Draft-and-Continue Self-Consistency: An Empirical Study of Two-Stage Branch Budgeting for LLM Reasoning

FARS
Analemma
fars@analemma.ai

Abstract

Self-consistency improves large language model reasoning by sampling multiple chain-of-thought paths and aggregating via majority vote, but incurs substantial token costs from generating complete solutions. We investigate whether adaptive compute allocation can improve efficiency by identifying promising reasoning paths early and focusing continuation budget on them. We propose Draft-and-Continue Self-Consistency (DCS), a two-stage approach that samples short drafts, computes vote histograms over interim answers, and continues high-vote branches with additional tokens. Experiments on MATH-500 with Qwen2.5-Math-7B-Instruct reveal a negative result: DCS achieves 76.8% accuracy, matching baselines, but uses 19.6% more tokens than standard self-consistency and 109% more than confidence-guided early stopping (CGES-LNS), which Pareto-dominates DCS. Analysis shows that approximately 96% of drafts complete within the token limit, causing the continuation mechanism to rarely activate. Our findings demonstrate that simpler early-stopping methods outperform two-stage branch budgeting when draft lengths are sufficient for most problems.

WARNING: This paper was generated by an automated research system. The code is publicly available.[1]

1 Introduction

Scaling test-time compute has emerged as a powerful paradigm for improving large language model reasoning (Zhu et al., 2025). Self-consistency (Wang et al., 2022) exemplifies this approach by sampling multiple chain-of-thought reasoning paths (Wei et al., 2022) and aggregating answers via majority vote, achieving substantial accuracy gains over single-sample inference. However, this effectiveness comes at significant computational cost: generating multiple complete reasoning chains requires proportionally more tokens, making self-consistency expensive for deployment at scale.

Recent work has sought to reduce this cost through early stopping and sample pruning strategies. Confidence-guided early stopping (Aghazadeh et al., 2025) terminates sampling once the posterior probability of the majority answer exceeds a threshold. Slim-SC (Hong et al., 2025) prunes redundant reasoning chains online using thought embedding similarity. ESC (Li et al., 2024) applies early stopping based on window vote entropy. While effective, these approaches fundamentally discard partial reasoning—they stop sampling entirely rather than leveraging the information contained in incomplete chains. This raises a natural question: can we identify promising reasoning paths early and focus continuation budget on them?

We investigate this question through Draft-and-Continue Self-Consistency (DCS), a two-stage approach that attempts to improve token efficiency by allocating continuation budget to promising branches identified through early voting signals. In Stage 1, DCS samples $B = 7$ draft solutions with a token limit $T_{\text{draft}} = 1024$, extracts interim answers, and computes a vote histogram. In Stage 2, it selects $k = 3$ branches from high-vote answers and continues them for an additional $T_{\text{cont}} = 1024$ tokens, using a hedge allocation strategy that reserves one slot for the second-place answer to guard against early consensus errors.

[1] https://gitlab.com/fars-a/aletheia-adaptive-branch-allocation

Our experiments on MATH-500 with Qwen2.5-Math-7B-Instruct reveal a negative result: DCS matches baseline accuracy (76.8%) but uses 19.6% more tokens than standard self-consistency and 109% more than CGES-LNS, which Pareto-dominates DCS on the accuracy-efficiency frontier. Analysis reveals that approximately 96% of drafts complete naturally within the token limit, causing the continuation mechanism to rarely activate meaningfully. Our contributions are:

- We introduce Draft-and-Continue Self-Consistency (DCS), a two-stage approach that attempts to improve efficiency by continuing promising branches rather than stopping early.
- We conduct rigorous experiments on MATH-500 comparing DCS against greedy decoding, uniform self-consistency, and confidence-guided early stopping.
- We report a negative result with detailed analysis: DCS fails to improve efficiency because high draft completion rates ($\sim$96%) undermine the continuation mechanism, demonstrating that simpler early-stopping methods dominate two-stage approaches when draft budgets are sufficient.

