Mitigating Overthinking in Language Models Using Dynamic Stopping Criteria

Anonymous ACL submission

Abstract

Large language models (LLMs) often overthink, generating unnecessarily long chains-of-thought that waste computation and can even reduce accuracy. We propose a dynamic stopping framework that detects overthinking in real time and halts generation once further reasoning is predicted to be unproductive. Our method computes a novel overthinking score from the model's own output and internal signals, combined with domain-specific stopping triggers. Experiments on the MATH500 benchmark show that our approach can reduce 30% of the generation tokens on average while maintaining competitive accuracy (within 5% of the no-stopping baseline). We compare against recent overthinking mitigation methods and demonstrate that our method achieves a favorable balance of efficiency and reliability without requiring additional training or external calibration. Finally, we discuss interpretability insights and future directions for understanding and intervening in overthinking behavior in LLMs.

1 Introduction

Advanced large language models (LLMs) have demonstrated strong reasoning capabilities by generating extended chain-of-thought (CoT) solutions. However, a growing body of recent work has shown that such reasoning is often *excessive*: models frequently produce far more intermediate steps than necessary, especially on simpler problems, leading to wasted computation and, in some cases, degraded accuracy (Chen et al., 2025; Pu et al., 2025). This phenomenon, commonly referred to as *overthinking*, typically manifests as repetitive, redundant, or overly detailed explanations that do not meaningfully contribute to the final answer.

Despite increasing recognition of this issue, effectively managing overthinking remains challenging. To our knowledge, several problems remain open: (1) **Definition**: how to precisely characterize overthinking in a reasoning trace; (2) **Detection**: how to identify early signals that a model is entering an unproductive reasoning regime; (3) **Interpretation**: how internal or observable indicators (e.g., output patterns or logit behavior) relate to overthinking; and (4) **Intervention**: how to halt or redirect generation before unnecessary reasoning accumulates. In this work, we attempt to address these problems, focusing on the detection and intervention. We introduce an operational definition of overthinking grounded in repetition and hedging cues in the generated text (Section 3), and propose a dynamic stopping criterion that monitors model outputs *in real time* to decide when to terminate generation. Our stopping framework is lightweight, primarily output-based, and applicable to any LLM, with an optional grey-box extension, and requires no additional training.

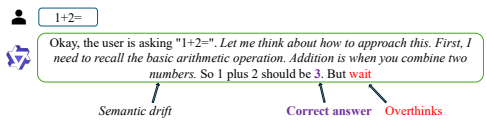

Figure 1: An illustration of overthinking: the model generates redundant or repetitive reasoning steps that do not improve the final answer.

The contributions of this paper are: (i) a novel dynamic stopping framework for mitigating overthinking, which uses a continuously updated *overthinking score* and other criteria to decide when to stop generation; (ii) a thorough evaluation on mathematical reasoning tasks demonstrating that our method significantly reduces generation length with minimal impact on accuracy; (iii) analysis comparing our approach with existing baselines and highlighting differences in assumptions and performance; and (iv) an expanded discussion on interpretability and future work, outlining how tools like the logit lens and attention analy-

sis could further illuminate overthinking in LLMs. We believe our approach is a step towards more efficient and controllable reasoning in large language models, aligning with the broader goal of intelligent test-time computation scaling.

2 Related Work

Overthinking in long reasoning. Overthinking in chain-of-thought generation has drawn attention as a key inefficiency of test-time scaling: models may spend many additional tokens on problems that do not benefit from extended reasoning. Chen et al. (2025) provide a comprehensive study of overthinking in long reasoning settings and propose mitigation strategies, with evaluation on benchmarks including GSM8K, MATH500, and AIME.

Token-budgeting and calibration-based decoding. A prominent line of work views overthinking as a calibration problem: for each question there exists an (unknown) compute budget beyond which additional tokens have diminishing returns. Pu et al. (2025) introduce THOUGHTTERMINATOR, a black-box decoding method that dynamically allocates a token budget based on an estimated problem difficulty, and propose DUMB500 as a stress test for overthinking on extremely easy problems. Their approach conditions generation on approximate difficulty and intervenes during decoding to encourage fewer tokens on easy instances and more on hard instances.

Budget forcing and supervised test-time scaling. Another related direction is *budget forcing* as part of a broader test-time scaling recipe. Muennighoff et al. (2025) present *s1*, which combines supervised fine-tuning on curated reasoning traces with budget forcing mechanisms that can either truncate thinking or encourage additional deliberation by appending tokens (e.g., a special "`[WAIT]`" token) when the model tries to stop early. Because s1 involves fine-tuning and carefully orchestrated decoding, it differs from training-free early stopping methods; however, it is conceptually connected through explicit control of test-time compute.

Dynamic early exit during generation. Most closely related to our method are training-free approaches that attempt to detect *when to stop* during generation. Yang et al. (2025) propose DEER (Dynamic Early Exit in Reasoning), which monitors

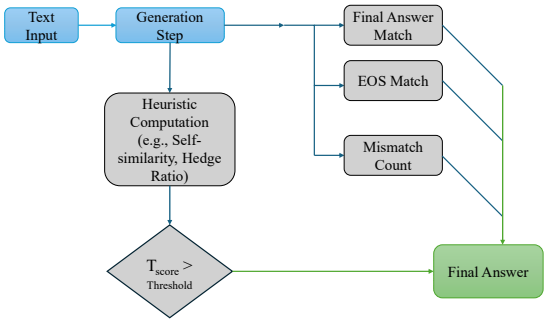

Figure 2: Overview of our dynamic stopping framework. After each generation step, we compute heuristic signals (e.g., self-similarity, hedge density) and calculate a t_{score} indicating potential overthinking. If the score exceeds a threshold, we inject a stop directive and collect the model's final answer.

potential reasoning transition points (e.g., occurrences of "Wait") and prompts the model to produce *trial answers* at these moments; early exiting decisions are then based on the model's confidence in the trial answer (using internal logits or an ensemble of checks). This design is effective when transition cues are frequent and confidence estimation is reliable, but it can depend on the presence of particular lexical markers and requires access to token-level probabilities to implement the decision rule.

