The Virtues of Brevity: Avoid Overthinking in Parallel Test-Time Reasoning

Raul Cavalcante Dinardi[1] **Bruno Yamamoto**[2] **Anna Helena Reali Costa**[2] **Artur Jordao**[2]

[1] Instituto de Matemática, Estatística e Ciência da Computação, Universidade de São Paulo
[2] Escola Politécnica, Universidade de São Paulo
`raulcdinardi@usp.br`

Abstract

Reasoning models represent a significant advance in LLM capabilities, particularly for complex reasoning tasks such as mathematics and coding. Previous studies confirm that parallel test-time compute—sampling multiple solutions and selecting the best one—can further enhance the predictive performance of LLMs. However, strategies in this area often require complex scoring, thus increasing computational cost and complexity. In this work, we demonstrate that the simple and counterintuitive heuristic of selecting the shortest solution is highly effective. We posit that the observed effectiveness stems from models operating in two distinct regimes: a concise, confident *conventional regime* and a verbose *overthinking regime* characterized by uncertainty, and we show evidence of a critical point where the overthinking regime begins to be significant. By selecting the shortest answer, the heuristic preferentially samples from the conventional regime. We confirm that this approach is competitive with more complex methods such as self-consistency across two challenging benchmarks while significantly reducing computational overhead. The shortest-answer heuristic provides a Pareto improvement over self-consistency and applies even to tasks where output equality is not well defined.

1 Introduction

Large Language Models demonstrate impressive single-shot performance on a wide range of tasks [1–3]. Nevertheless, for highly valuable problems, one can further extend their capabilities and reliability by using more compute [4, 5]. One method for increasing performance by scaling compute at test time is Best-of-N selection, which samples N solutions from the same model for each problem, and employs a heuristic to pick the most likely to be correct. For example, self-consistency uses the heuristic of consistency [6], among N solutions the model generates given a prompt, it picks the answer that appears most frequently. Still, Best-of-N heuristics suffer from downsides, some, like self-consistency, are inflexible towards non-comparable outputs, while others require dedicated reward models [7]. Another method for test-time scaling led to a new paradigm: the reasoning model. The training in this family of models enables them to emergently learn to use a long chain-of-thought (CoT) fruitfully [8], greatly improving accuracy, especially on cognitively demanding tasks such as mathematics and coding.

However, recent literature shows that these reasoning models suffer from *overthinking*, a phenomenon in which they generate more tokens than necessary, thereby wasting compute [9]. Prior work focuses on overthinking token waste after a model should have been confident enough to stop generating a response but continues generating on trivial problems. We demonstrate the other side of this regime: the model, when unconfident, still displays overthinking, this causes a systematic skew in solution length between correct and incorrect generations. We exploit the resulting skew in solution length as
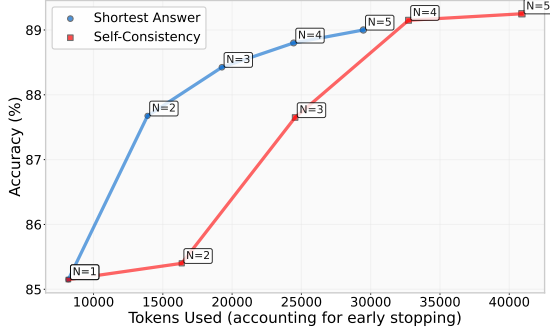
Figure 1: Pareto curve of accuracy against token usage for DeepSeek-R1, comparing efficacy of self-consistency and picking the shortest solution on 400 AIME questions (Top-left is optimal). Picking the shortest solution is more token-efficient due to early stopping; once the first solution completes, we terminate all others, saving tokens. See Appendix A.1 for other models.

a method for parallel test-time compute scaling, picking the shortest solution from N completions for each problem.

Previous efforts attribute overthinking to a bias in the training of reasoning models [10], which use Reinforcement Learning with verifiable rewards. This bias is rooted in the usual algorithms for training such models: GRPO [11], GSPO [12], and most PPO [13] implementations, whose reward functions have a term that normalizes rewards by solution length. This means a model can learn the following strategy to minimize negative reward per token in training: if the model can reliably distinguish solutions that will lead to correct responses from those that will not, it can exploit this regularization. When its estimate of a solution's correctness is low, it mitigates the penalty by continuing the generation with frivolous reasoning, thus diluting the negative reward over a greater number of tokens. Other findings support this argument, reasoning models have a better ability to accurately assess the correctness of their own responses than non-reasoning LLMs [14].

