ε-LEAF ENUMERATION: NON-REPEATING SELF-CONSISTENCY VIA TRUNCATED TREE SEARCH

Xueyan Li[1,2] **Johannes Zenn**[1,3,4,6] **Ekaterina Fadeeva**[2] **Guinan Su**[1,3]
Mrinmaya Sachan[2] **Jonas Geiping**[1,3,5]

[1]Max Planck Institute for Intelligent Systems [2]ETH Zurich [3]AI Center Tübingen
[4]University of Tübingen [5]ELLIS Institute Tübingen [6]IMPRS-IS

ABSTRACT

Self-consistency boosts inference-time performance by sampling multiple reasoning traces in parallel and voting. However, in constrained domains like math and code, this strategy is compute-inefficient because it samples with replacement, repeatedly generating the same early prefixes. We show that this can be fully replaced by a deterministic enumeration of completions. While enumerating exponentially many completions would become infeasible for standard samplers, we show that, when the search space is truncated to tokens above a probability threshold, partial enumeration of the most likely or the most diverse branches is significantly more efficient. We propose ε-Leaf Enumeration (ELE), a deterministic tree-search method that imposes a strict non-repetition condition, covering the search space more optimally than via sampling. ELE naturally enables prefix reuse by caching shared prefixes across branches, reducing redundant computation. Empirically, at equal token budgets, ELE explores more distinct completions than self-consistency, which translates into consistent improvements in math, coding and general reasoning tasks through efficient parallel reasoning.

1 INTRODUCTION

Parallel test-time computation has emerged as a principled way of scaling the performance of large language models (LLMs) (Snell et al., 2024; Lin et al., 2024). In domains where voting is possible, such as math or code, self-consistency improves reasoning performance at inference time (Wang et al., 2023). By sampling multiple solution traces and aggregating them via majority voting or answer-weighted selection (Wang et al., 2024a; Taubenfeld et al., 2025), test-time compute can be leveraged for accuracy gains. However, naive stochastic sampling, while parallel, is compute-inefficient. For domains like math and code where problems are solved by narrow sets of solutions, similar reasoning steps are repeatedly generated (Zhu et al., 2024; Gao et al., 2025; Hong et al., 2025).

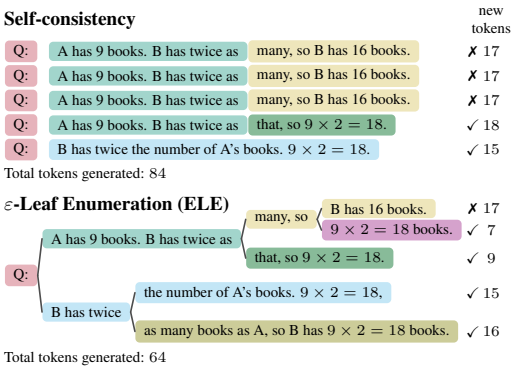


Figure 1: **ε-leaf enumeration (ELE) avoids recomputing the same prefixes, and explores the generation space systematically.** See Section 1.

Figure 1 **(top)** shows this redundancy. For code generation, 20% of prefixes repeat across generations (Figure 10). Thus, in the corresponding tree structure, early branches are repeatedly traversed, while the number of possible continuations grows exponentially with each token requiring efficient search.

Search methods such as beam search (Lowerre, 1976; Sutskever et al., 2014) make this tree structure explicit and control the traversed range with their number of branches (beams). However, beam search leads to degenerate repetitions (Vijayakumar et al., 2016; Cohen & Beck, 2019), quality deterioration (Koehn & Knowles, 2017; Yang et al., 2018), and reduced diversity (Su et al., 2022).

