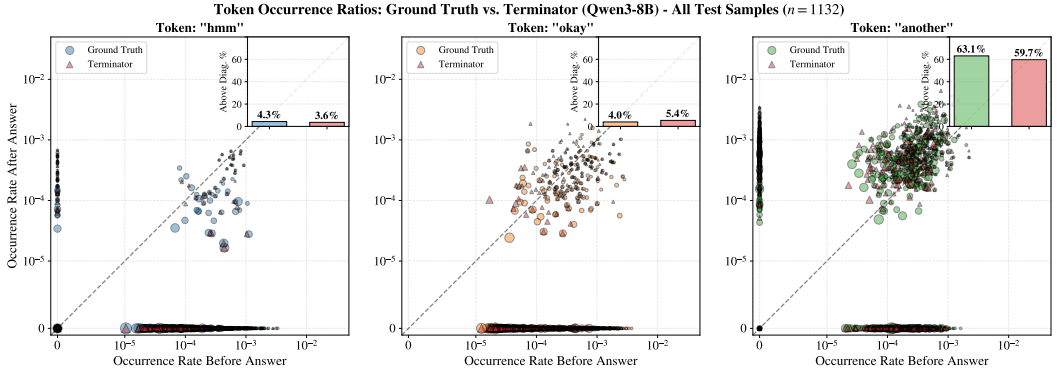


Figure 11: **TERMINATOR Token Usage Biases.** The exit positions predicted by TERMINATOR recover the same biases in the "thinking token" occurrence rates as the ground-truth answer positions. The inset axes on each panel show the percentage of dots that lie above the diagonal when the ground-truth and TERMINATOR answer positions are used.

C Additional Experimental Results

C.1 Early-Exit Signal Analysis

TERMINATOR Recovers Early-Exit Signals. As discussed in Sec. 5.3, 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. 10 and 11 establish this claim. Discussion on our findings are provided in Sec. 5.3

Event-Locked Averaging Token-Confidence. The results shown in Figs. 2 and 3 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 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.

Token Usage Frequency Shift. Beyond providing motivation for our method, the results of Fig. 3 offer some interesting insights on the usage of "thinking tokens." These plots show the strong usage

Table 5: **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 10). 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 10)	**91%**	**45%**	**69%**	71%	**96%**	70%	**56%**	86%	**78%**	68%

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. 3 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. 3, 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.

C.2 Adaptive Strategies for TERMINATOR

In principle, adaptive early-exiting strategies offer a promising way to tailor inference to the target domain [44]. 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 on these two design choices.

Table 5 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 10. 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-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 $5^4 = 625$

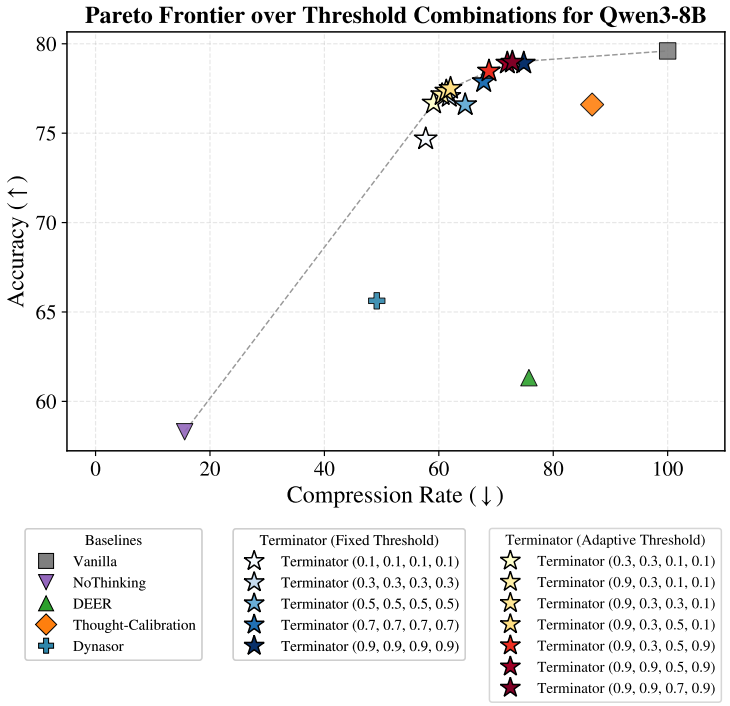


Figure 12: **Adaptive Thresholding for TERMINATOR.** We plot TERMINATOR at various thresholds (blue stars, fixed for all domains) and compare with a domain-adaptive threshold selection (red stars). The tuples correspond to the thresholds chosen for (MATH-500, AIME 25, HumanEval, GPQA) accordingly. For 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.

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.

C.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, MATH-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.

Exit Timing Relative to Answer Arrival. On harder problems, reasoning after the first arrival of the answer may be needed for self-correction rather than being wasteful. This failure mode is absent from our training labels by construction: the hindsight-optimal length is the earliest arrival

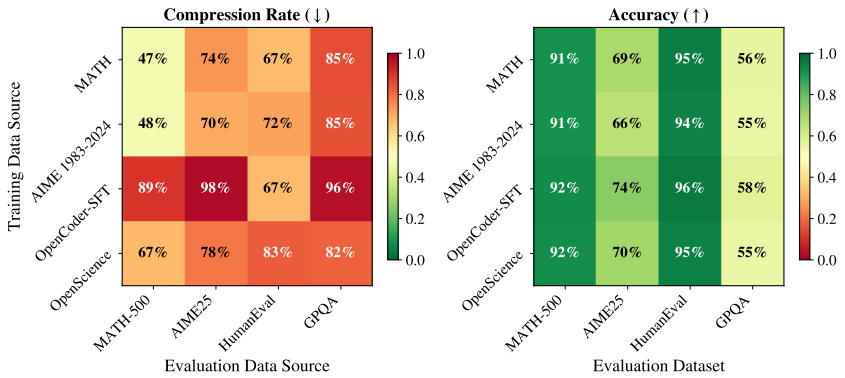


Figure 13: **OOD Performance of TERMINATOR.** The best trade-off between accuracy and compression rate is achieved when the evaluation set is in-distribution with the training dataset. Here the out-of-distribution performance of TERMINATOR with respect to the compression rate **(left)** and the accuracy **(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.

at the *eventual final* answer, so any answer-changing self-correction falls before it. The LRM may also mention an answer-like value before it has completed its reasoning. Such superficial answer mentions do occur, but they primarily precede the position the pipeline selects. Isolating their causal effect would require detecting every candidate answer in a CoT, and unlike the final answer, these appear without a reliable logical anchor, so the measurement is not directly tractable. In both cases, the open question is whether the trained probe exits early at inference. We bin each CoT by where the pipeline's answer falls (an answer within the first 10% of tokens goes in the 0–10% bin) and plot the median and IQR of the signed difference between TERMINATOR's predicted exit and the pipeline's position in Fig. 14. Negative values mean that TERMINATOR exits earlier. Bin index acts as a difficulty proxy: a late-arriving answer means most of the CoT was spent deriving rather than verifying.

On MATH-500, the median TERMINATOR exit falls 6–7 tokens after the pipeline's position whenever that position lies within the first 70% of tokens, which covers 93% of CoTs; this matches the minimum delay of 6 tokens imposed by our strict majority vote (App. B.5). Premature exits appear only once the answer arrives after 70% of the CoT; those three bins hold only 7% of CoTs, so the behavior is rare. The trend is dataset-specific. On GPQA, the harder benchmark, the median is positive in every bin, so TERMINATOR does not exit early at the median. It exits later than the pipeline's position throughout, by a wide margin when the answer arrives early, but converges to a median of 1 token for the 1,251 CoTs whose answer appears in the last 10%. AIME25 resembles the shape of MATH-500, although its median turns negative only after 80% of the CoT, and those two bins hold 34% of AIME25 CoTs. HumanEval resembles the shape of GPQA over the first 80% of the CoT, but unlike GPQA, its median turns negative in the last two bins (10% of CoTs).

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.