Our position. Our method is training-free and targets *online* detection of unproductive continuations using signals computed from the model's *own output* (semantic repetition and hedging cues) together with lightweight consistency checks. Unlike difficulty-calibration methods, we do not require an explicit difficulty predictor; unlike methods operationalizing stopping via fixed cue tokens or special prompts, we make no strong assumptions about specific lexical markers. In Section 6, we compare directly with THOUGHTTERMINATOR and DEER under a consistent single-shot evaluation protocol.

3 Dynamic Overthinking Detection and Stopping Framework

Our framework monitors the model's chain-of-thought as it is generated and triggers an early stop when it detects signs of overthinking. We achieve this through a combination of immediate stop triggers and a continuously updated *overthinking score* that quantifies the model's tendency to overthink at each step (Figure 2). We describe

the main components below.

3.1 Immediate Termination Triggers

Certain signals indicate overthinking so strongly that we terminate generation as soon as they appear, overriding other criteria. Specifically:

- **Final-Answer Indicators**: If the model produces a known end-of-reasoning token or phrase (e.g., an explicit token like ⟨End_of_Solution⟩ or a phrase such as "final answer","boxed","correct answer", "best answer"), we interpret that the model is ready to give a final answer. We allow a small number (here 2) of such occurrences; on the second instance of a final-answer indicator, we stop the reasoning and prompt the model to output the final answer immediately (to avoid the model stating multiple "final" answers). We also use a counter for word "answer" for a high count of 30 or similar as its a frequently used word.
- **Logit Lens Mismatch**: We employ a simple *logit lens* analysis during generation (Nostalgebraist, 2020). After each new token in the reasoning, we examine the model's next-token prediction distribution *from an earlier layer* of the network (we use a middle layer as a probe). If the token that was actually chosen by the model at the output layer has low probability according to the probe layer's distribution, this indicates a *mismatch* or surprise in the model's token choice, which can signal confusion or a shift in context. Our framework counts these mismatches cumulatively. If the count of significant mismatches exceeds a threshold (set to 35 in our experiments), we trigger an immediate stop, hypothesizing that the model's hidden state has diverged considerably from a confident trajectory. This criterion is inspired by observations that a sudden spike in token-level inconsistencies often precedes the model reaching a correct answer or drastically changing its approach (a phenomenon we term "semantic drift" following a confusion spike). The logit-lens mismatch trigger is not required for the framework to function and serves primarily as a safety net in rare pathological cases.

These immediate triggers act as hard safety nets. In practice, the final-answer indicator is crucial for problems where the model might otherwise continue explaining after arriving at an answer, and the mismatch trigger is a proxy for catching severe reasoning derailments. In our ablation studies (Section 5.2 and table 3), we analyze the impact of removing these triggers.

3.2 Dynamic Overthinking Score

Beyond the hard triggers above, our core contribution is a *dynamic overthinking score* (t_{score}) that is computed for each new sentence (or reasoning step) the model generates. Each Sentence is identified by punctuation marks. The intuition is to continuously estimate how likely the model is overthinking at this point, based on features of the generated text itself. We consider three key features:

1. **Semantic Similarity (Sim)**: We compute the maximum cosine similarity between the embedding of the current sentence and the embeddings of all previous sentences in the reasoning so far. A high similarity suggests the model might be essentially *repeating* a point it already made earlier, which is a strong sign of overthinking or circling on the same idea. We use sentence embeddings from a pre-trained model (all-MiniLM-L6-v2), updating all embeddings as the output grows.
2. **Repetition Ratio (RepRatio)**: This metric captures how much the model is reusing exact tokens or phrases. We define RepRatio as the fraction of n-grams in the current sentence that have appeared before in the model's output. In implementation, we use a simplified approximation: we track if a sentence is paraphrasing a previous one by checking if its similarity with any prior sentence exceeds a threshold. This yields a coarse repetition fraction.
3. **Hedge Score (Hedge)**: We count occurrences of *hedging or stalling words* in the current sentence (e.g., "maybe", "I think", "let me double-check", "however") and divide by the sentence length. A high density of tentative or self-correcting phrases indicates the model might be stalling or second-guessing itself. The full list of hedging markers used to compute the Hedge signal is provided in Appendix C.

At each step, we combine these features into the overthinking score:

$$t_{\text{score}} = \lambda_1 \cdot \text{Sim} + \lambda_2 \cdot \text{RepRatio} + \lambda_3 \cdot \text{Hedge}\ , \quad (1)$$

with $\lambda_1 + \lambda_2 + \lambda_3 = 1$. We chose a weighted

linear combination as it provides flexibility to capture different manifestations of overthinking. The weights λ_1, λ_2 and λ_3 are hyperparameters.

We compare t_{score} against a threshold τ (tuned to 0.925). If $t_{\text{score}} > \tau$ for the current sentence, it indicates a high likelihood of overthinking, and we prepare to trigger a stop. If t_{score} remains below τ, generation continues normally.

Activation Delay: We do not activate the t_{score} mechanism from the very start of generation. To avoid spurious triggers when the model is just beginning to work on the problem, we introduce an *activation length* L_{act}. The framework ignores t_{score} (and mismatch count) for the first L_{act} tokens of the model's output, relying only on the final-answer indicator in that initial phase. We set $L_{\text{act}} = 160$ tokens based on preliminary experiments. In practice, this allows the model to lay out an approach without interruption.

3.3 Intervention and Stopping Mechanism

Once the framework is active, as the model generates each new sentence, we perform the checks above:

1. If any immediate termination trigger fires (e.g., a final answer phrase is repeated, or mismatch count exceeds threshold), we halt the generation at once.
2. Otherwise, we compute t_{score} for the new sentence. If $t_{\text{score}} > \tau$, we interpret this as the model entering an overthinking regime. At this point, we intervene by instructing the model to stop reasoning.