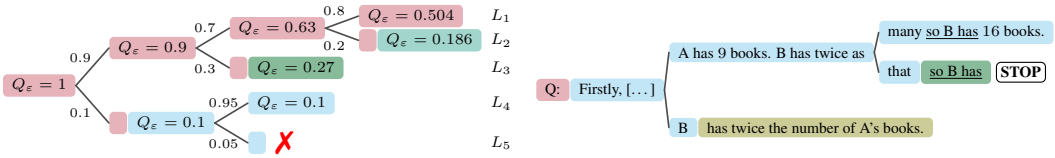


Figure 2: **Left**: **Branching variations of ELE with $\varepsilon = 0.1$.** PROBFIRST: most likely branches are explored as L_1, L_3, L_2, L_4. DIVFIRST: earliest branches are explored as L_1, L_4, L_3, L_2. Branch $L_5 < \varepsilon$ is not explored. Colors distinguish sampling rollouts. **Right**: **Early stopping for ELE.** Illustration uses $n = 3$ repeated tokens as early stopping condition. When the middle branch is generated, early stopping halts generation due to the tokens "so B has" being repeated. See Section 2.

In this work, we argue that effective tree-based generation for modern LLMs requires two ingredients: a principled way to *truncate* the exponentially large tree of possible completions, and a good heuristic for *partial exploration* under limited compute. Since the full decoding tree is intractable, it can be reduced by truncating the next-token distribution. Truncated samplers such as top-k and nucleus sampling (Fan et al., 2018; Holtzman et al., 2020) are widely used to stabilize generation, and are further motivated by the observation that extremely low-probability tokens are often correlated with epistemic errors (Li et al., 2026). A particularly effective truncation strategy is ε-sampling (Hewitt et al., 2022), which keeps only tokens whose probability exceed a threshold ε. Like other truncated sampling strategies, ε-sampling is learning-free, easy to implement, and adds no extra compute overhead.

Motivated by these observations, we propose **ε-Leaf Enumeration (ELE)**, an alternative to (sampled) self-consistency that minimizes repetitions via tree search. ELE *enumerates* distinct leaves in the ε-sampling generation tree by choosing *deterministically* which tokens to explore instead of *sampling* from the vocabulary probability distribution. The output is a set of diverse reasoning chains that never generate duplicate traces. Figure 1 **(bottom)** shows the search tree of ELE as compared to self-consistency. Note that ELE expands a much larger fraction of the tree (5 leaves), resulting in a diverse set of answers. This makes ELE a natural replacement for self-consistency in settings where repeated sampling is wasteful, such as in code synthesis and tightly specified math problems. Since ELE expands a shared tree, it never re-generates the same prefix, but computes each prefix once, caches it, and then reuses it for all descendant branches leading to explicit *prefix reuse*. Therefore, ELE lowers the number of generated tokens for the same number of sequences compared to self-consistency.

In this paper, we introduce the proposed ELE (see Section 2), and evaluate the algorithm on mathematical reasoning, code generation, and commonsense reasoning tasks. Empirically, at equal token budgets, ELE explores more distinct completions than self-consistency, which translates into up to 5% higher accuracy. It also generates fewer new tokens for similar performance. This results in reduced inference time especially in memory-constrained conditions (see Section 3). ELE utilizes prefix caching in vLLM (Kwon et al., 2023) and SGLang (Zheng et al., 2024) and demonstrates how the tree structure of ELE translates into speed benefits, especially under memory constraints.

2 DEFINING ε-LEAF ENUMERATION

ε-sampling (Hewitt et al., 2022) defines a straightforward truncation sampling strategy. At each generation step, only tokens whose probability exceeds a fixed threshold ε can be sampled. Let $p_t(\cdot \mid \mathbf{x}_{<t})$ denote the next-token distribution of the model at step t given prefix $\mathbf{x}_{<t}$ over vocabulary $\mathcal{V}$. ε-sampling restricts the next-token distribution by only considering tokens in the active set $A_\varepsilon(\mathbf{x}_{<t}) = \{v \in \mathcal{V} : p_t(v \mid \mathbf{x}_{<t}) > \varepsilon\}$, where $\varepsilon \in (0, 1]$. If the active set is empty due to all tokens being too low in probability ($A_\varepsilon(\mathbf{x}_{<t}) = \emptyset$) ε-sampling chooses the token with largest probability $g(\mathbf{x}_{<t}) \coloneqq \arg\max_u p_t(u \mid \mathbf{x}_{<t})$. More specifically, ε-sampling samples the truncated distribution