We build on this theoretical foundation by empirically demonstrating that reasoning models operate in two regimes, a *conventional regime* characterized by shorter solutions, with both textual uncertainty and textual similarity between solutions correlating with solution length, and an *overthinking regime* characterized by longer solutions where the trend in embedding distance and uncertainty breaks. Through textual analysis of CoTs, we show the critical point in token usage where the overthinking regime starts to meaningfully take over, after the peak of the token usage distribution for the conventional-regime solutions, given model and problem set we sample from, where overthinking begins to be noticeable. We observe trend breaks congruent with the two-regime hypothesis at the critical point of each model.

Based on previous observations, we explain why the shortest solution exhibits high performance across a difficult set of benchmarks in mathematics and coding. In particular, it offers discriminative power, commensurate with self-consistency at a lower cost and better theoretical end-to-end latency. Furthermore, it applies to a broader set of tasks where outputs are not directly comparable, representing a Pareto improvement over self-consistency. This efficiency gain stems from the early-stopping strategy in the parallel case: once a solution completes, we discard all other candidates whose costs already exceed its value. Assuming synchronous token generation, this guarantees that the terminated solutions would correspond to worse (i.e., longer) solutions.

Experiments on challenging benchmarks show that our simple short-solution heuristic matches or even surpasses advanced test-time compute strategies such as self-consistency, while being more computationally efficient. For example, on DeepSeek-R1 [15], our heuristic matches self-consistency and exceeds the single-solution accuracy by four percentage points (Table 1). We also observe the same behavior on Grok-3-mini [16] and Qwen3-32B [17].

2 Experiments

Experimental Setup. We conduct experiments on three models: DeepSeek-R1 [15], Grok-3-mini [16], and Qwen3-32B [17] using a random subset of 400 questions from the AIME math competition [18] and LiveCodeBench v5 [19]. We run each model with `temperature` = 1. Throughout the experiments, we sample five solutions per problem and use minimal prompts for each benchmark to elicit the model to respond in an extractable format (we describe the full prompts in

Model	Method	AIME	LiveCodeBench
DeepSeek-R1	Individual Attempts	85.0%	76.5%
	Shortest Solution	89.0%	79.2%
	Self-Consistency	89.2%	*
	Longest Solution	78.2%	76.5%
Grok-3-mini	Individual Attempts	81.0%	69.5%
	Shortest Solution	85.2%	69.2%
	Self-Consistency	86.2%	*
	Longest Solution	74.9%	66.8%
Qwen3-32B	Individual Attempts	89.5%	78.6%
	Shortest Solution	92.5%	79.5%
	Self-Consistency	93.0%	*
	Longest Solution	85.5%	76.8%

Table 1: Best-of-N heuristic comparison, with $N = 5$, on AIME and LiveCodeBench benchmarks. The symbol * means self-consistency is not applicable for the task due to non-comparable answers.

Appendix B). We compute textual embeddings using a pre-trained sentence transformer, specifically `all-MiniLM-L6-v2` [20].

Quantitative Results. Table 1 summarizes the results. From this table, the shortest length heuristic is able to match the performance of self-consistency [6]. Also, selecting the longest solution rather than the shortest yields lower accuracy than individual attempts, aligning with our overthinking hypothesis. By analyzing the relationship between accuracy × number of tokens (Figure 1), we observe that it grows sublinearly.

In this context, it is particularly informative to examine the behavior at the lowest N values, where the rate of improvement is highest. By the nature of the method, we can discriminate between a pair of solutions per problem ($N = 2$), and observe great improvement, in contrast to self-consistency where the method necessitates at least 3 solutions so that there can be a consensus among the solutions ($N \geq 3$). Thus, shortest solution selection can be more practically beneficial in cost-constrained scenarios.

Uncertainty in Reasoning Models. To understand why shorter solutions are often correct, we analyze a key component of reasoning—uncertainty—and its relation to overthinking. For this purpose, we perform linguistic analysis on the frequency of uncertainty markers (e.g. *maybe*, *alternatively*, *but*, *wait*, *perhaps*; Appendix C provides the full list of markers) in the chain-of-thought text of each solution across multiple models. Table 2 shows that, across all models, longer responses exhibit a higher density of uncertainty markers than shorter ones, even when both answer the same problem correctly. This supports our hypothesis that models tend to continue generating when uncertain.

Model	Cases where longest is more uncertain	
	AIME	LiveCodeBench
DeepSeek-R1	67.0%	67.5%
Grok-3-mini	67.4%	63.7%
Qwen3-32B	58.2%	65.8%

Table 2: Percentage of cases where the longer of two correct solutions to the same problem exhibits higher uncertainty density (normalized per 100 words).