The intervention is implemented by injecting a final prompt to the model to produce the answer directly. Specifically, we append a short directive to the model's context: *"[STOP] Provide the final answer now in one sentence."* (In our experiments with math problems, we used a phrasing: `### Final answer only. Do not explain. Just output the answer in 35 tokens or less:`, to align with the prompt format.) This causes the model to output the final answer it has in mind, without any further reasoning. By doing so, we cut off the chain-of-thought at an opportune moment and prevent additional redundant steps. The model's answer is then compared to the ground truth for evaluation.

Our stopping criteria aim to intervene *just after* the model has likely found a solution or when continuing would mostly yield repetition. If the model truly has not solved the problem yet (and is not looping), t_{score} tends to remain below the threshold, allowing reasoning to continue. In Section 5, we show that our method preserves accuracy on difficult problems while dramatically reducing tokens on easier ones, demonstrating difficulty-sensitive adaptation.

4 Experiments and setup

4.1 Dataset and Tasks

We primarily evaluate on MATH500, a benchmark of 500 challenging math word problems derived from the MATH dataset (Hendrycks et al., 2021). MATH500 is well-suited for testing overthinking because it contains problems of varying difficulty (some can be solved in a few steps, others require extended reasoning), and it was used by prior overthinking mitigation work for evaluation (Pu et al., 2025; Yang et al., 2025). We use the standard test split for MATH500.

We also evaluate on two additional reasoning-intensive benchmarks to test generalization: the GPQA dataset (Graduate-Level Problem QA) (Rein et al., 2023) and the Misguided Attention dataset (cpldcpu, 2025). We observe similar trends on CommonsenseQA (Talmor et al., 2019). These represent different domains (scientific QA, common sense reasoning and robustness to distractors). Due to space constraints, details and results for these datasets are presented in Appendix A.

4.2 Models

We experiment with open-weight large language models from the DeepSeek-R1 and LLaMA families (DeepSeek-AI, 2025). In particular, we evaluate DeepSeek-R1-8B variants, which are fine-tuned for mathematical and logical reasoning and are comparable in scale to LLaMA-8B models trained with reasoning distillation. These reasoning-specialized models frequently produce overly long chains of thought on benchmarks such as MATH500, making them a suitable testbed for studying overthinking and dynamic stopping. All model inference is performed using 4-bit quantization for efficiency, and the same quantization setting is applied uniformly across all baselines to ensure a fair comparison.

We additionally evaluate LLaMA-3.1 8B Instruct (Meta AI, 2024), a general-purpose instruction-tuned language model, to assess

whether dynamic stopping generalizes beyond reasoning-specialized models. The results are mentioned in Sections 1 and 5.

4.3 Decoding Strategies

We consider three decoding settings for generating the chain-of-thought: (1) Greedy decoding ($T = 0$), which yields a deterministic CoT; (2) Near-deterministic sampling, with an extremely low temperature ($T = 10^{-7}$) and nucleus sampling ($p = 0.95$), which preserves determinism in practice while still using the sampling interface; and (3) Low-temperature stochastic decoding, with $T = 0.6$ and $p = 0.95$, which introduces moderate randomness allowing exploration of different reasoning trajectories. Unless otherwise specified, all main results use the near-deterministic setting, but we verified that our stopping framework remains effective under both fully greedy and moderately stochastic decoding.

4.4 Implementation and Hyperparameter Tuning

We implemented our framework to work with 7B–13B scale transformer models (in particular, the DeepSeek series of LLMs, similar to LLaMA in architecture). Optuna was used for hyperparameter tuning. We ran 50 trials on a held-out set, using tree-structured Parzen estimators to guide the search. The parameters tuned included the weights $\lambda_1, \lambda_2, \lambda_3$ in t_{score}, the threshold τ, the mismatch count threshold, and the activation length L_{act}. Each trial evaluated 20 sample problems. The search suggested a threshold $\tau \approx 0.93$, a mismatch threshold at the maximum tested (35), and $L_{\text{act}} \approx 160$. We fixed these values and λ as $\lambda_1 = 0.50$, $\lambda_2 = 0.375$, and $\lambda_3 - 0.125$ for all final experiments, and found they generalized well across model sizes.

4.5 Evaluation Metrics

Primary metrics. We evaluate each method using two primary metrics that directly reflect effectiveness and efficiency:

- Accuracy: Whether the final answer produced is correct. For mathematical problems, we strip formatting (e.g., remove any $boxed$) and compare against the ground truth, allowing minor numeric or symbolic equivalences (via SymPy).
- Tokens Generated: The total number of tokens produced in the chain-of-thought *before* the final answer. This serves as our main proxy for inference-time computation. We report average tokens per problem and relative token savings compared to the no-stopping baseline.

Analysis and diagnostic metrics. In addition to the primary evaluation metrics above, we compute several auxiliary metrics to analyze the behavior of the proposed stopping mechanism and to illustrate how overthinking manifests during generation:

- Mismatch Frequency: The average number of logit-lens mismatch events per problem (Section 3), used as a diagnostic of internal inconsistency during reasoning.
- Semantic Drift: The average cosine distance between embeddings of reasoning sentences and the problem statement, measuring topical drift over the course of generation.
- Repetition Rate: The fraction of non-stopword tokens in a solution that have appeared earlier in the same solution, capturing redundancy and looping behavior.

These diagnostic measures are not used for model selection or quantitative comparison across methods, but serve to illustrate the qualitative effects of dynamic stopping. Our evaluation script extracts the final answer from halted generations using consistent regex-based parsing for all methods, following prior work (e.g., Yang et al., 2025), ensuring fair comparison.

Following prior work on overthinking and test-time scaling, we measure efficiency primarily in terms of the number of generated tokens, which directly reflects model-side computation and inference cost under fixed decoding settings. Wall-clock runtime depends on implementation and hardware characteristics and is therefore reported in the appendix for reproducibility.

4.6 Baselines

We compare our dynamic stopping framework against the following baselines:

(1) **Vanilla CoT**: the model generates a full chain-of-thought and final answer without intervention, reflecting the original overthinking-prone behavior.

(2) **Fixed Budget**: reasoning is truncated after a fixed number of tokens (256), illustrating the trade-off between computation and accuracy under aggressive stopping.