$$q_\varepsilon(v \mid \mathbf{x}_{<t}) = \begin{cases} \mathbb{1}[v = g(\mathbf{x}_{<t})] & \text{if } |A_\varepsilon(\mathbf{x}_{<t})| \leq 1, \\ \dfrac{p_t(v \mid \mathbf{x}_{<t})}{Z_\varepsilon(\mathbf{x}_{<t})} \mathbb{1}[v \in A_\varepsilon(\mathbf{x}_{<t})] & \text{if } |A_\varepsilon(\mathbf{x}_{<t})| \geq 2, \end{cases}, \tag{1}$$

where $\mathbb{1}$ denotes the indicator function and the normalization constant is given by $Z_\varepsilon(\mathbf{x}_{<t}) = \sum_{v \in A_\varepsilon(\mathbf{x}_{<t})} p_t(v \mid \mathbf{x}_{<t})$. For a sequence $\mathbf{x}_{1:T} \in \mathcal{V}^T$, the ε-sampling probability Q_ε is the product of the conditional probabilities of Equation (1), $Q_\varepsilon(\mathbf{x}_{1:T}) = \prod_{t=1}^{T} q_\varepsilon(\mathbf{x}_t \mid \mathbf{x}_{<t})$.

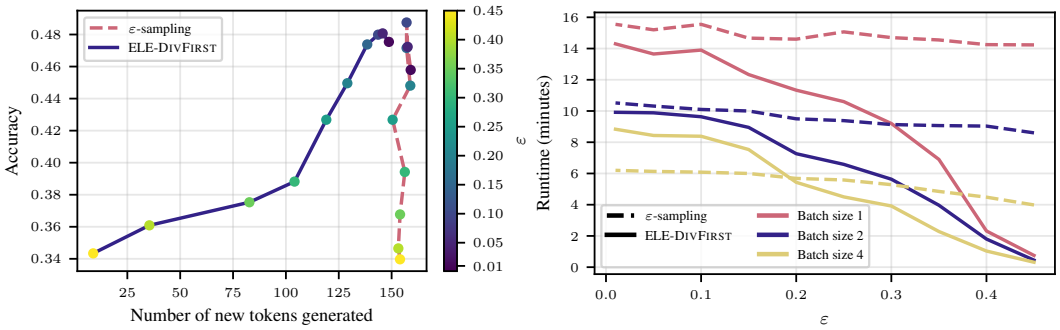


Figure 3: **Left**: **ELE outperforms ε-sampled self-consistency with far fewer new tokens generated (average over $k = 32$).** ε-sampling generates many more tokens for similar performance as compared to ELE. ε is colored according to its value. **Right**: **Higher ε results in much lower runtime for ELE than ε-sampling at $k = 32$.** The time benefit is the most significant at small batch sizes and high ε. Higher ε enables more prefix reuse and larger runtime benefits. See Section 3 for details.

Li et al. (2026) show that low probability tokens (with $p_t(v \mid \mathbf{x}_{<t}) < \varepsilon$) correlate with large *epistemic uncertainty*, i.e., errors that stems from the lack of knowledge in the model. ε-sampling and other truncated samplers effectively improve performance by removing those tokens from consideration.

ε-Leaf Enumeration is our proposed tree-based sampling algorithm operating on the tree defined by the truncated ε-sampling distribution (=: *ε-pruned tree*). Nodes in the tree $\mathbf{x}_{<t}$ are prefixes with active sets $|A_\varepsilon(\mathbf{x}_{<t})| > 1$, and edges follow the next-token distribution $q_\varepsilon(\cdot \mid \mathbf{x}_{<t})$. ELE replaces sampling from q_ε with a *deterministic* expansion at a chosen node. It follows three steps: (1) From a prompt, ELE generates a greedy sequence $\mathbf{x}_{1:T}$ to an end-of-sequence token (EOS), while keeping track of active sets along the path. (2) ELE decides on the next branching location i of the tree and picks an alternative token v from the active set at that position. (3) The alternative token is appended to the prefix $\mathbf{x}_{1:i-1} \circ v$. Then, ELE produces a greedy generation from that point until an EOS token is reached. Step (2) and Step (3) are repeated until there are no remaining active sets with size > 1 or until a maximum of k *leaf nodes* are created. ELE returns a set of distinct leaves and corresponding weights.