C.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. 15 provides a rough estimate of the number of prompts needed to

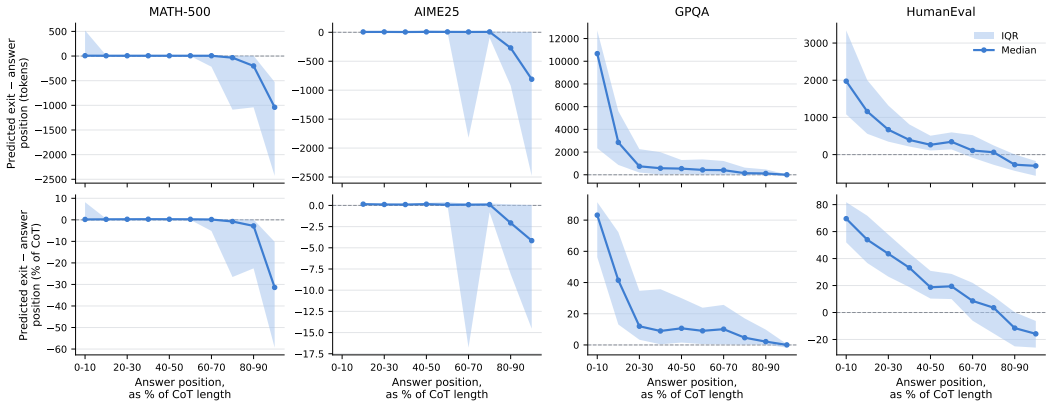


Figure 14: **Exit Timing Relative to Answer Arrival.** Each Qwen3-8B CoT is binned by where the pipeline's answer position falls in the CoT. We plot the median and IQR of the signed difference between TERMINATOR's predicted exit position and the answer position, in tokens **(top)** and as a percentage of CoT length **(bottom)**. Negative values indicate that TERMINATOR exits earlier.

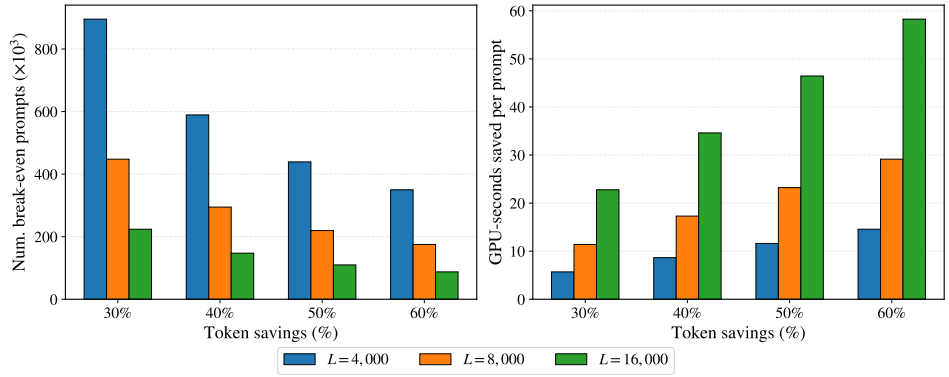


Figure 15: **Pipeline Cost-Benefit Analysis.** We report an estimate on the number of prompts to TERMINATOR needed to break even with the compute cost associated with running our data curation pipeline **(left)** and on the number of GPU-seconds saved per prompt **(right)**. We use the Qwen3-8B throughput data from Table 1 and calculate the compute 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 [14]. If just a single prompt were run per download on Qwen3-8B with TERMINATOR, the compute offset would be compensated for many times over.

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 [14]). 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.

C.5 Memory Overhead

We profile TERMINATOR's VRAM cost at inference on Qwen3-8B. TERMINATOR adds a static overhead but reduces peak KV cache by exiting earlier; the net effect on memory is favorable. At fixed GPU memory utilization, TERMINATOR reduces the available KV blocks by 3.2% and the KV pool by 0.5% (68.81 to 68.45 GiB; Table 7). Despite this, median peak KV cache per request falls 22% on GPQA and 50% on MATH-500 (Table 6).

Table 6: **Peak KV Cache per Request.** Peak KV cache per request for Qwen3-8B-based vanilla and TERMINATOR models (batch size 1). Mean values are reported as mean $\pm$ 95% CI.

Dataset	Method	Mean Peak KV (MiB)	Median Peak KV (MiB)
GPQA	*Vanilla*	1424.1 ± 306.5	1396.1
	TERMINATOR	1237.0 ± 314.9	1095.0
MATH-500	*Vanilla*	508.9 ± 174.2	348.8
	TERMINATOR	284.1 ± 155.9	175.8

Table 7: **Static Memory Footprint.** Static memory footprint of Qwen3-8B-based vanilla and TERMINATOR models at fixed GPU memory utilization. Head parameters are reported at BF16 serving precision (the FP32 checkpoint is 736.1 MiB on disk).

Model	Head Params (MiB)	Extra Layers	KV Blocks	KV Pool (GiB)
Vanilla-Qwen3-8B	–	0	31,317	68.81
TERMINATOR-Qwen3-8B	368.05	1	30,311	68.45

Profiling used 20 random samples per dataset, a batch size of 1, and disabled prefix caching. We report a batch size of 1 because our vLLM integration is an out-of-tree prototype in which the termination decision runs outside CUDA graph capture; supporting continuous batching is a serving-systems task rather than a change to the method.

C.6 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 8.

Table 8: **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 9: **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		Overall	
	MATH-500		AIME25		HumanEval		GPQA			
	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 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.

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 9 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 61.9% and compression rate of 24.1%, compared to 77.9% 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. C.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 9 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.

Table 10: **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. 1 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%**	79.1%	15,748	67.8%	96.5%	2,960	**66.6%**	58.2%	9,588	77.4%	80.7%	**64.9%**
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%	85.4%	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%**	**84.2%**	15,518	**63.5%**	**97.8%**	2,903	71.0%	**61.2%**	7,727	76.5%	**83.4%**	**63.7%**

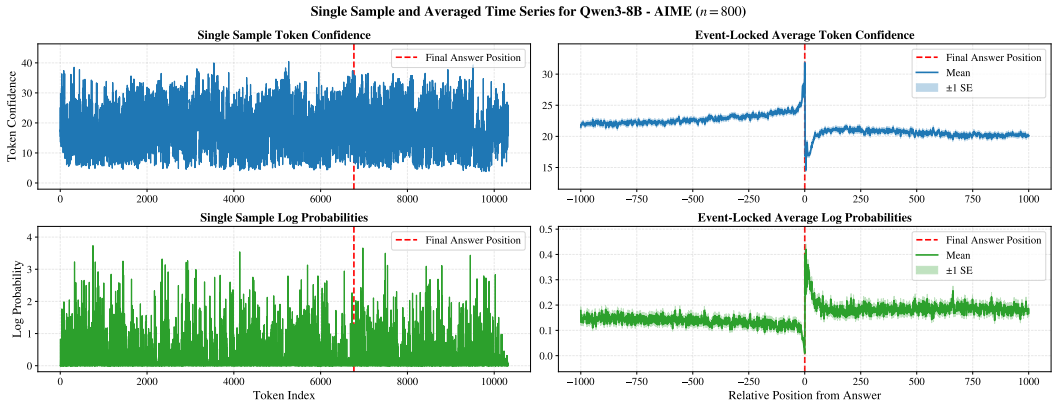


Figure 16: **Event-Locked Averaging of Token-Confidence for AIME (1983–2024).** A reproduction of Fig. 2, but only using CoTs from 800 randomly selected AIME (1983–2024) problems.

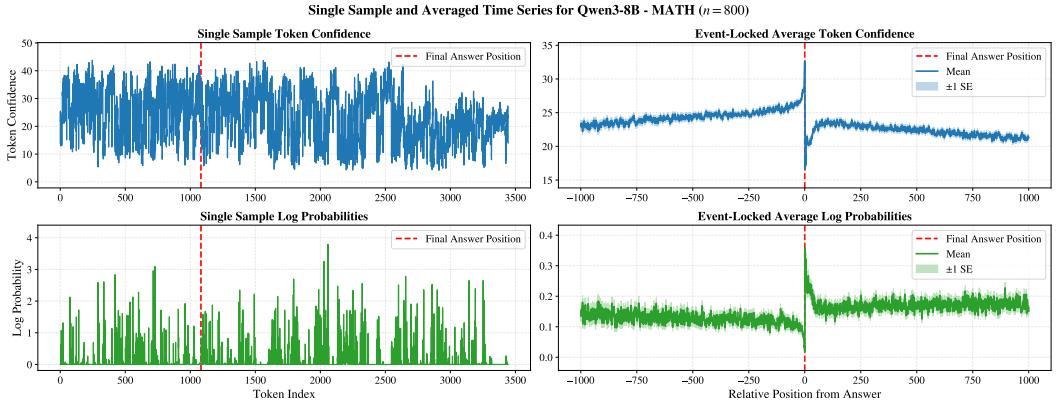


Figure 17: **Event-Locked Averaging of Token-Confidence for MATH.** A reproduction of Fig. 2, but only using CoTs from 800 randomly selected MATH problems.