(3) **ThoughtTerminator** (Pu et al., 2025): We

simulate ThoughtTerminator using a fixed 5000-token budget with periodic (every 250 tokens) budget reminders and omit difficulty prediction to avoid task-specific training or oracle assumptions. This conservative upper-bound configuration preserves reasoning capacity but empirically results in lower accuracy than the baseline while achieving only $\sim$2% token savings, suggesting that constant prompting alone is ineffective for reducing overthinking.

(4) **DEER** (Yang et al., 2025): we use the authors' released implementation. DEER monitors for cue tokens (e.g., "Wait") during generation and injects a prompt for a trial final answer when such cues appear. We evaluate DEER under a single-answer protocol.

(5) **s1 (Simple Test-Time Scaling)** (Muennighoff et al., 2025): s1 relies on supervised fine-tuning and special tokens to control reasoning length. As this is outside our training-free scope, we include s1 for discussion only and refer to reported results where appropriate.

5 Results and Analysis

5.1 Main Results

Table 1 summarizes accuracy and average generation length on MATH500 for our method and baselines.

Results on DeepSeek-R1. Without any stopping, DeepSeek-R1-8B achieves high accuracy (82.6%) but produces very long chains-of-thought (around 2700 tokens per problem on average, with a cap at 5000 tokens). This is computationally expensive. A fixed budget of 256 tokens dramatically reduces length (over 90% fewer tokens) but yields only 50.5% accuracy, as many problems require more reasoning steps than the budget allows.

The ThoughtTerminator strategy (with a 5000-token budget) achieves 69.6% accuracy while barely reducing token usage (only $\sim$2% savings), indicating that static or weakly adaptive budgets alone are insufficient to mitigate overthinking on this dataset. In contrast, DEER substantially shortens generations: on DeepSeek-R1-8B, DEER reduces the average chain-of-thought to approximately 1500 tokens (saving $\sim$43%), but its pass@1 accuracy drops to around 60%. These results suggest that cue-based early-exit mechanisms can significantly improve efficiency but may incur notable accuracy degradation when confidence estimates or stopping cues are unreliable. Our dynamic stopping framework provides a more favorable balance. Under near-deterministic decoding, it reaches 81.4% accuracy on DeepSeek-R1-8B while using about 1900 tokens on average—a 30.5% reduction in tokens relative to the no-stopping baseline. This efficiency gain comes at only a minor accuracy cost (within 1.2% of the baseline). On easier subsets of MATH500, our method stops much earlier, yielding substantially larger savings (over 60% token reduction on the easiest 100 problems) without sacrificing correctness.

Results on LLaMA-3.1 8B Instruct. We additionally evaluate our framework on LLaMA-3.1 8B Instruct (Meta AI, 2024), a general-purpose instruction-tuned model that is not explicitly optimized for long-form mathematical reasoning. In this setting for MATH500 (Hendrycks et al., 2021), the no-stopping baseline achieves only 57.4% accuracy while generating lengthy explanations. Applying our dynamic stopping framework with the same decoding setup yields a qualitatively different outcome: with the standard threshold ($\tau = 0.925$), accuracy *improves* to 59.4% while reducing reasoning tokens by 72.76%. Using a more aggressive threshold ($\tau = 0.55$) further increases token savings to 80%, at the cost of a modest accuracy drop (55.4%).

These results suggest that overthinking can be actively harmful for instruction-tuned models: excessive chain-of-thought often manifests as speculative or redundant reasoning that does not improve, and can even degrade, the final answer. In such cases, dynamic stopping not only improves efficiency but can also improve accuracy by truncating unproductive continuation.

Overall, across both reasoning-specialized (DeepSeek-R1) and instruction-tuned (LLaMA-3.1) models, dynamic stopping consistently adapts reasoning length on the fly and offers a strong accuracy–efficiency trade-off. Compared to static truncation or cue-based early exit, it provides a unified mechanism that mitigates overthinking without relying on fixed budgets, external difficulty predictors, or task-specific lexical cues.

In addition to average performance, we observed that an overly aggressive stopping policy can severely hurt accuracy. For example, our initial prototype cut about 60% of tokens but answered only 50% of questions correctly, underscoring the importance of carefully tuning the

Dataset	Method	Model / Decode	Acc. (%)	Avg Tokens	Tokens Saved (%)
MATH500	Vanilla CoT	DS-R1-8B, $T = 10^{-7}$	82.6	2700	0.0
	Fixed Budget	DS-R1-8B, 256 tokens	50.5	256	90.5
	ThoughtTerminator	DS-R1-8B, $T = 10^{-7}$	69.6	2500	2.0
	DEER (official)	DS-R1-8B, $T = 0$	60.2	1532	43.3
	DEER (official)	DS-R1-8B, $T = 10^{-7}$	60.0	1525	43.5
	Ours (Dynamic Stop)	DS-R1-8B, $T = 10^{-7}$	**81.4**	1900	**30.5**
	Vanilla CoT	LLaMA-3.1-8B Instruct, $T = 10^{-7}$	57.4	4555	0.0
	Ours (Dynamic Stop)	LLaMA-3.1-8B Instruct, $T = 10^{-7}$	**59.4**	1241	**72.76**
GPQA	Vanilla CoT	DS-R1-8B, $T = 10^{-7}$	42.85	–	0.0
	Ours (Dynamic Stop)	DS-R1-8B, $T = 10^{-7}$	40.84	–	30.92
	Vanilla CoT	LLaMA-3.1-8B Instruct, $T = 10^{-7}$	29.67	–	0.0
	Ours (Dynamic Stop)	LLaMA-3.1-8B Instruct, $T = 10^{-7}$	**30.0**	–	**62.0**
CSQA	Vanilla CoT	DS-R1-8B, $T = 10^{-7}$	59.56	–	0.0
	Ours (Dynamic Stop)	DS-R1-8B, $T = 10^{-7}$	58.9	–	26.76

Table 1: Main results across datasets and models. DS-R1 denotes DeepSeek-R1. Tokens Saved are reported relative to the corresponding Vanilla CoT baseline under the same decoding setting. DEER results use the authors' released implementation and are evaluated under a pass@1 (single-answer) protocol.

stopping criteria. Our final approach achieves a much better balance (81% accuracy with 30% token reduction).