Expanding a node. A node $\mathbf{x}_{<t}$ is expanded depending on the size of its active set. *Branching case* $|A_\varepsilon(\mathbf{x}_{<t})| > 1$: One child is created for each token in the active set with prefix $\mathbf{x}_{<t} \circ v$, path probability $Q_\varepsilon(\mathbf{x}_{<t} \circ v)$, and edge weight $q_\varepsilon(v \mid \mathbf{x}_{<t})$. *Non-branching case* $|A_\varepsilon(\mathbf{x}_{<t})| \leq 1$: One child is created from $v = g(\mathbf{x}_{<t})$ with prefix $\mathbf{x}_{<t} \circ v$, path probability $Q_\varepsilon(\mathbf{x}_{<t} \circ v)$, and edge weight 1.0.

For any prefix $\mathbf{x}_{<t}$, its path probability is the product of edge weights on its unique root-to-leaf path. If the number of leaves, k, is arbitrarily large, the support of ε-sampling and ELE is the same (see Appendix D.2) which can be practically infeasible to explore the full ε-pruned tree, depending on the size of ε. We define *coverage* in Definition 2.1 to quantify how much of the tree has been explored.

Definition 2.1 (Coverage)**.** Let $\mathcal{L}$ be the set of all leaves in an ε-pruned tree. Let $\overline{\mathcal{L}}$ be the set of leaves returned by ELE, and let $Q_\varepsilon(\mathbf{x})$ for $\mathbf{x} \in \overline{\mathcal{L}}$ be their corresponding weights. We define the *coverage* m of $\overline{\mathcal{L}}$ as the sum over all leaves in the three $m(\overline{\mathcal{L}}) = \sum_{\mathbf{x} \in \overline{\mathcal{L}}} Q_\varepsilon(\mathbf{x})$.

If all leaf probabilities were known, the maximizer of m would be the set of the k *distinct* highest-probability leaves. In practice, $\mathcal{L}$ is exponentially large, and leaf probabilities are only available in expanded regions of the tree; unexpanded regions remain unknown. Note that the *expected* coverage of ε-sampling leads to diminishing returns for an increasing k due to duplicate leaves (Appendix D.1).

In contrast, ELE deterministically expands unexplored branches of the same ε-pruned tree to avoid generating duplicate leaves. For the same leaf budget k, ELE allocates more compute towards uncovering new probability mass. Therefore, ELE acts as a *compute-budgeted* procedure to order the exploration of the tree in a way that maximizes coverage (see Appendix D.3). We can contrast different local strategies to *deterministically* enumerate the branches and select the order of nodes to explore.

Branching with ProbFirst. ProbFirst expands the branch that carries the largest probability mass seen so far. Figure 2 **(left)** shows the generation in order. With $\varepsilon = 0.1$, ProbFirst first generates branch L_1 greedily. Then, it explores branch L_3 and branch L_2. When branch L_5 emerges, it falls below the threshold so it is pruned. Next, L_4 is deterministic under ε-sampling at that node.

Table 1: **ELE outperforms all other methods.** Performance (maj@k and pass@k) of various methods on GSM8K (Cobbe et al., 2021), Humaneval (Chen et al., 2021), and MMLU-Pro (Wang et al., 2024c) with the Qwen2.5-0.5B-Instruct (Qwen et al., 2025) model with temperature $\tau = 1$ and $\varepsilon = 0.05$ unless otherwise stated. Values are blank when the method is not applicable to the metric.