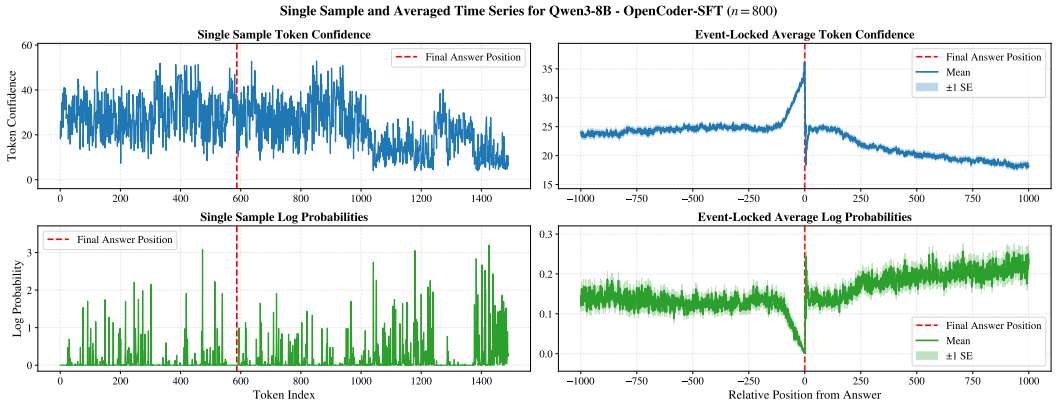


Figure 18: **Event-Locked Averaging of Token-Confidence for OpenCoder-SFT.** A reproduction of Fig. 2, but only using CoTs from 800 randomly selected OpenCoder-SFT problems.

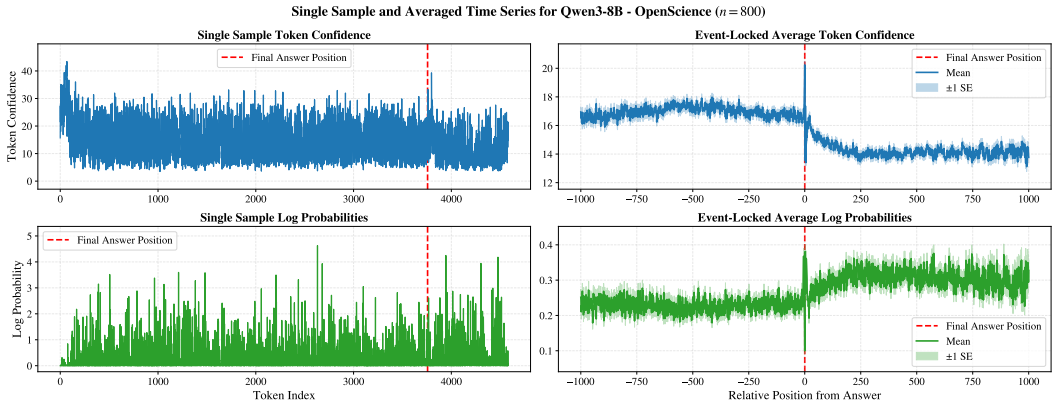


Figure 19: **Event-Locked Averaging of Token-Confidence for OpenScience.** A reproduction of Fig. 2, but only using CoTs from 800 randomly selected OpenScience problems.

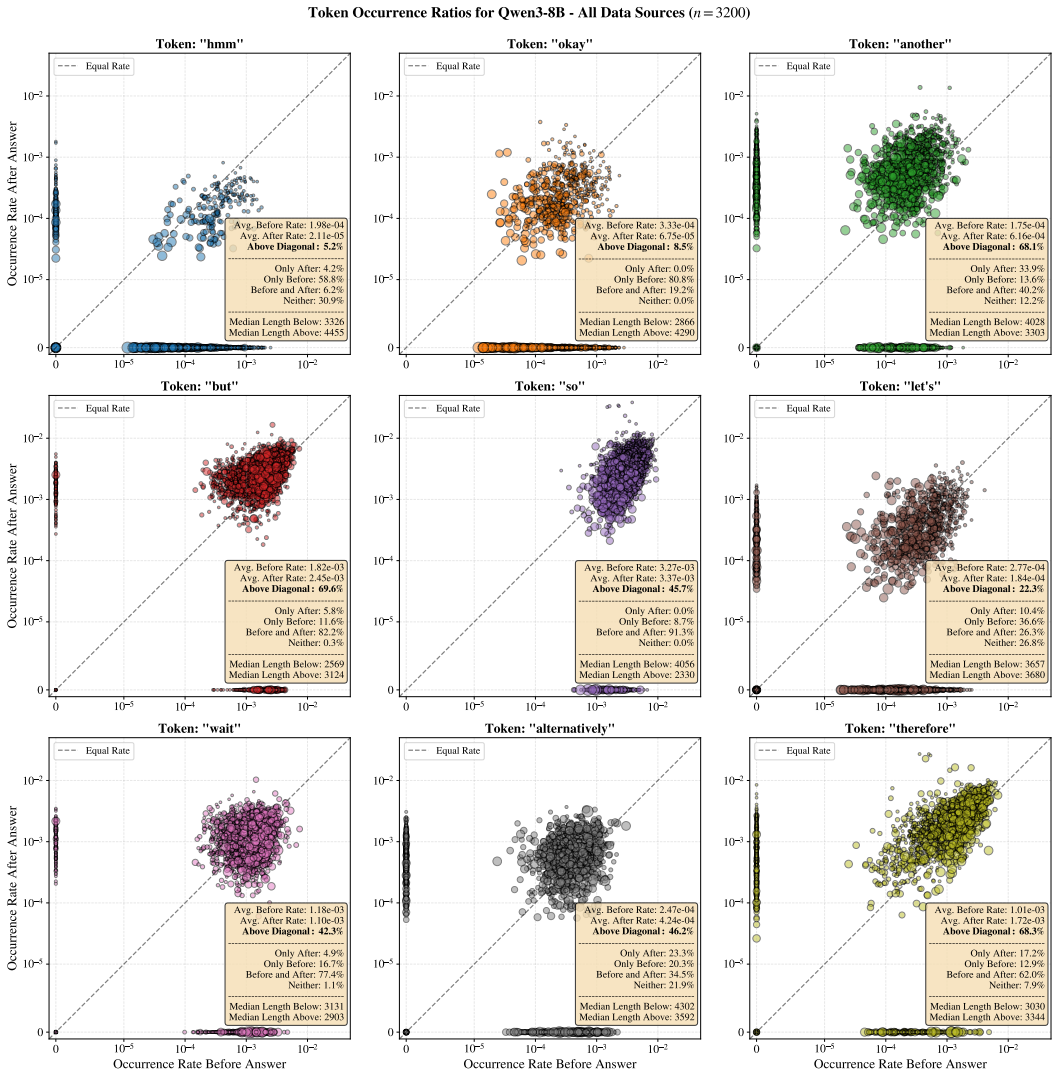


Figure 20: **Token Usage Frequency Shift.** An extension of the results shown in Fig. 3, highlighting additional "thinking tokens." While most "thinking tokens" shown here have some bias, often occurring before and after the first occurrence of the final answer as indicated by the **Above Diagonal** statistic, some tokens, like "`so`," are close to an equal rate on average.