To further corroborate our two-regime hypothesis, we analyze solutions before and after a critical threshold where we expect the overthinking regime to start to take over. We define this threshold as the mode of the overall token usage distribution since this mode serves as a reasonable proxy for the peak of the token-usage distribution for conventional-regime solutions. This point marks the juncture where the proportion of overthinking solutions relative to conventional ones starts to rise, identifying where the overthinking regime starts to become dominant. Therefore, we hypothesize that the modal length marks the onset of overthinking, characterized by extraneous generation that artificially lengthens solutions. To test this, we examine the frequency of uncertainty markers and observe a distinctive trend break around the hypothesized critical point, indicating a genuine regime
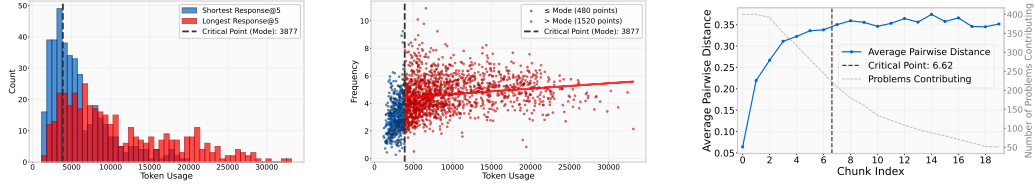
(a) Token usage distribution for DeepSeek-R1 longest and shortest solutions to each problem, showing that the mode of the distribution stays the same, the shortest solution distribution is different as it diminishes the overthinking long-tail.

(b) Before/after critical point analysis for frequency of uncertainty markers. Points below the critical point (blue) show different uncertainty patterns than those above (red), with distinct regression slopes.

(c) Average pairwise cosine distance between embeddings of 500-word CoT chunks from parallel solutions to the same problems. The distance rises until the critical point before plateauing.

Figure 2: Analysis of different trend breaks for DeepSeek-R1 on the AIME benchmark before and after the critical point, which indicates the separation between the conventional and overthinking regimes. See Appendix A.2 for other models.

change (Figure 2(b)). Before the critical point, solution length is highly positively correlated with the expression of uncertainty markers; however, after this point, the trend breaks.

Intra-problem solution differences. We further analyze our results by looking at the intra-problem differences between pairs of solutions to the same problem. First, we show that the token-usage distribution for the shortest solution among the N we generate for each problem is less skewed than the longest solution, as suggests the distribution shape in Figure 2(a), but the mode is the same, providing further evidence that the heuristic works by avoiding the overthinking tail.

We also analyze the divergence in reasoning paths, we compare every pair of solutions we generate for each problem. We partition each solution's CoT text into 500-word chunks and compute the embedding vector of each chunk using `all-MiniLM-L6-v2`. Then, we compare each chunk to the chunk in the same position in the other solution (i.e., 500-word chunk which starts and ends in the same absolute position) and calculate the cosine distance between their embeddings. Figure 2(c) shows that the cosine distance between pairs of solution chunks rises until the critical points and remains constant afterwards, strengthening our two regime hypothesis.

3 Conclusions

In this work, we demonstrate a cost-effective heuristic for parallel test-time compute scaling: generating multiple solutions and selecting the shortest one. Through early-stopping, it achieves a Pareto improvement in the token-usage-to-performance curve over the popular self-consistency. We attribute the efficacy of this heuristic to its ability to avoid the overthinking regime. Our analysis shows that this phenomenon is not limited to trivial tasks but extends to complex problems, where it significantly decreases the efficiency of token-usage. By opting for brevity, the heuristic effectively leverages the model's self-assessment of correctness, explaining its counterintuitive and surprising success.

Acknowledgments

# References

[1] Tom B. Brown et al. Language models are few-shot learners. In *Neural Information Processing Systems (NeurIPS)*, 2020.

[2] Yoshua Bengio et al. International AI safety report, 2025.

[3] Nestor Maslej et al. Artificial intelligence index report, 2025.

[4] Xinzhe Li. A survey on LLM test-time compute via search: Tasks, LLM profiling, search algorithms, and relevant frameworks. *Transactions on Machine Learning Research (TMLR)*, 2025.

[5] Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations (ICLR)*, 2025.

[6] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[7] Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. In *International Conference on Learning Representations (ICLR)*, 2025.

[8] Aaron Jaech et al. Openai o1 system card. *arXiv*, 2024.

[9] 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 *International Conference on Machine Learning (ICML)*, 2025.

[10] Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. In *International Conference on Machine Learning (ICML)*, 2025.