Method	GSM8K			Humaneval			MMLU-Pro		
	maj@2	maj@4	maj@8	pass@2	pass@4	pass@8	maj@2	maj@4	maj@8
Self-consistency ($\tau = 1$)	19.26	25.32	31.61	18.29	26.22	33.54	13.61	15.15	16.24
Self-consistency ($\tau = 0.6$)	29.57	36.24	41.32	31.10	42.07	48.17	16.15	16.43	17.84
Self-consistency (ε-sampling)	26.84	33.51	40.94	32.93	39.63	47.56	14.89	16.07	17.20
Beam search	35.10	36.47	36.92	31.10	29.88	32.93	15.49	15.29	15.01
Diverse beam search	-	36.47	41.39	-	32.32	33.54	-	16.20	16.58
DeepConf	21.68	26.91	33.36	-	-	-	14.41	15.58	16.76
Self-certainty	20.09	25.32	32.22	-	-	-	14.51	15.92	14.68
ELE-DivFirst	**35.41**	40.41	**44.35**	**39.02**	**46.34**	51.22	16.46	16.96	17.84
ELE-ProbFirst	34.57	**40.64**	44.05	38.41	45.12	**52.44**	**16.64**	**17.37**	**17.97**

Branching with DivFirst. DivFirst encourage diversity *early* on by selecting the smallest token position. This forces early divergence near the root. In Figure 2 **(left)**, fist, branch L_1 is generated greedily. Then, at the first branching point, L_4 is created and L_5 is rejected. The next earliest branch points is at L_3, and then at L_2. We also compare other branching rules in Appendix D.1.

Early stopping. Completions can differ at the branching token but still lead to the *same greedy suffix* a few steps later (Hong et al., 2025). To avoid repeatedly generating the same continuation, we early-stop *after* the branch point once such a merge is detected (Figure 2 **(right)**). This rule only skips redundant suffix decoding; it does not affect branching prefixes or leaf-weight computation. Early stopping substantially increases coverage without introducing wasted rollouts (Appendix C).

3 Experiments

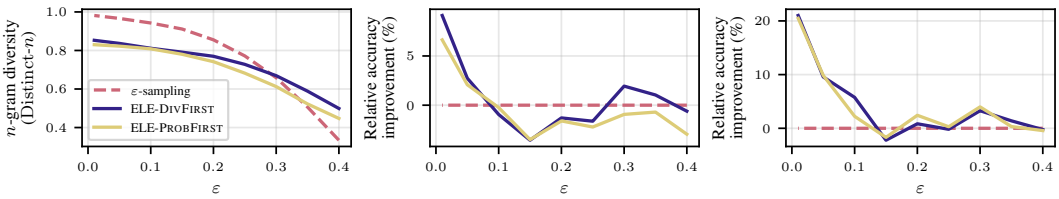
We run experiments with the Qwen (Qwen et al., 2025) and Llama (Grattafiori et al., 2024) model families. We test mathematical reasoning with GSM8K (Cobbe et al., 2021), code generation with HumanEval (Chen et al., 2021), and general reasoning with MMLU-Pro (Wang et al., 2024c). We compare ELE and its variations to other state-of-the-art parallel reasoning strategies such as DeepConf (Fu et al., 2025), Self-Certainty (Kang et al., 2025), beam search (Lowerre, 1976; Sutskever et al., 2014), and diverse beam search (Vijayakumar et al., 2016). Also, refer to Appendix B and Appendix E.

Table 1 shows maj@k for Qwen2.5-0.5B-Instruct where $k \in \{2, 4, 8\}$ denotes the number of leaves for ELE. The same k is used as beam size for beam search, and to indicate completed solutions for other methods. On GSM8K, ELE-DivFirst outperforms ε-sampling with $\approx 8\%$, $\approx 7\%$, and $\approx 3\%$ for $k = 2, 4, 8$, respectively. On Humaneval, ELE-DivFirst improves by $\geq 6\%$. For MMLU-Pro, the improvements are smaller and ELE-DivFirst improves over ε-sampling by $\approx 1\%$. (Diverse) Beam search does not result in consistent performance improvements as larger beams and more compute is used. This shows its limited effectiveness as a breadth-first, per-step tree search algorithm.

ELE matches accuracy with fewer tokens and improves accuracy for fixed token budget. Figure 3 **(left)** shows that ELE reaches comparable majority-voting and pass@k accuracy with far fewer tokens than ε-sampling. For $\approx 34\%$, ε-sampled self-consistency with $\varepsilon = 0.45$ needs > 150 tokens while ELE-DivFirst uses < 20 new tokens on average per question (per sequence). Larger ε increases prefix reuse by branching later (see also Figure 12). For small ε, ELE plateaus whereas ε-sampling *degrades*. With a fixed token budget, prefix reuse lets ELE run more sequences and thus improve majority

Table 2: Diversity and coverage comparison between methods at $\varepsilon = 0.3$ and $k = 8$ showing that ELE-DivFirst has higher diversity whereas ELE-ProbFirst has higher coverage.