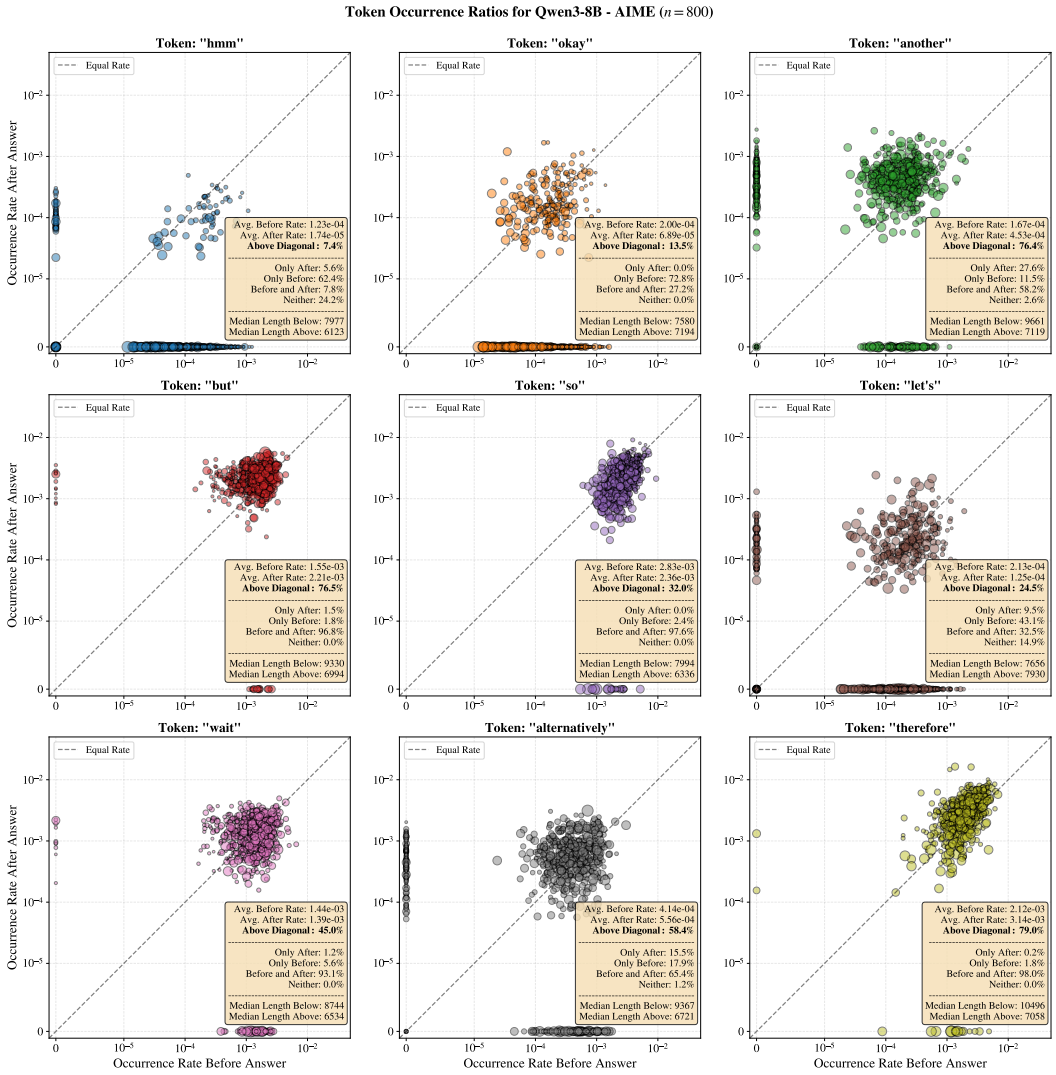


Figure 21: **Token Usage Frequency Shift for AIME (1983–2024.** A reproduction of Fig. 20, but only using CoTs from 800 randomly selected AIME (1983–2024) problems.

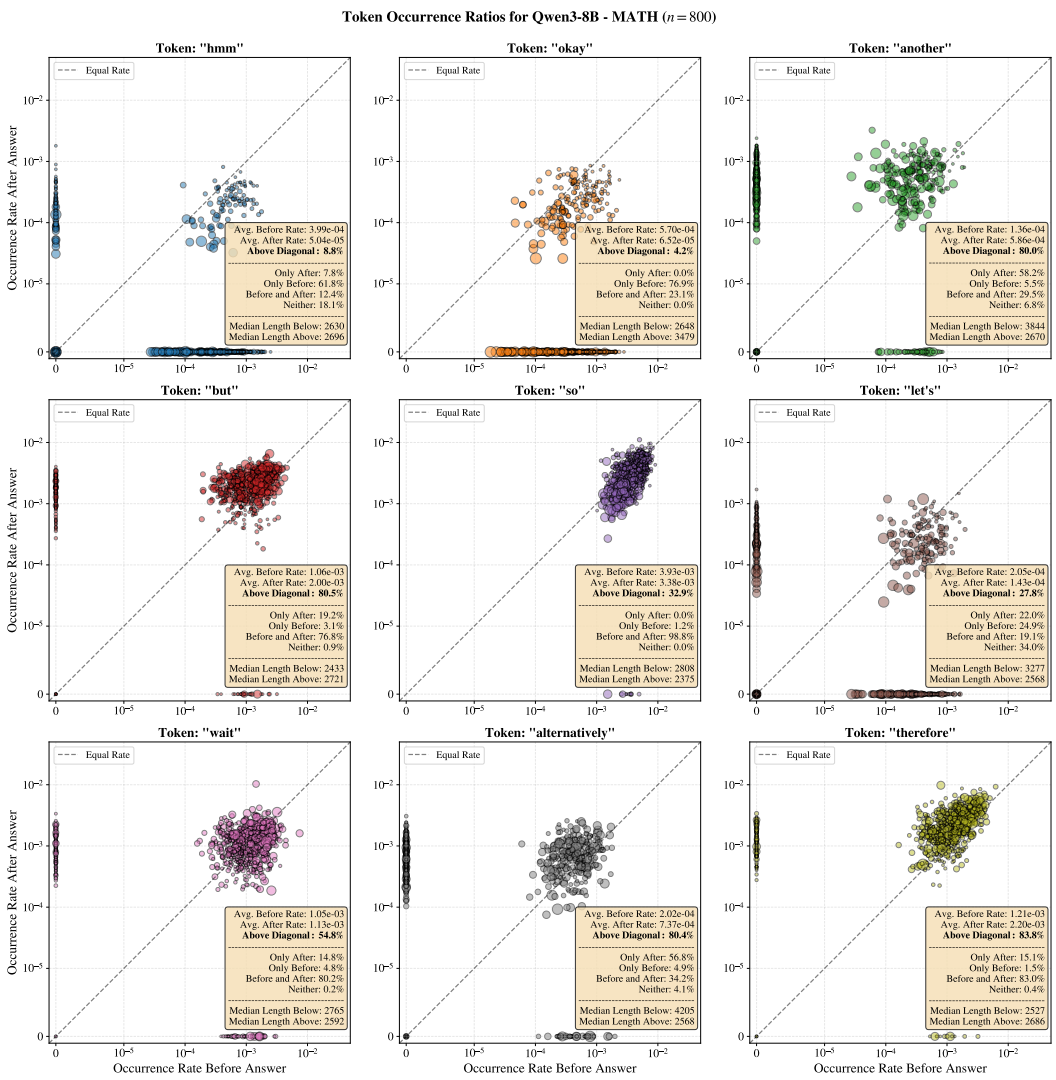


Figure 22: **Token Usage Frequency Shift for MATH.** A reproduction of Fig. 20, but only using CoTs from 800 randomly selected MATH problems.