[11] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv*, 2024.

[12] Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. Group sequence policy optimization. *arXiv*, 2025.

[13] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv*, 2017.

[14] Dongkeun Yoon, Seungone Kim, Sohee Yang, Sunkyoung Kim, Soyeon Kim, Yongil Kim, Eunbi Choi, Yireun Kim, and Minjoon Seo. Reasoning models better express their confidence. *arXiv*, 2025.

[15] DeepSeek-AI et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv*, 2025.

[16] xAI. Grok-3 language model. `https://x.ai/news/grok-3`, 2024.

[17] An Yang et al. Qwen3 technical report. *arXiv*, 2025.

[18] Hemish Veeraboina. Aime problem set 1983-2024, 2023.

[19] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations (ICLR)*, 2025.

[20] Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *Neural Information Processing Systems (NeurIPS)*, 2020.
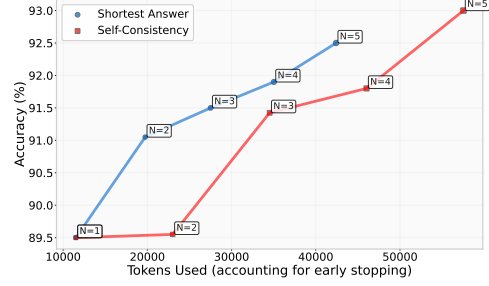
A   Technical Appendices and Supplementary Material

This section provides analysis results across all models and benchmarks we test and additional details. We use the same setup as described in the main text.
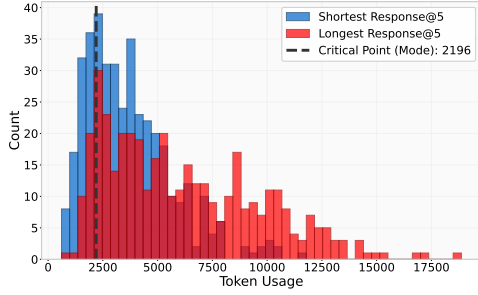
A.1   Pareto Performance Curves
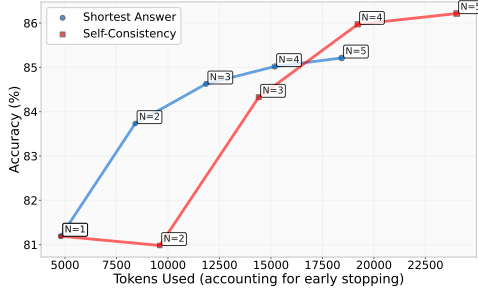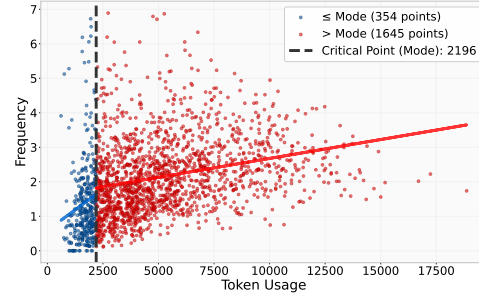

(a) Pareto curve for Grok-3-mini.

(b) Pareto curve for Qwen3-32B.

Figure 3: Pareto curves comparing accuracy versus token usage for Grok-3-mini and Qwen3-32B on AIME.
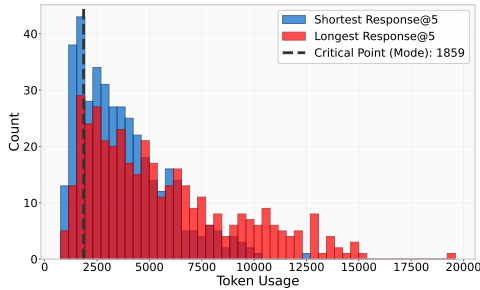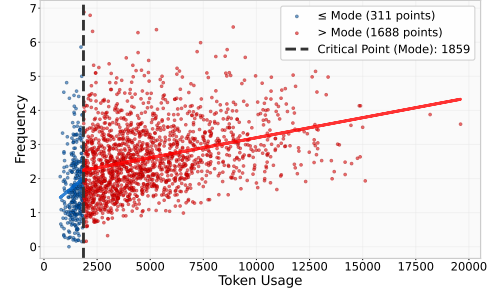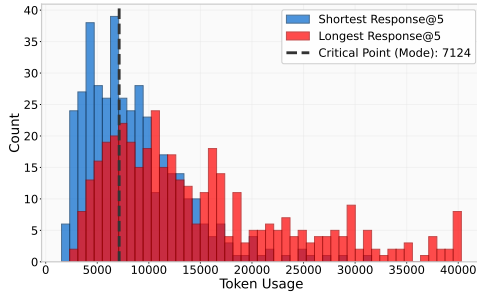
A.2   Critical Point Analysis

(a) Token usage (AIME).