5.2 Ablation Studies

We ablated each component of our framework to assess its contribution: Removing the dynamic t_{score} (i.e., disabling the score-based trigger and relying only on the final-answer and mismatch triggers) drastically reduces the framework's effectiveness. In this variant, the model rarely stops early unless it explicitly says a final answer. Token savings dropped to 18%, and accuracy slightly decreased to 80.4%, meaning the framework intervened too seldom to yield benefits.

Removing the mismatch trigger (while keeping t_{score}) had negligible effect on results (tokens saved remained 31%, accuracy 78.4%). This indicates that the mismatch condition was rarely the deciding factor on MATH500. It serves as a safety net for pathological cases (e.g., when the model's hidden state diverges without obvious repetition in text), but such cases were uncommon in this setting. Removing mismatch changes tokens count by 1% and accuracy by 2%.

We also tested variations of the overthinking score. Using only the similarity and hedge components (omitting RepRatio) reduced token savings to 24%, as the system became less sensitive to subtle content reuse. Using only similarity and repetition (omitting hedge) had little impact on the stop decisions, suggesting the hedge signal is secondary. Overall, the semantic similarity feature was the most critical: without it, the system failed to detect many looping behaviors, underscoring the importance of capturing semantic redundancy.

6 Discussion

What are the underlying reasons behind a successful overthinking mitigation method? Here, we make a mechanistic comparison between our method and the baselines.

Table 2 contrasts our approach with ThoughtTerminator (Pu et al., 2025), DEER (Yang et al., 2025), and s1 (Muennighoff et al., 2025) along key dimensions. ThoughtTerminator imposes a top-down plan by setting a token budget in advance. This framing aligns with prior work on difficulty-aware or budget-calibrated inference for reasoning models (Pu et al., 2025; Shen et al., 2025). Such approaches require either external predictors or engineered prompts to guide generation, and their effectiveness hinges on accurate calibration. In practice, mispredictions can either truncate needed reasoning or allow overthinking to persist, a failure mode explicitly discussed in prior work (Pu et al., 2025; Chen et al., 2025). Our method, by contrast, is bottom-up: it monitors the model's behavior and adapts on the fly without any prior knowledge of problem difficulty.

DEER (Yang et al., 2025) takes a cue-based dynamic exit strategy, prompting for a final answer when certain tokens (like "Wait") appear. This design is conceptually related to earlier dynamic

early-exit mechanisms in neural networks, where intermediate signals trigger inference termination (Xin et al., 2020; Liu et al., 2020). DEER can save many tokens and even avoid some errors by cutting off self-contradictory reasoning. However, DEER depends on the presence of specific lexical cues and uses internal confidence checks (via logits or multiple attempts) to decide whether to accept a trial answer. This reliance on internal model signals and confidence estimation makes faithful implementation challenging in strictly black-box settings. In our experiments, DEER's benefits also diminish in single-shot evaluations (we observed a noticeable accuracy drop on the 8B model), possibly because it lacked the second-chance opportunities permitted in its original evaluation protocol. Our approach does not rely on any particular cue token or multiple answer attempts; instead, it halts when output-derived signals suggest diminishing returns, treating the model as a black box aside from a lightweight logit probe (Nostalgebraist, 2020).

The s1 method involves supervised fine-tuning and special tokens to encourage or limit reasoning depth, placing it within a broader class of test-time scaling approaches that explicitly control computation through training-time interventions (Muennighoff et al., 2025). While this yields explicit control over reasoning length, it comes at the cost of additional training and integration of new tokens into the model's vocabulary and decoding behavior. In contrast, our framework operates entirely at test time and requires no model modifications or retraining.

In summary, dynamic stopping is a lightweight yet flexible solution: it requires no extra training or external models, no task-specific cue tokens, and it generalizes across tasks by leveraging universal signs of repetitiveness and hesitation in the model's own output. This perspective aligns with recent arguments that overthinking is best addressed through online, behavior-aware intervention rather than fixed budgets or static heuristics (Chen et al., 2025; Pu et al., 2025). We view our approach as complementary to methods such as ThoughtTerminator or DEER-one could potentially combine a difficulty-based prior with our online signals for even better performance, which is an exciting avenue for future work.

Method	Extra Training?	External Info	Access Needed
ThoughtTerm.	No	Difficulty predictor	Prompt control
DEER	No	None (cue-based)	Logits (confid.)
s1	Yes (finetune)	None (static)	Special tokens
Ours	No	None	Output only

Table 2: Comparison of early stopping methods. "External Info" denotes reliance on external predictors or cues beyond the model's raw output.

7 Conclusion

We propose a dynamic stopping framework that mitigates overthinking in large language models by monitoring output-derived signals such as semantic repetition, similarity, and hedging, and intervening when continued reasoning becomes unproductive. Across challenging reasoning benchmarks, our method achieves substantial efficiency gains, reducing generated tokens by approximately 30% on the reasoning-specialized DeepSeek-R1-8B model (DeepSeek-AI, 2025) with only minor accuracy loss, and outperforming both static truncation and prior adaptive stopping approaches. Importantly, the framework generalizes to instruction-tuned models: on LLaMA-3.1 8B (Meta AI, 2024), it reduces reasoning tokens by over 70% and can even improve accuracy, suggesting that excessive chain-of-thought generation may be harmful in such settings. By operationally defining overthinking as low-information or repetitive reasoning and detecting it online without modifying model weights, introducing special tokens, or relying on external difficulty predictors, this work demonstrates that dynamic intervention during generation provides a flexible and effective path toward more efficient and controllable reasoning across diverse model families.

8 Limitations and Future Directions

Our framework relies on a small set of interpretable, output-derived indicators (e.g., semantic repetition and hedging) to detect unproductive reasoning. While these signals are heuristic by design, empirical results across multiple datasets (MATH500, GPQA, CSQA, and Misguided Attention) and decoding regimes show that they capture a wide range of overthinking behaviors in practice. We do not claim that these indicators exhaustively characterize all possible failure modes of long-form reasoning.