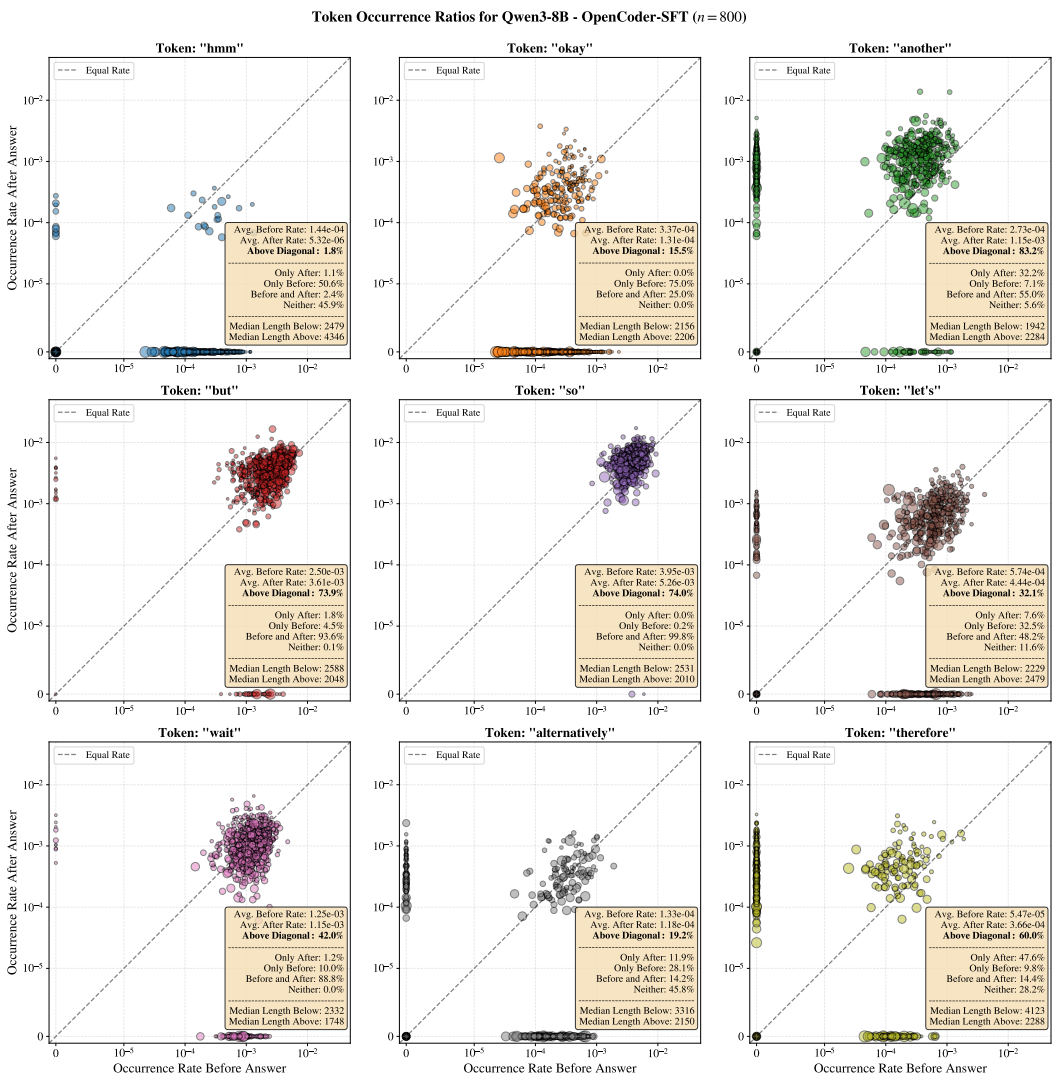


Figure 23: **Token Usage Frequency Shift for OpenCoder-SFT.** A reproduction of Fig. 20, but only using CoTs from 800 randomly selected OpenCoder-SFT problems.

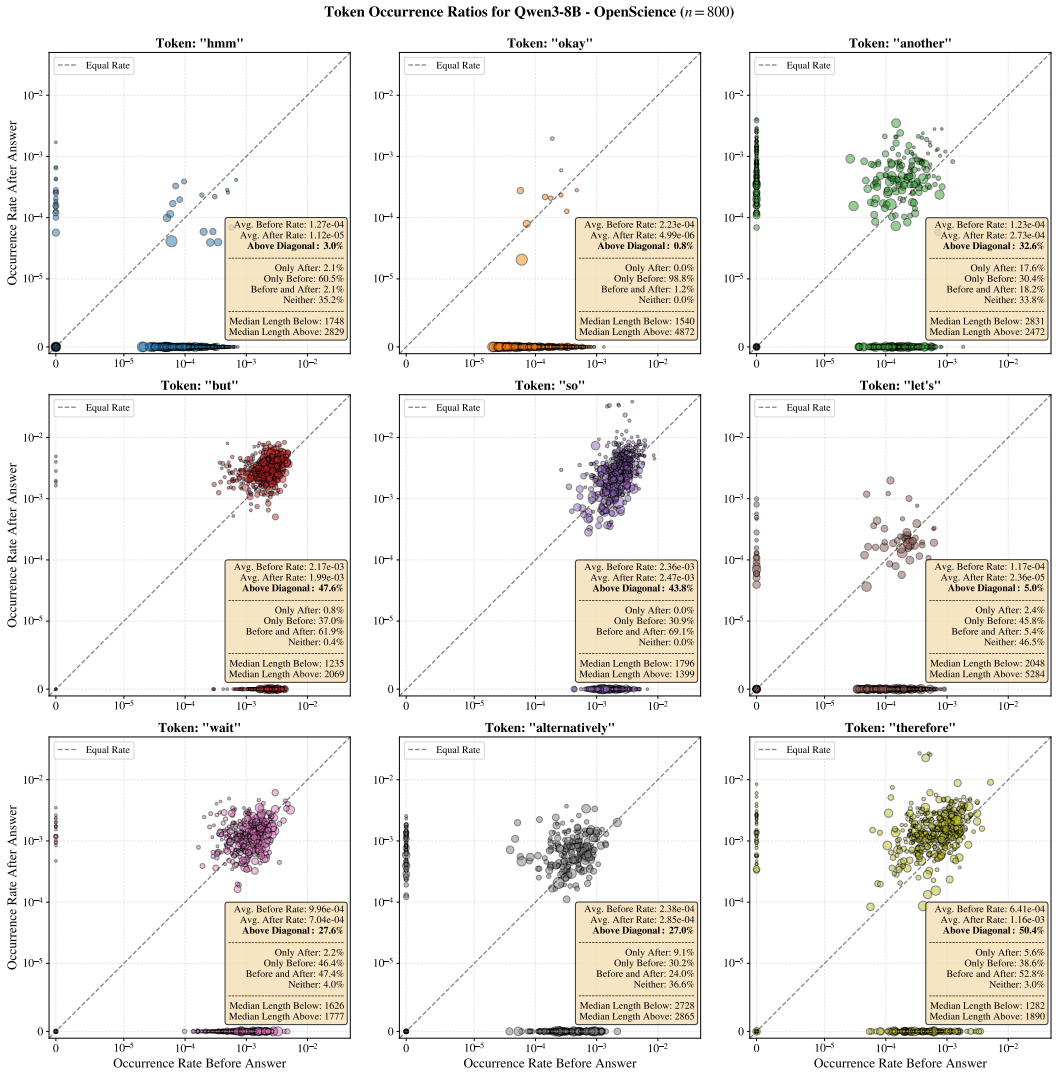


Figure 24: **Token Usage Frequency Shift for OpenScience.** A reproduction of Fig. 20, but only using CoTs from 800 randomly selected OpenScience problems.

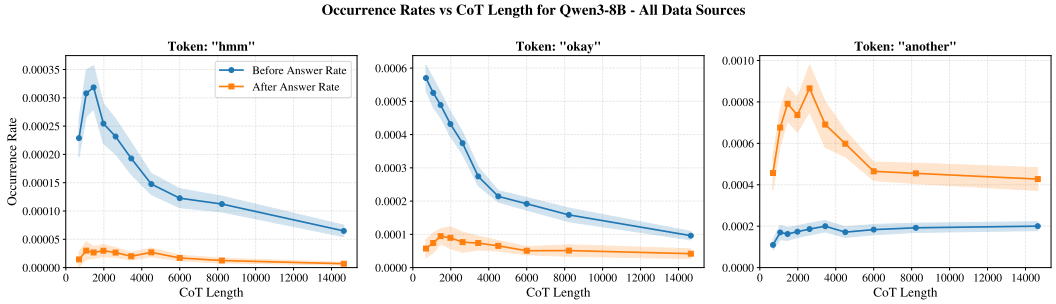


Figure 25: **Token Occurrence Rate vs CoT Length.** For some tokens, such as these three "thinking tokens," occurrence rates decrease rapidly as CoT length increases. Interestingly, the side of the answer (either "before" or "after") with the highest rate is the one that decays the most (the "before" 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.