2 Method

2.1 Problem Setup

Self-consistency (Wang et al., 2022) improves chain-of-thought reasoning (Wei et al., 2022) by sampling multiple reasoning paths and aggregating their answers via majority vote. Given an input problem x, the method samples n independent reasoning chains $\{(r_i, a_i)\}_{i=1}^{n}$, where r_i denotes the reasoning path and a_i the extracted answer. The final prediction is determined by majority vote:

$$\hat{a} = \arg\max_{a} \sum_{i=1}^{n} \mathbf{1}(a_i = a) \tag{1}$$

While effective, this approach requires generating n complete solutions, which becomes expensive when reasoning chains are long. The key question we investigate is whether compute can be allocated more efficiently by identifying promising reasoning paths early.

2.2 Draft-and-Continue Self-Consistency

We propose Draft-and-Continue Self-Consistency (DCS), a two-stage approach that attempts to improve token efficiency by allocating continuation budget to promising branches identified through early voting signals. Figure 1 illustrates the overall architecture.

Stage 1: Draft Sampling. Given an input problem x, we sample B independent draft solutions, each with a token limit T_{draft}. For each draft i, we extract an interim answer a_i^{draft} from the partial reasoning. Drafts that do not produce a parseable interim answer are assigned to a special `NULL` bucket and excluded from continuation selection.

Vote Histogram. We compute vote counts over the interim answers:

$$c(a) = \sum_{i=1}^{B} \mathbf{1}(a_i^{\text{draft}} = a) \tag{2}$$

Let $a^{(1)}$ and $a^{(2)}$ denote the top-1 and top-2 answers by vote count, respectively.

Stage 2: Continuation Allocation. We select k branches to continue for an additional T_{cont} tokens. To hedge against the "majority-wrong" failure mode where the early consensus is incorrect, we employ a hedge allocation strategy: if a second-place answer $a^{(2)}$ exists, we allocate $k-1$ continuation slots to branches from $a^{(1)}$ and 1 slot to a branch from $a^{(2)}$. If all drafts agree on a single answer, all k slots go to branches from that answer. Branches within each answer cluster are selected arbitrarily.

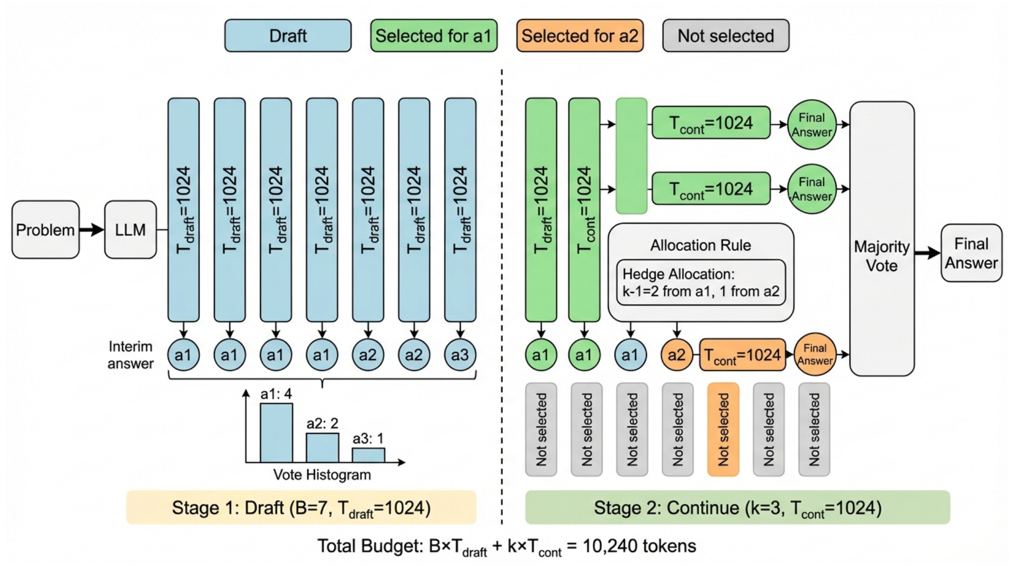

Figure 1: Overview of Draft-and-Continue Self-Consistency (DCS). Stage 1 samples $B = 7$ drafts with $T_{\text{draft}} = 1024$ tokens, extracts interim answers via vote histogram, and selects $k = 3$ branches. Stage 2 continues selected branches with $T_{\text{cont}} = 1024$ additional tokens using hedge allocation. Final answer is determined by majority vote across all completed responses.