Although our primary evaluation emphasizes mathematical reasoning, we intentionally include

benchmarks from distinct domains with different sources of uncertainty and structure. The consistent trends observed across scientific QA, commonsense reasoning, and robustness-focused datasets suggest that overthinking is a general phenomenon rather than a domain-specific artifact. Nevertheless, extending the framework to new tasks may benefit from incorporating additional task-aware signals or adapting thresholds based on domain characteristics.

Our method also introduces several stopping-related hyperparameters. Extensive ablation studies indicate that the framework is not brittle and degrades gracefully under parameter perturbations, though optimal settings may vary slightly across models or deployment contexts. Automating or learning these settings remains an important direction.

Looking forward, a key opportunity is deeper **interpretability** of overthinking. While this work focuses on detection and intervention, understanding *why* models overthink and how this behavior emerges internally remains open. More detailed logit-lens analyses could reveal early convergence signals before a final answer is produced, while attention-based analyses may uncover whether overthinking correlates with attention drift or oscillation between prior steps.

We also plan to explore trajectory-based analyses that compare a model's reasoning path against reference minimal solutions. Deviations—measured via embedding similarity or alignment scores—may help identify the onset of unproductive continuation. Relatedly, probabilistic modeling of reasoning-step embeddings could provide softer, uncertainty-aware stopping signals.

Finally, while the current framework is rule-based, future work could investigate learned detectors of overthinking. Lightweight classifiers trained on reasoning traces labeled as "on-track" or "overthinking," potentially using larger models to supervise smaller ones, could offer more flexible and robust stopping decisions.

References

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. 2025. Do not think that much for 2+3=? on the overthinking of o1-like llms. *Preprint*, arXiv:2412.21187.

cpldcpu. 2025. Misguided attention: A collection of prompts to challenge reasoning of large language models. https://github.com/cpldcpu/MisguidedAttention. Accessed: 2025-XX-XX.

DeepSeek-AI. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *NeurIPS*.

Weijie Liu, Peng Zhou, Zhe Zhao, Zhiruo Wang, Haotang Deng, and Qi Ju. 2020. Fastbert: a self-distilling bert with adaptive inference time. *Preprint*, arXiv:2004.02178.

Meta AI. 2024. The llama 3 model family. *arXiv preprint arXiv:2407.21783*.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. 2025. s1: Simple test-time scaling. *Preprint*, arXiv:2501.19393.

Nostalgebraist. 2020. Interpreting gpt: the logit lens. Online forum post on LessWrong. https://www.lesswrong.com/posts/jqXMYk9xEKEu6LGS6/interpreting-gpt-the-logit-lens.

Xiao Pu, Michael Saxon, Wenyue Hua, and William Yang Wang. 2025. Thoughtterminator: Benchmarking, calibrating, and mitigating overthinking in reasoning models. *Preprint*, arXiv:2504.13367.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2023. Gpqa: A graduate-level google-proof qa benchmark. *Preprint*, arXiv:2311.12022.

Yi Shen, Jian Zhang, Jieyun Huang, Shuming Shi, Wenjing Zhang, Jiangze Yan, Ning Wang, Kai Wang, Zhaoxiang Liu, and Shiguo Lian. 2025. Dast: Difficulty-adaptive slow-thinking for large reasoning models. *Preprint*, arXiv:2503.04472.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4149–4158, Minneapolis, Minnesota. Association for Computational Linguistics.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. 2020. Deebert: Dynamic early exiting for accelerating bert inference. *Preprint*, arXiv:2004.12993.

Chenxu Yang, Qingyi Si, Yongjie Duan, Zheliang Zhu, Chenyu Zhu, Qiaowei Li, Minghui Chen, Zheng Lin, and Weiping Wang. 2025. Dynamic early exit in reasoning models. *Preprint*, arXiv:2504.15895.

A Additional Results on Other Datasets

We provide additional evaluation on two reasoning-intensive benchmarks beyond MATH500 to demonstrate the generality of our approach.

Graduate-Level Problem QA (GPQA). GPQA (Rein et al., 2023) consists of challenging multiple-choice questions from graduate-level STEM domains (physics, chemistry, biology, etc.). These questions require multi-step conceptual reasoning rather than straightforward calculation. On GPQA extended, our dynamic stopping framework consistently reduced the length of explanations with only a modest impact on answer accuracy. Specifically, the no-stopping baseline achieved an accuracy of 42.85%, while DeepSeek-R1-8B with our dynamic stopping framework achieved 40.84% accuracy while reducing the average number of generated reasoning tokens by 30.92%. In qualitative analysis, we observed that the framework prevented redundant rephrasing and speculative tangents that the model often produces when uncertain.

We further evaluate our approach on LLaMA-3.1 8B Instruct (Meta AI, 2024), a general-purpose instruction-tuned model. In this setting, the no-stopping baseline achieves an accuracy of 29.67%, reflecting the difficulty of GPQA for non-reasoning-specialized models. Applying dynamic stopping yields a slight accuracy improvement to 30.0% while reducing reasoning tokens by 62%. This result suggests that, for instruction-tuned models, excessive chain-of-thought generation can lead to speculative or low-confidence reasoning, and that truncating such continuation can simultaneously improve efficiency and final answer quality.

Overall, these results indicate that overthinking also arises in scientific question answering, and that output-based signals—particularly semantic repetition and hedging—can effectively identify stalling or unproductive reasoning across both reasoning-specialized and instruction-tuned model families without substantially degrading performance.

CommonsenseQA (CSQA). We additionally evaluate our framework on the CommonsenseQA (Talmor et al., 2019) benchmark, which emphasizes commonsense reasoning rather than symbolic computation. Under near-deterministic decoding, the no-stopping baseline achieves an accuracy of 59.56%, while our dynamic stopping framework attains 58.9% accuracy, with a reduction of 26.76% in generated reasoning tokens. Although this represents a modest accuracy drop, the substantial reduction in reasoning length indicates that overthinking also manifests in commonsense reasoning as redundant or speculative continuation.