	Coverage	Diversity	pass@8	maj@8
ε-sampling	0.157	0.6588	52.92	37.00
ELE-DivFirst	0.228	0.6692	55.65	37.91
ELE-ProbFirst	0.296	0.6454	52.08	38.44



(a) Diversity **(b)** pass@8 **(c)** maj@8

Figure 4: **Higher diversity does not necessarily lead to better majority-voted performance. (a)** ε-sampling shows the largest diversity for $\varepsilon < 0.3$. DIVFIRST has larger diversity than PROBFIRST. **(b)** Relative accuracy improvement (%, pass@8) of ELE variations compared to ε-sampling largest at low ε. Pass@k reflects diversity does not necessarily lead to better maj@k performance as seen in **(c)** ELE has the highest maj@8 performance overall. See Section 3.

voting (see Figure 8 in Appendix E), yielding the largest gains at low budgets. The performance saturates as the budget grows but ELE remains better than ε-sampling.

Diversity does not generally improve accuracy. Figure 4 **(a)** shows that for $\varepsilon < 0.3$, ε-sampling yields much higher 10-gram diversity which does not improve maj@k performance (Figure 4 **(b)** and Figure 4 **(c)**). Despite lower diversity, ELE outperforms the baseline. This suggests n-gram diversity to be a surface-level proxy which is only weakly aligned with correctness: it can change wording without increasing the probability mass on correct solutions.

ELE covers a larger part of the ε-pruned search tree. DIVFIRST branches early to increase diversity and it achieves the highest n-gram diversity and pass@8 in Table 2 (also showing diversity). PROBFIRST prioritizes high-probability branches and, therefore, covers the largest (immediate) mass of the search tree (see Figure 9 where PROBFIRST consistently has higher coverage than DIVFIRST).

Both variations achieve substantially larger coverage than ε-sampling. A key reason is that ε-sampling shares its compute budget across i.i.d. trajectories *with replacement*, thus, many samples repeat. Contrary, ELE allocates a per-sequence budget of k to distinct trajectories and, thus, explores more of the tree (see Appendix D.1). Early-stopping increases coverage further, especially for small ε (Appendix C.2) with negligible added token overhead (see Figure 6 in Appendix E).

ELE improves runtimes and cache hit rates. Figure 3 **(right)** shows that ELE runs faster than ε-sampling on SGLang, with largest gains at small batch sizes $\in \{1, 2\}$ where the baseline cannot effectively exploit parallelization in the generation of all traces. At high ε, ELE often traverses the full tree before the k-leaf budget is exhausted. Additionally, ELE achieves a higher cache hit rate because answer prefixes can be reused. In contrast, baseline ε-sampling can only reuse $k - 1$ of all k question prompts. All answer tokens have to be generated. Further results are provided in Appendix E.4.

4 DISCUSSION AND CONCLUSION

ELE as a general recipe for truncated sampling. Although we instantiate ELE with ε-truncation, the underlying idea is more general. Whenever a decoding policy induces a *truncated vocabulary* at each step, generation corresponds to traversing a pruned tree, and we can apply repetition-free leaf enumeration. This includes truncation schemes (Nguyen et al., 2025), locally-typical sampling (Meister et al., 2025), and entropy-based truncation (Zhang et al., 2024). Viewed this way, ELE is best seen as a general *enumeration rule* to reduce redundancy and increase distinct coverage under a fixed leaf budget.

Branch structure as a training signal. The tree view also enables training-time uses for methods that treat self-consistency as a reward (Zuo et al., 2025). Instead of assigning reward only to full sequences, ELE exposes *explicit branch points* with prefix probabilities and leaf outcomes, allowing finer-grained credit assignment which might be beneficial in settings where full rollouts are expensive.