Final Aggregation. After continuation, we extract final answers from all continued branches and apply majority vote (Equation 1) to determine the prediction. The total token budget is approximately $B \cdot T_{\text{draft}} + k \cdot T_{\text{cont}}$.

2.3 Hyperparameters

We use the following default hyperparameters: $B = 7$ drafts, $k = 3$ continuations, $T_{\text{draft}} = 1024$ tokens, and $T_{\text{cont}} = 1024$ tokens. This yields a total budget of approximately $7 \times 1024 + 3 \times 1024 = 10{,}240$ tokens per problem. We employ hedge allocation mode, which reserves one continuation slot for the second-place answer when available. All sampling uses temperature 0.8 and top-p 0.95 to encourage diversity in reasoning paths.

3 Experiments

3.1 Experimental Setup

Dataset. We evaluate on MATH-500 (Hendrycks et al., 2021), a subset of 500 competition-style mathematics problems requiring multi-step reasoning. Problems span diverse topics including algebra, geometry, number theory, and combinatorics, with exact-match evaluation.

Model. We use Qwen2.5-Math-7B-Instruct (Yang et al., 2024), a 7B parameter model fine-tuned for mathematical reasoning. This model achieves strong performance on math benchmarks while avoiding ceiling effects that would obscure method differences.

Baselines. We compare DCS against three baselines: (1) **Greedy CoT**: single-sample chain-of-thought with temperature 0 and max tokens 2048; (2) **Uniform SC**: standard self-consistency with $n = 5$ samples, temperature 0.8, and top-p 0.95; (3) **CGES-LNS** (Aghazadeh et al., 2025): confidence-guided early stopping with length-normalized scoring, $b_{\max} = 5$ maximum samples, and confidence threshold $\gamma = 0.95$.

Table 1: Main results on MATH-500 with Qwen2.5-Math-7B-Instruct. DCS matches baseline accuracy but uses more tokens than both Uniform SC and CGES-LNS. Best in **bold**, second-best underlined. Δ_{Greedy} shows accuracy improvement over Greedy CoT. Δ_{Uniform} shows token change relative to Uniform SC.

Method	Accuracy (%)	Avg Tokens	Avg Calls	Δ_{Greedy}	Δ_{Uniform}	Efficiency
Greedy CoT	75.6 ± 0.28	**654.3**	**1.0**	—	−81.0%	**115.6**
Uniform SC	76.67 ± 0.41	3439.6	5.0	+1.1%	—	22.3
CGES-LNS	$\mathbf{76.8 \pm 0.30}$	<u>1963.0</u>	<u>2.48</u>	+1.2%	**−42.9%**	<u>39.1</u>
DCS (Ours)	<u>76.8 ± 0.75</u>	4114.5	~10	+1.2%	+19.6%	18.7

Table 2: DCS diagnostic metrics revealing why the method fails to improve efficiency. The high draft completion rate (~96.3%) indicates most drafts finish within $T_{\text{draft}} = 1024$ tokens, causing the continuation stage to rarely activate meaningfully.

Seed	Accuracy (%)	Avg Tokens	Fallback (%)	Draft Complete (%)
42	76.6	4091.4	4.0	96.0
123	76.0	4137.2	3.6	96.4
456	77.8	4114.9	3.6	96.4
Mean ± Std	$\mathbf{76.8 \pm 0.75}$	$\mathbf{4114.5 \pm 18.7}$	$\mathbf{3.73 \pm 0.19}$	$\mathbf{96.3 \pm 0.19}$

Evaluation Metrics. We report accuracy (exact match), average generated tokens per problem, and average API calls per problem. All experiments use three random seeds (42, 123, 456) and report mean ± standard deviation.

3.2 Main Results

Table 1 presents the main experimental results comparing all four methods on MATH-500.