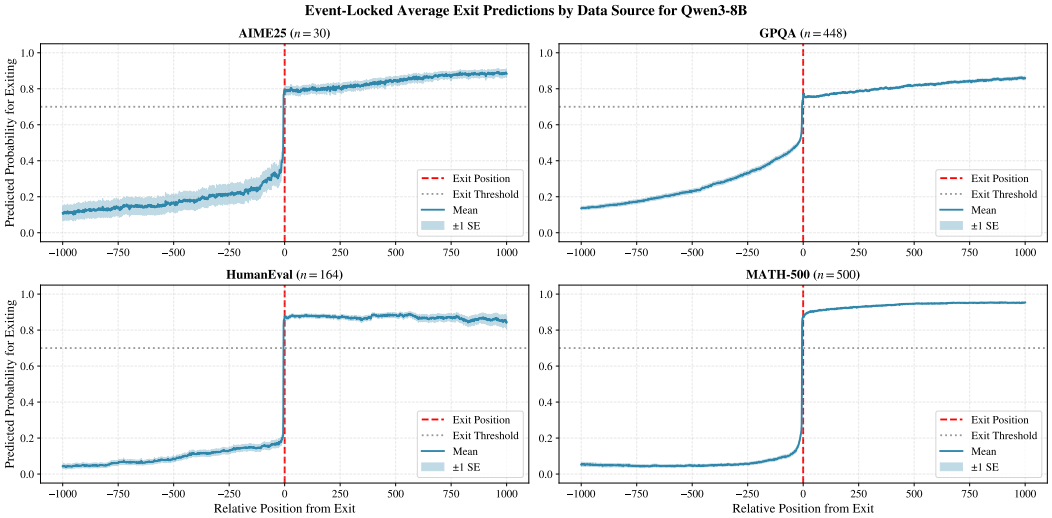


Figure 26: **Predicted Probabilities Event-Locked Averaging.** The dashed vertical line shows where TERMINATOR terminates the CoT with a sliding window of 10 and an exit threshold of 0.7, as indicated by the horizontal dotted line. We show the average predicted probability stream across all test problems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

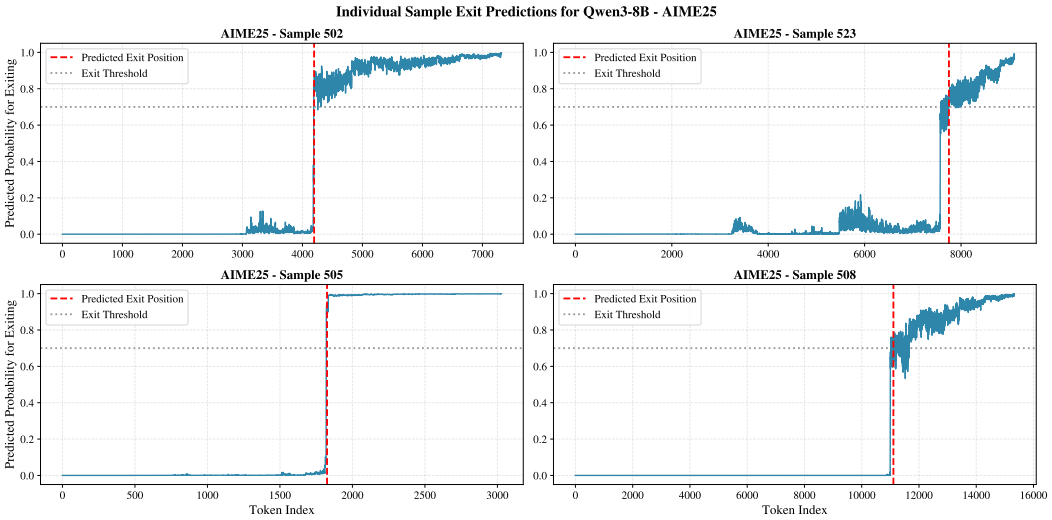


Figure 27: **Predicted Probabilities for AIME25.** TERMINATOR's predicted probability stream for early-exiting on four randomly chosen samples from AIME25.

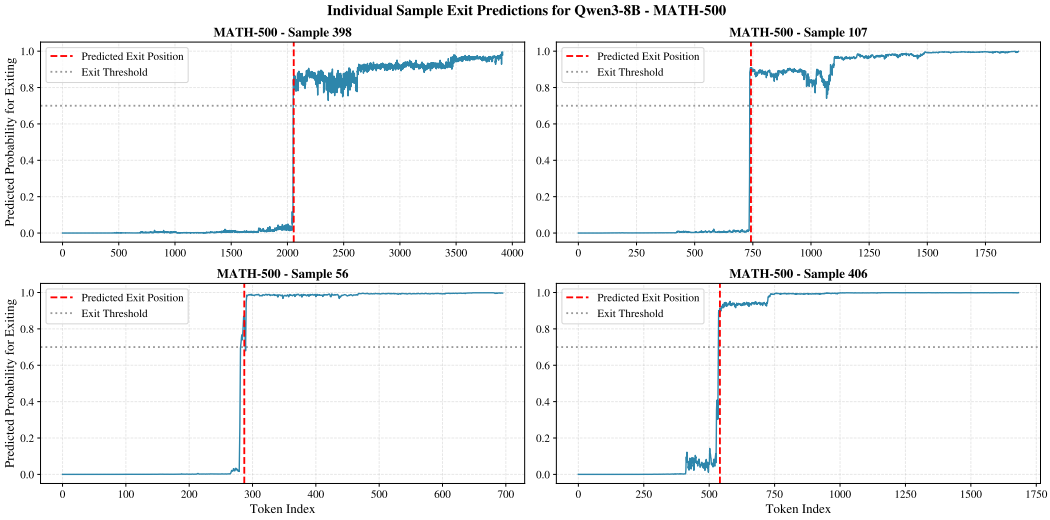


Figure 28: **Predicted Probabilities for MATH-500.** TERMINATOR's predicted probability stream for early-exiting on four randomly chosen samples from MATH-500.

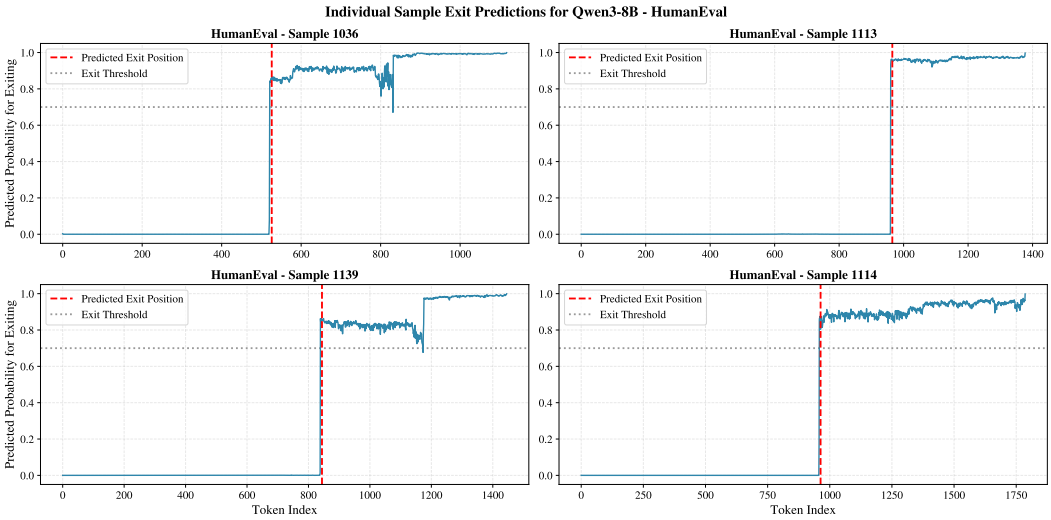


Figure 29: **Predicted Probabilities for HumanEval.** TERMINATOR's predicted probability stream for early-exiting on four randomly chosen samples from HumanEval.

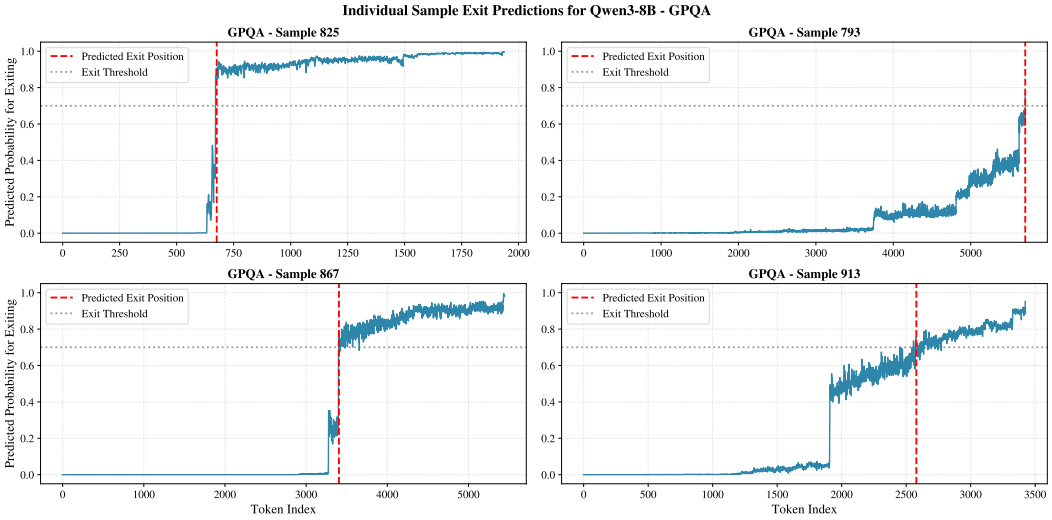


Figure 30: **Predicted Probabilities for GPQA.** TERMINATOR's predicted probability stream for early-exiting on four randomly chosen samples from GPQA.