Conclusion. ELE treats decoding as exploring a pruned tree with *coverage* as the objective. By deterministically enumerating distinct leaves and making prefix sharing explicit, ELE results in broader coverage of plausible continuations and yields practical gains on inference engines leading to faster runtime under memory constraints. We hope this perspective encourages future work on tree-based methods for decoding making coverage and prefix reuse explicit and enabling better allocation of compute.

ACKNOWLEDGEMENTS

This work was supported by the Tübingen AI Center. Xueyan Li is supported by ETH Zurich and Max Planck ETH Center for Learning Systems. Johannes Zenn acknowledges funding by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC number 2064/1 – Project number 390727645. The authors thank the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for supporting Johannes Zenn.

REFERENCES

Pranjal Aggarwal, Aman Madaan, Yiming Yang, et al. Let's sample step by step: Adaptive-consistency for efficient reasoning and coding with llms. In *Conference on Empirical Methods in Natural Language Processing*, 2023.

Richard Bellman. Dynamic programming. *science*, 153(3731):34–37, 1966.

Brian J Chan, Mao-xun Huang, Jui-Hung Cheng, Chao-Ting Chen, and Hen-Hsen Huang. Efficient beam search for large language models using trie-based decoding. In *Conference on Empirical Methods in Natural Language Processing*, 2025.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri 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, Josh 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. *arXiv preprint arXiv:2107.03374*, 2021.

Xinyun Chen, Renat Aksitov, Uri Alon, Jie Ren, Kefan Xiao, Pengcheng Yin, Sushant Prakash, Charles Sutton, Xuezhi Wang, and Denny Zhou. Universal self-consistency for large language models. In *International Conference on Machine Learning, Workshop on In-Context Learning*, 2024.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training Verifiers to Solve Math Word Problems. *arXiv preprint arXiv:2110.14168*, 2021.

Eldan Cohen and Christopher Beck. Empirical analysis of beam search performance degradation in neural sequence models. In *International Conference on Machine Learning*, 2019.

Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical Neural Story Generation. In Iryna Gurevych and Yusuke Miyao (eds.), *Annual Meeting of the Association for Computational Linguistics, Long Papers*, 2018.

Yichao Fu, Xuewei Wang, Yuandong Tian, and Jiawei Zhao. Deep Think with Confidence. *arXiv preprint arXiv:2508.15260*, 2025.

Weizhi Gao, Xiaorui Liu, Feiyi Wang, Dan Lu, and Junqi Yin. Decoding memories: An efficient pipeline for self-consistency hallucination detection. *arXiv preprint arXiv:2508.21228*, 2025.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 548 other authors. The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783*, 2024.

John Hewitt, Christopher D. Manning, and Percy Liang. Truncation Sampling as Language Model Desmoothing. *arXiv preprint arXiv:2210.15191*, 2022.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The Curious Case of Neural Text Degeneration. *arXiv preprint arXiv:1904.09751*, 2020.

Colin Hong, Xu Guo, Anand Chaanan Singh, Esha Choukse, and Dmitrii Ustiugov. Slim-sc: Thought pruning for efficient scaling with self-consistency. In *Conference on Empirical Methods in Natural Language Processing*, 2025.

Jordan Juravsky, Bradley Brown, Ryan Saul Ehrlich, Daniel Y. Fu, Christopher Re, and Azalia Mirhoseini. Hydragen: High-Throughput LLM Inference with Shared Prefixes. In *International Conference on Machine Learning, Workshop on Efficient Systems for Foundation Models II*, 2024.

Zhewei Kang, Xuandong Zhao, and Dawn Song. Scalable Best-of-N Selection for Large Language Models via Self-Certainty. *arXiv preprint arXiv:2502.18581*, 2025.

Philipp Koehn and Rebecca Knowles. Six Challenges for Neural Machine Translation. In *Annual Meeting of the Association for Computational Linguistics, Workshop on Neural Machine Translation*, 2017.

Wouter Kool, Herke van Hoof, and Max Welling. Stochastic Beams and Where to Find Them: The Gumbel-Top-k Trick for Sampling Sequences Without Replacement. *arXiv preprint arXiv:1903.06059*, 2019.