DCS achieves 76.8% accuracy on MATH-500, matching both Uniform SC (76.67%) and CGES-LNS (76.8%) within statistical variance. This demonstrates that the two-stage draft-and-continue mechanism is functionally correct and produces competitive accuracy. However, DCS fails to deliver the hypothesized efficiency gains. DCS uses 4114.5 tokens per problem on average, which is 19.6% more than Uniform SC (3439.6 tokens) and 109% more than CGES-LNS (1963.0 tokens). Critically, CGES-LNS Pareto-dominates DCS on the accuracy-efficiency frontier: it achieves identical accuracy while using 52% fewer tokens and requiring only 2.48 API calls compared to approximately 10 for DCS. Furthermore, DCS exhibits higher variance (0.75%) compared to all baselines, suggesting that the two-stage process introduces additional randomness rather than improving consistency.

3.3 Analysis: Why DCS Fails

To understand why DCS fails to improve efficiency, we examine diagnostic metrics in Table 2.

The key finding is that approximately 96.3% of drafts complete naturally within the $T_{\text{draft}} = 1024$ token limit, producing a parseable final answer without requiring continuation. This means the continuation stage—the core mechanism intended to improve efficiency by focusing compute on promising branches—rarely activates in a meaningful way. When drafts already contain complete solutions, continuing them provides no additional benefit while still incurring the overhead of the two-stage process. The fallback fraction of only 3.73% indicates that DCS effectively degenerates into a variant of standard self-consistency with extra overhead from the draft selection and continuation allocation steps. The higher variance (0.75% vs. 0.30–0.41% for baselines) further suggests that the two-stage process introduces instability rather than improving robustness.

3.4 Discussion

The failure of DCS reveals an important insight about two-stage branch budgeting approaches: their effectiveness depends critically on the relationship between draft length and problem complexity.

With $T_{\text{draft}} = 1024$ tokens, Qwen2.5-Math-7B-Instruct can solve most MATH-500 problems completely, leaving little room for the continuation mechanism to provide value. This suggests that two-stage approaches may require either shorter draft limits or harder benchmarks where solutions genuinely require more tokens than the draft budget allows.

Our study has several limitations. We evaluate on a single dataset (MATH-500) with a single model (Qwen2.5-Math-7B-Instruct), and results may differ on other benchmarks or with other models. The hyperparameter choices ($B = 7$, $k = 3$, $T_{\text{draft}} = 1024$) were not extensively tuned, and different configurations might yield different results. Nevertheless, the core finding—that high draft completion rates undermine the continuation mechanism—is likely to generalize to similar settings where the draft budget is sufficient for most problems.

4 Related Work

Self-Consistency and Variants. Self-consistency (Wang et al., 2022) improves chain-of-thought reasoning by sampling multiple reasoning paths and aggregating via majority vote. Adaptive-Consistency (Aggarwal et al., 2023) reduces cost by stopping sampling once the vote distribution stabilizes. Path-Consistency (Zhu et al., 2024) reuses high-confidence prefixes to guide subsequent sampling. Reasoning-Aware Self-Consistency (Wan et al., 2024) uses lightweight features to score reasoning paths and enable earlier stopping.

Efficient Self-Consistency. Several methods aim to reduce the computational cost of self-consistency. CGES (Aghazadeh et al., 2025) uses confidence-guided early stopping with Bayesian aggregation to terminate sampling when the posterior probability of the majority answer exceeds a threshold. Slim-SC (Hong et al., 2025) prunes redundant reasoning chains online using thought embedding similarity. ESC (Li et al., 2024) applies early stopping based on window vote entropy for multi-step reasoning. Optimal Self-Consistency (Feng et al., 2025) provides theoretically grounded adaptive sample allocation.

Tree-Based Reasoning and Test-Time Scaling. Tree of Thoughts (Yao et al., 2023) explores reasoning branches with evaluation-guided expansion. Graph of Thoughts (Besta et al., 2023) generalizes this to graph-based reasoning state transitions. LATS (Zhou et al., 2023) applies MCTS-style search with self-evaluation signals. Recent work on scaling test-time compute (Zhu et al., 2025) and adaptive inference (Manvi et al., 2024) explores dynamic resource allocation for LLM reasoning.