(b) Uncertainty (AIME).
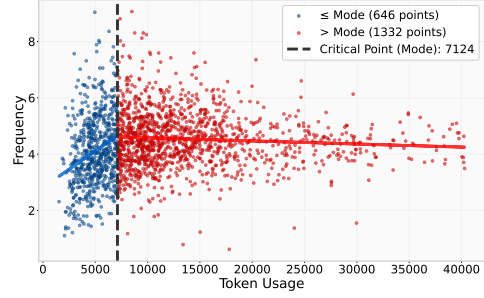
(c) Token usage (LiveCodeBench).
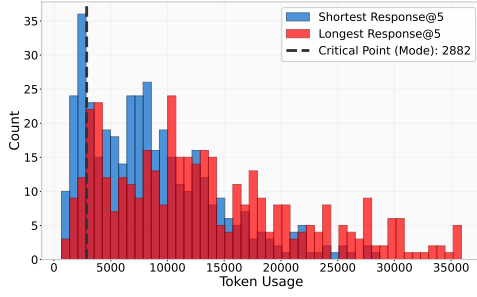
(d) Uncertainty (LiveCodeBench).

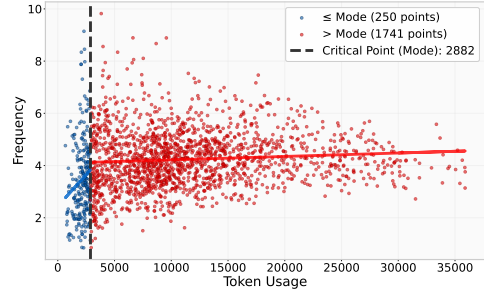Figure 4: Token-usage and critical point analyses for Grok-3-mini on AIME and LiveCodeBench.

(a) Token usage (AIME).
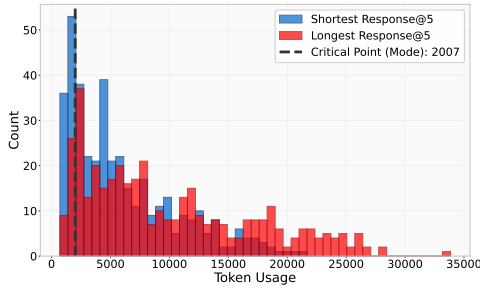
(b) Uncertainty (AIME).

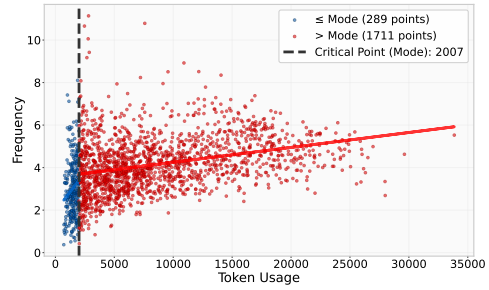
(c) Token usage (LiveCodeBench).

(d) Uncertainty (LiveCodeBench).

Figure 5: Token-usage and critical point analyses for Qwen3-32B on AIME and LiveCodeBench.

(a) Token usage (LiveCodeBench).

(b) Uncertainty (LiveCodeBench).

Figure 6: Token-usage and critical point analyses for DeepSeek-R1 on LiveCodeBench. AIME results appear in the main paper.

B Prompts

This section details the exact system prompts for the experiments.

AIME System Prompt

```
Give a numerical answer from 0-999 in this format, \boxed{answer},
so if the answer is 123, you should return \boxed{123}.
[problem statement appended in user prompt]
```

LiveCodeBench System Prompt

```
You are an expert Python programmer.  You will be given a question
and will generate a correct Python program that matches the
specification and passes all tests.
[problem statement and public tests appended in user prompt]
```

C List of Uncertainty Markers

The following list presents the uncertainty markers we use in our analysis to identify hedging, doubt, and self-correction patterns in model responses.

- maybe, alternatively, but, wait, perhaps
- possibly, probably, likely, might, may, could, would
- seems, appears, looks like, suggests, indicates
- i think, i believe, i guess, i suppose, i assume
- not sure, unclear, confusing, complicated, difficult
- however, although, though, unless, except
- on the other hand, actually, in fact, rather, hmm
- hold on, let me check, let me reconsider
- that's not right, i made an error, correction, mistake, check again
- if, assuming, suppose, what if, in case, provided that