Misguided Attention Benchmark. The Misguided Attention dataset (cpldcpu, 2025) tests whether models can maintain focus on relevant information when presented with distracting or misleading context. Models often overthink in this setting by repeatedly attending to irrelevant details, leading to verbose but unproductive chains-of-thought. With our stopping mechanism, we found that high semantic repetition and logit mismatches tended to coincide with these attention misalignments. The dynamic stopping triggered early in such cases, significantly shortening the reasoning traces while preserving the correctness of the final answers. This indicates that our overthinking score captures not only repetition but also the effect of attention drift (as this often leads to loops in the explanation). Overall, on both GPQA and Misguided Attention, our method achieved similar efficiency gains as in MATH500, with no significant drop in performance, highlighting its applicability beyond math problems.

B Wall-Clock Runtime and Implementation Details (Reproducibility)

All runtime measurements are reported for completeness and reproducibility and should be interpreted with care. The reported wall-clock times were obtained using a research prototype implementation that prioritizes clarity and modularity over execution speed. As such, the implementation does not incorporate standard performance optimizations such as embedding caching, batched similarity computation, asynchronous execution, or kernel fusion.

Experiments were conducted on a system with 4 NVIDIA A6000 GPUs, with the 500 MATH500

Setting	Decoding	Acc. (%)	Tokens Saved (%)
Vanilla CoT (no stopping)	Near-det. ($T = 10^{-7}$)	82.6	0.0
Full (Optuna-tuned)	Near-det. ($T = 10^{-7}$)	81.4	30.5
No t_{score} (triggers only)	Near-det. ($T = 10^{-7}$)	81.4	17.55
Full (Optuna-tuned)	Stochastic ($T = 0.6$)	76.2	30.0

Table 3: Ablation study on MATH500 with DeepSeek-R1-8B. "No t_{score}" disables the continuous overthinking score and retains only hard stopping triggers. Tokens Saved are computed relative to Vanilla CoT under the same decoding setting.

problems split into four disjoint batches (120, 130, 120, and 130 problems), each processed independently on a separate GPU. All batches were executed in parallel, and reported runtimes correspond to the wall-clock time of the slowest batch (130 problems), which determines the end-to-end runtime.

Under this setup, the vanilla chain-of-thought baseline required approximately 4 hours per batch, while our dynamic stopping framework required approximately 4 hours and 40 minutes per batch. This difference reflects additional controller-side computations (e.g., online embedding similarity, repetition statistics, and logit probe evaluation) and implementation-level overhead, rather than fundamental increases in model-side inference cost.

We note that absolute wall-clock runtime can vary substantially depending on multiple factors, including GPU utilization, batching strategy, CUDA kernel scheduling, memory contention, system load, and environmental conditions. Consequently, we emphasize token count as the primary and hardware-agnostic measure of inference efficiency, and expect the reported runtime overhead to be significantly reduced in an optimized implementation.

C Overthinking Hedging Lexicon

As part of the dynamic overthinking score (Section 3), we include a lightweight *hedging signal* that captures tentative, self-corrective, or stalling language in the model's generated reasoning. This signal is computed using a small, manually curated lexicon of discourse markers and phrases commonly associated with hesitation, revision, or redundant explanation.

Table 4 lists the hedging words and phrases used in our experiments. The lexicon is not intended to be exhaustive or linguistically precise; rather, it serves as a coarse heuristic that complements repetition- and similarity-based signals. In practice, this component receives a relatively small weight in the overall overthinking score and primarily acts as a secondary indicator of stalled or uncertain reasoning.

Hedging and Overthinking Markers
Classic hedges: but, however, though, although, yet, still, maybe, perhaps, possibly, probably, apparently, seems, appear
Hesitation markers: wait, hold on, actually, in fact, well, hmm, uh, erm, alternatively
Self-doubt and correction: I think, I guess, I suppose, I believe, not sure, unsure, uncertain, let me, let's see, on second thought, reconsider, recalculate, retry, again, sorry, my mistake, correction, instead
Discourse fillers: so, anyway, okay, right, well then, alright, as I was saying
Verbose self-explanations: what I mean, in other words, to clarify, let me explain, let me try again

Table 4: Lexicon of hedging and overthinking-related words and phrases used to compute the Hedge component of the dynamic overthinking score.

D Additional Figures

This section presents supplementary visualizations that provide additional insight into the behavior of our dynamic stopping framework.