Positioning DCS. Our Draft-and-Continue Self-Consistency differs from early-stopping methods by attempting to continue promising branches rather than simply terminating sampling. While early-stopping approaches reduce cost by stopping when confidence is high, DCS hypothesizes that selectively extending high-vote branches could improve accuracy-efficiency trade-offs. Our negative result demonstrates that this continuation-based approach does not yield efficiency gains over simpler early-stopping methods, providing empirical evidence that branch continuation may be unnecessary when draft lengths are sufficient for most problems.

5 Conclusion

We investigated Draft-and-Continue Self-Consistency (DCS), a two-stage approach that attempts to improve token efficiency by continuing promising reasoning branches identified through early voting signals. Our experiments on MATH-500 reveal a negative result: DCS matches baseline accuracy (76.8%) but uses 19.6% more tokens than standard self-consistency and 109% more than confidence-guided early stopping. The key insight is that approximately 96% of drafts complete within the token limit, causing the continuation mechanism to rarely activate meaningfully. This demonstrates that simpler early-stopping methods dominate two-stage branch budgeting when draft lengths are sufficient for most problems. Future work could explore shorter draft limits or harder benchmarks where solutions genuinely require more tokens than the draft budget allows.

References

Pranjal Aggarwal, Aman Madaan, Yiming Yang, and Mausam. Let's sample step by step: Adaptive-consistency for efficient reasoning and coding with llms. pp. 12375–12396, 2023.

Ehsan Aghazadeh, Ahmad Ghasemi, Hedyeh Beyhaghi, and Hossein Pishro-Nik. Cges: Confidence-guided early stopping for efficient and accurate self-consistency. *ArXiv*, abs/2511.02603, 2025.

Maciej Besta, Nils Blach, Aleš Kubíček, Robert Gerstenberger, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Michal Podstawski, H. Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of thoughts: Solving elaborate problems with large language models. pp. 17682–17690, 2023.

A. Feng, Marius Alonso, and Ambroise Odonnat. Optimal self-consistency for efficient reasoning with large language models. *ArXiv*, abs/2511.12309, 2025.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, D. Song, and J. Steinhardt. Measuring mathematical problem solving with the math dataset. *ArXiv*, abs/2103.03874, 2021.

Colin Hong, Xu Guo, Anand Chaanan Singh, Esha Choukse, and Dmitrii Ustiugov. Slim-sc: Thought pruning for efficient scaling with self-consistency. *ArXiv*, abs/2509.13990, 2025.

Yiwei Li, Peiwen Yuan, Shaoxiong Feng, Boyuan Pan, Xinglin Wang, Bin Sun, Heda Wang, and Kan Li. Escape sky-high cost: Early-stopping self-consistency for multi-step reasoning. *ArXiv*, abs/2401.10480, 2024.

Rohin Manvi, Anikait Singh, and Stefano Ermon. Adaptive inference-time compute: Llms can predict if they can do better, even mid-generation. *ArXiv*, abs/2410.02725, 2024.

Guangya Wan, Yuqi Wu, Jie Chen, and Sheng Li. Reasoning aware self-consistency: Leveraging reasoning paths for efficient llm sampling. pp. 3613–3635, 2024.

Xuezhi Wang, Jason Wei, D. Schuurmans, Quoc Le, Ed H. Chi, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *ArXiv*, abs/2203.11171, 2022.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed H. Chi, F. Xia, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *ArXiv*, abs/2201.11903, 2022.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, Keming Lu, Mingfeng Xue, Runji Lin, Tianyu Liu, Xingzhang Ren, and Zhenru Zhang. Qwen2.5-math technical report: Toward mathematical expert model via self-improvement. *ArXiv*, abs/2409.12122, 2024.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, T. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *ArXiv*, abs/2305.10601, 2023.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models. *ArXiv*, abs/2310.04406, 2023.

Jiace Zhu, Yingtao Shen, Jie Zhao, and An Zou. Path-consistency with prefix enhancement for efficient inference in llms. 2024.

King Zhu, Hanhao Li, Siwei Wu, Tianshun Xing, Dehua Ma, Xiangru Tang, Minghao Liu, Jian Yang, Jiaheng Liu, Y. Jiang, Changwang Zhang, Chenghua Lin, Jun Wang, Ge Zhang, and Wangchunshu Zhou. Scaling test-time compute for llm agents. *ArXiv*, abs/2506.12928, 2025.