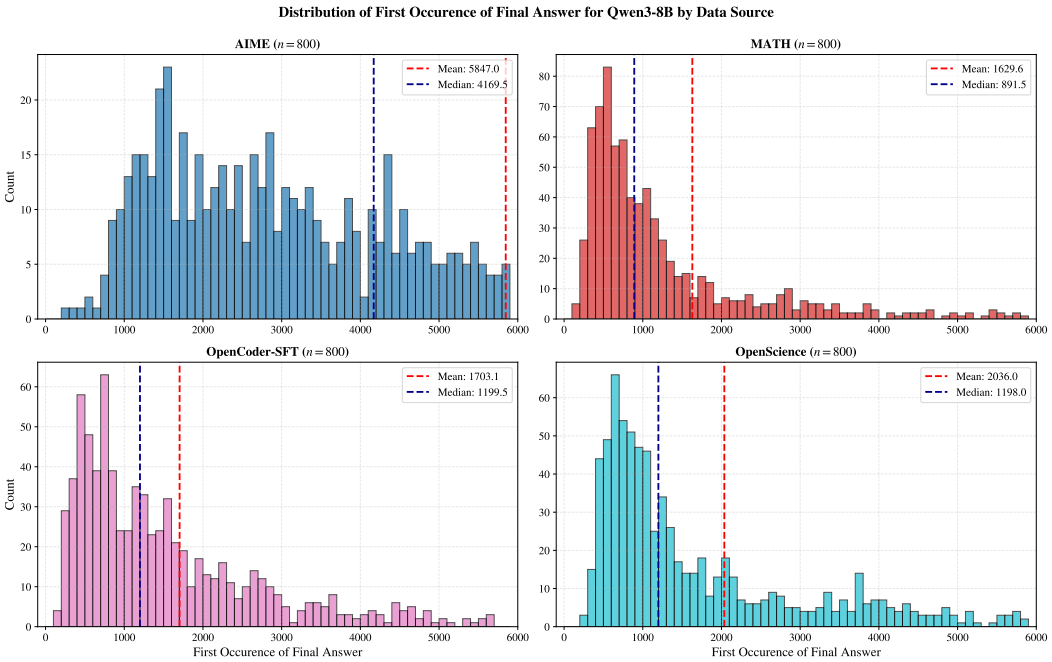


Figure 31: **First Answer Occurrence Histogram.** A histogram of the first occurrence of the final answer for each data source used in our training dataset is shown.

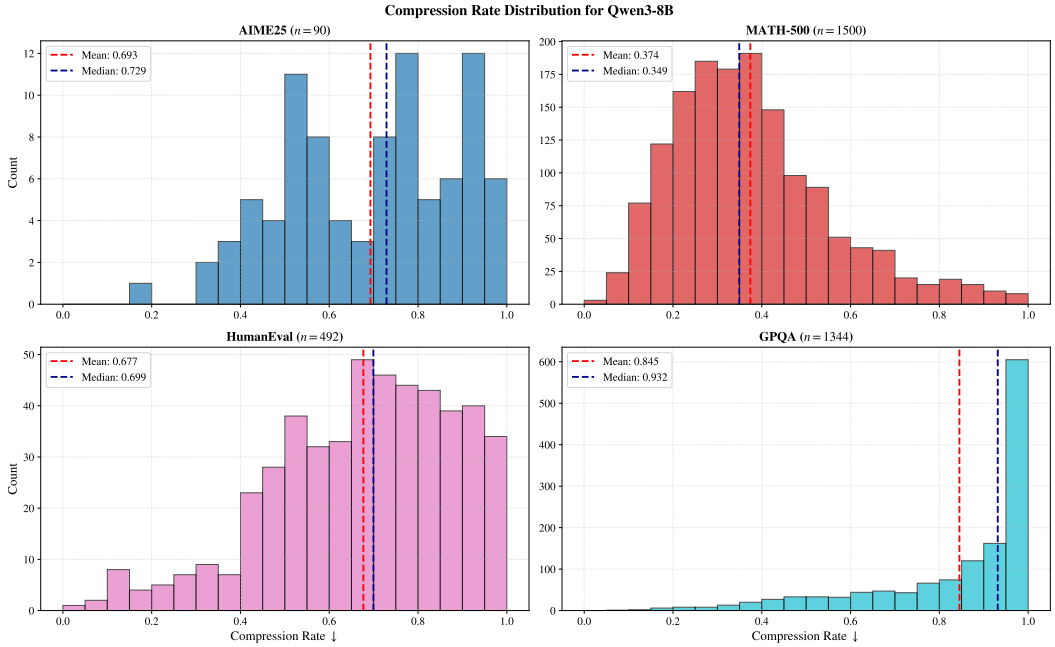


Figure 32: **Compression Ratio Histograms for TERMINATOR.** Each histogram shows the frequency of an achieved compression rate when early-exiting with TERMINATOR. ↓ indicates that a lower compression rate is better, as it results in more tokens saved with our method. Three CoTs are sampled per data source.



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.

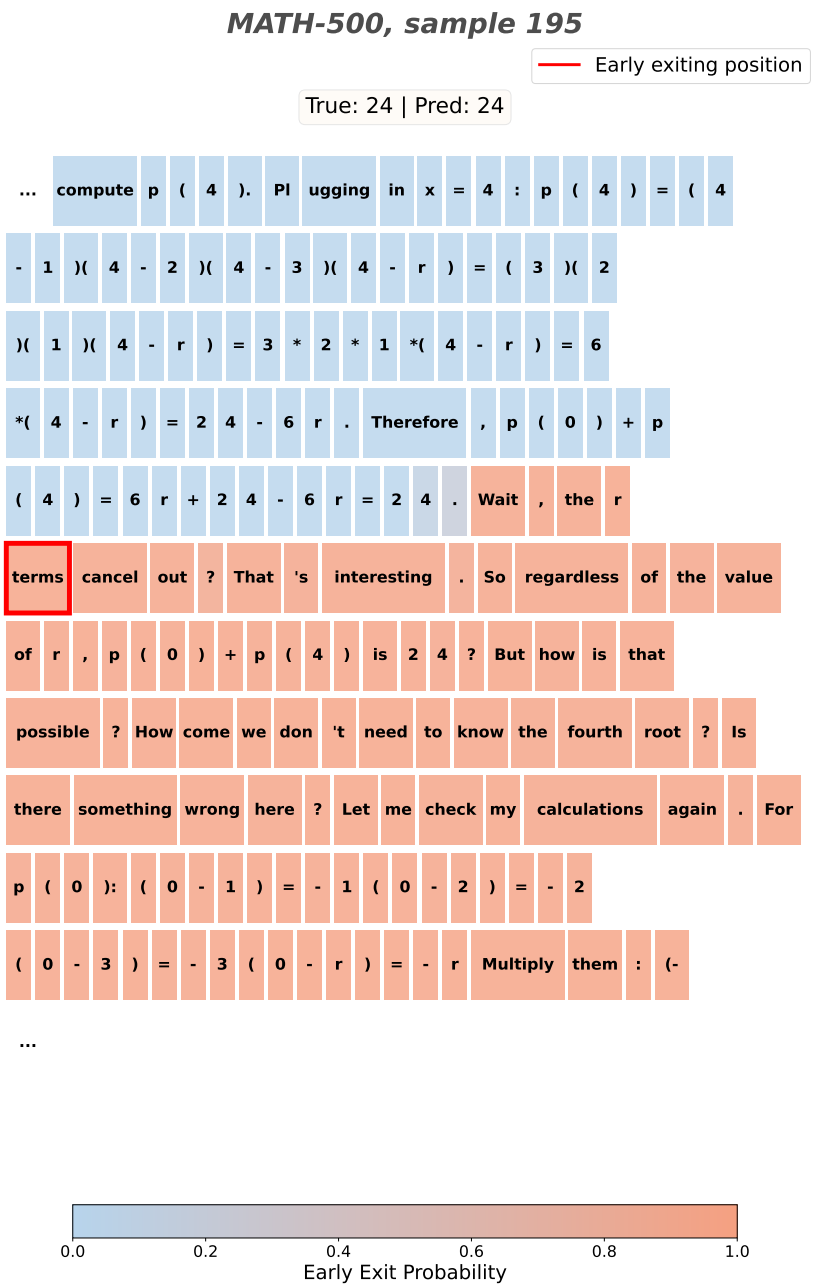


Figure 34: **Predicted Probabilities for MATH-500.** TERMINATOR's predicted probabilities for early-exit on a randomly chosen sample from MATH-500. The beginning and the end are truncated for better visibility.