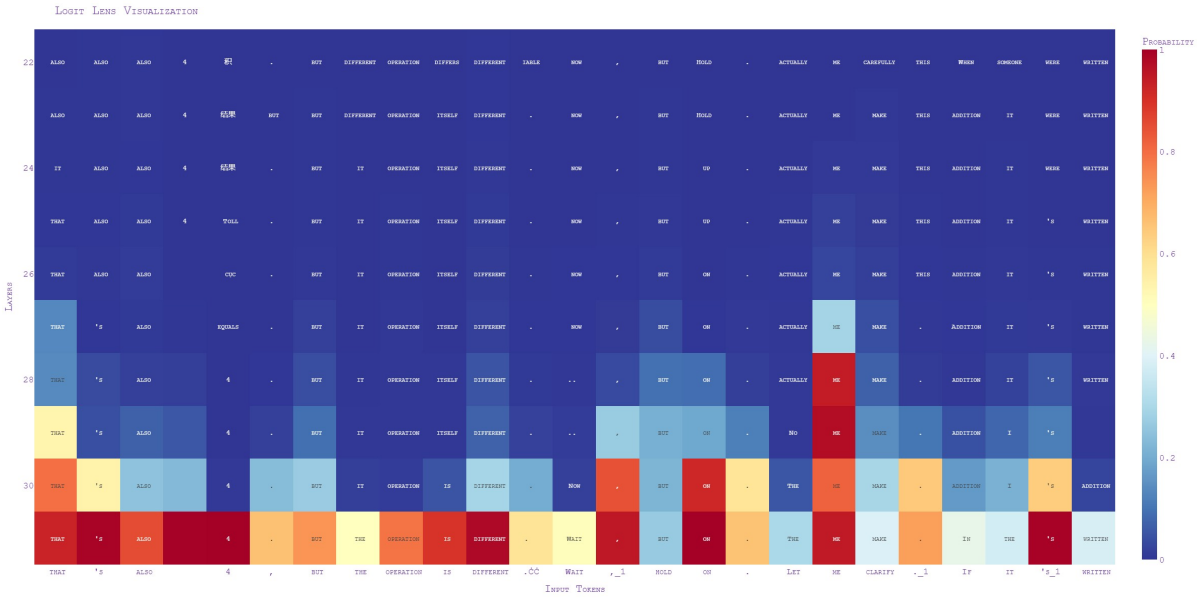

Figure 3: Logit lens heatmap during chain-of-thought generation. Each column corresponds to an output token, and each row corresponds to a transformer layer. Color intensity reflects the layer-wise logit lens probe score (see colorbar). Localized regions of disagreement across layers illustrate mismatch patterns used by our stopping signal.

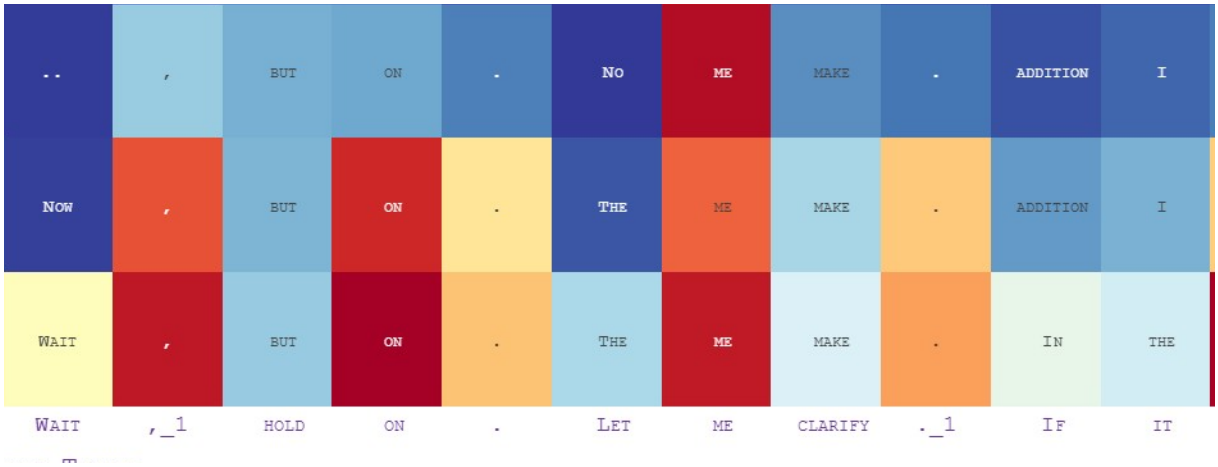

Figure 4: Zoomed-in logit lens heatmap highlighting token-level mismatch. This figure shows a focused subset of Figure 3, centered on a short span of tokens where the final output token does not appear among the top predictions of intermediate transformer layers, illustrating localized internal disagreement monitored by our mismatch-based stopping signal.

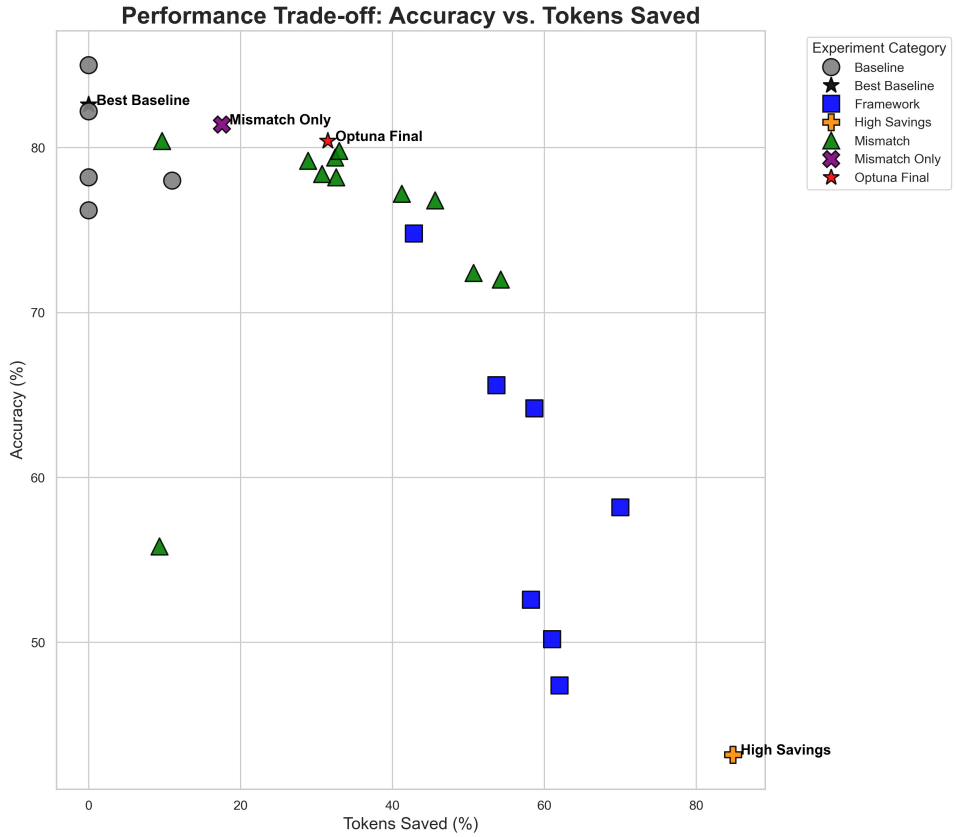

Figure 5: Accuracy vs. tokens saved on MATH500 across stopping strategies. Each point represents a configuration evaluated under a specific stopping mechanism. Gray circles denote baseline runs without dynamic stopping, while colored markers correspond to variants of our framework. The red star marks the Optuna-selected configuration achieving a favorable accuracy–efficiency trade-off.