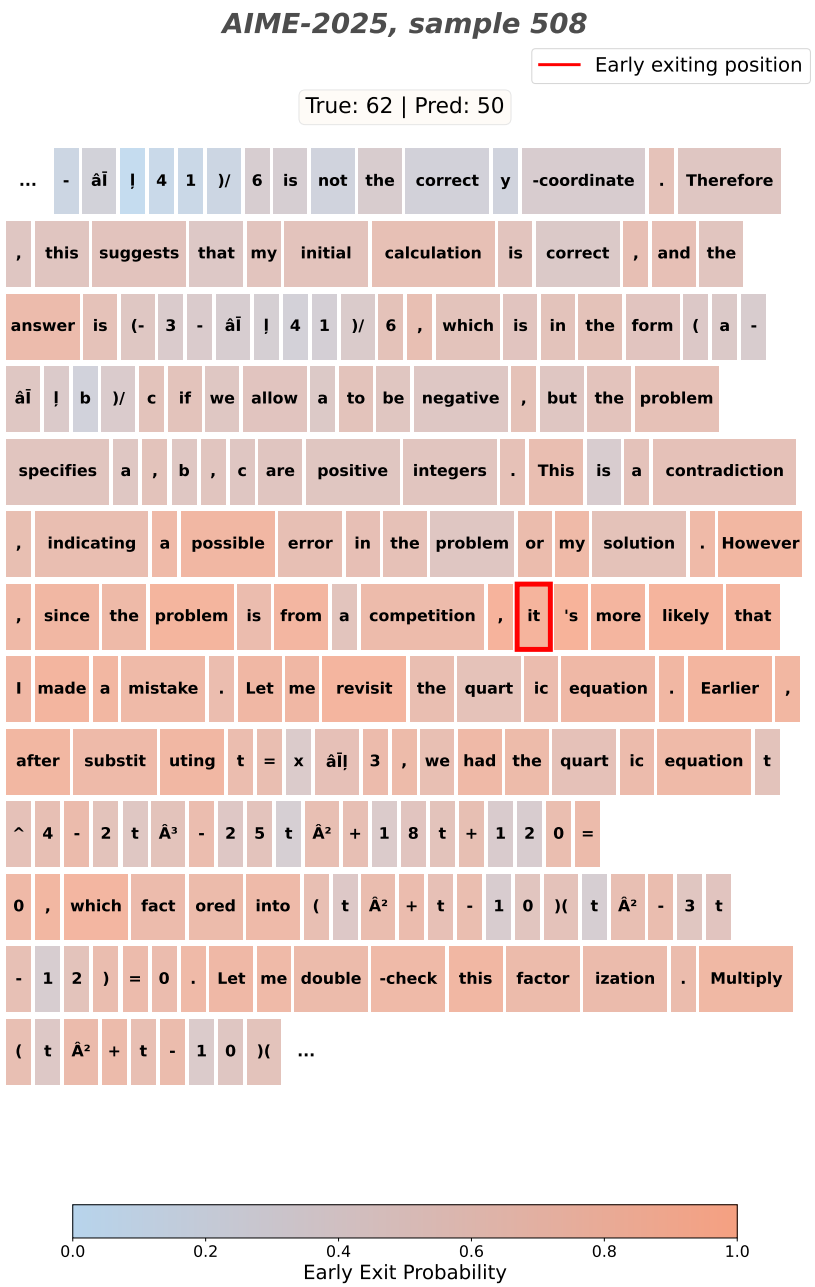


Figure 35: **Predicted Probabilities for AIME25.** TERMINATOR's predicted probabilities for early-exit on a randomly chosen sample from AIME-2025. The beginning and the end are truncated for better visibility.

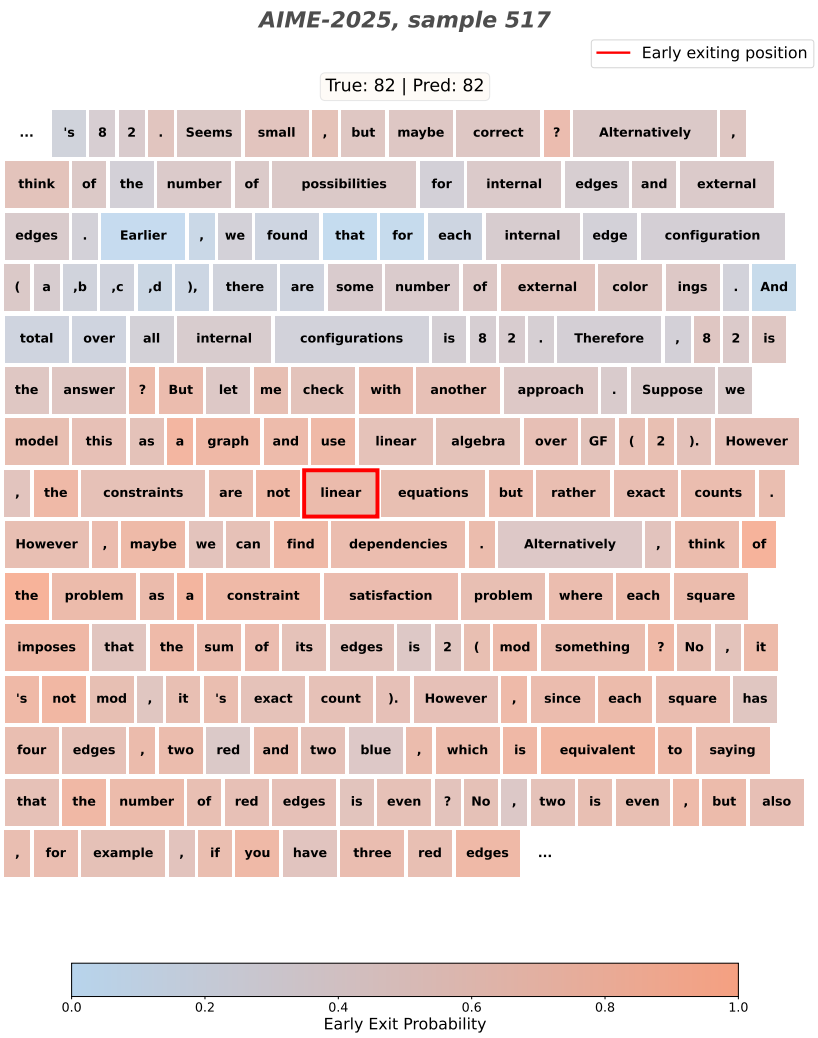


Figure 36: **Predicted Probabilities for AIME25.** TERMINATOR's predicted probabilities for early-exit on a randomly chosen sample from AIME-2025. The beginning and the end are truncated for better visibility.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We provide a boxed area which outlines the main contributions of our paper in the introduction Sec. 1.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Please see App. A.1.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

Justification: Our paper does not include theoretical results.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Our approach is fully outlined in Sec. 4. Additional details on training data, model training, and model inference is provided in Apps. B.3 to B.5.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We include our code and a sample of our training data as supplementary material, and we will fully release our code and data.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: See Apps. B.3 to B.5.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Statistical significance information is explicitly reported in Figs. 2, 3, 10 to 11, 16 to 24 and 27 to 30 and Table 1.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The details are given in App. C.4.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Our research conforms to the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Please see App. A.2.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: Our work builds on top of existing pre-trained language models, which have safeguards in place already. Our work does not further increase the risk of misuse.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We properly cite all owners of the assets from which we derive our work.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Our original assets are well-documented throughout the paper and in our code.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: We do not use crowdsourcing nor involve human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: We do not use crowdsourcing nor involve human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: Our pipeline for training dataset curation relies heavily on LRM usage, which is fully detailed in Sec. 4.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.