Solomon Kullback and Richard A Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22, 1951.

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 *Symposium on Operating Systems Principles*, 2023.

Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. A Diversity-Promoting Objective Function for Neural Conversation Models. *arXiv preprint arXiv:1510.03055*, 2016.

Xueyan Li, Guinan Su, Mrinmaya Sachan, and Jonas Geiping. Sample smart, not hard: Correctness-first decoding for better reasoning in LLMs. In *International Conference on Learning Representations*, 2026.

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. In *International Conference on Learning Representations*, 2024.

Lei Lin, Jiayi Fu, Pengli Liu, Qingyang Li, Yan Gong, Junchen Wan, Fuzheng Zhang, Zhongyuan Wang, Di Zhang, and Kun Gai. Just Ask One More Time! Self-Agreement Improves Reasoning of Language Models in (Almost) All Scenarios. *arXiv preprint arXiv:2311.08154*, 2024.

Bruce T Lowerre. *The harpy speech recognition system.* Carnegie Mellon University, 1976.

Clara Meister, Tiago Pimentel, Gian Wiher, and Ryan Cotterell. Locally Typical Sampling. *arXiv preprint arXiv:2202.00666*, 2025.

Minh Nhat Nguyen, Andrew Baker, Clement Neo, Allen Roush, Andreas Kirsch, and Ravid Shwartz-Ziv. Turning Up the Heat: Min-p Sampling for Creative and Coherent LLM Outputs. *arXiv preprint arXiv:2407.01082*, 2025.

Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 Technical Report. *arXiv preprint arXiv:2412.15115*, 2025.

Kensen Shi, David Bieber, and Charles Sutton. Incremental Sampling Without Replacement for Sequence Models. *arXiv preprint arXiv:2002.09067*, 2021.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Yixuan Su, Tian Lan, Yan Wang, Dani Yogatama, Lingpeng Kong, and Nigel Collier. A contrastive framework for neural text generation. In *International Conference on Neural Information Processing Systems*, 2022.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 2014.

Amir Taubenfeld, Tom Sheffer, Eran Ofek, Amir Feder, Ariel Goldstein, Zorik Gekhman, and Gal Yona. Confidence improves self-consistency in LLMs. In *Findings of the Association for Computational Linguistics*, 2025.

Ashwin K Vijayakumar, Michael Cogswell, Ramprasath R Selvaraju, Qing Sun, Stefan Lee, David Crandall, and Dhruv Batra. Diverse beam search: Decoding diverse solutions from neural sequence models. *arXiv preprint arXiv:1610.02424*, 2016.

Luke Vilnis, Yury Zemlyanskiy, Patrick Murray, Alexandre Tachard Passos, and Sumit Sanghai. Arithmetic sampling: parallel diverse decoding for large language models. In *International Conference on Machine Learning*, 2023.

Han Wang, Archiki Prasad, Elias Stengel-Eskin, and Mohit Bansal. Soft Self-Consistency Improves Language Models Agents. In *Annual Meeting of the Association for Computational Linguistics, Short Papers*, 2024a.

Xinglin Wang, Yiwei Li, Shaoxiong Feng, Peiwen Yuan, Boyuan Pan, Heda Wang, Yao Hu, and Kan Li. Integrate the essence and eliminate the dross: Fine-grained self-consistency for free-form language generation. In *Annual Meeting of the Association for Computational Linguistics, Long Papers*, 2024b.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-Consistency Improves Chain of Thought Reasoning in Language Models, March 2023. arXiv:2203.11171 [cs].

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. MMLU-Pro: A More Robust and Challenging Multi-Task Language Understanding Benchmark. *arXiv preprint arXiv:2406.01574*, 2024c.

Fangzhi Xu, Hang Yan, Chang Ma, Haiteng Zhao, Jun Liu, Qika Lin, and Zhiyong Wu. ϕ-decoding: Adaptive foresight sampling for balanced inference-time exploration and exploitation. In *Annual Meeting of the Association for Computational Linguistics, Long Papers*, 2025.

Yilin Yang, Liang Huang, and Mingbo Ma. Breaking the Beam Search Curse: A Study of (Re-)Scoring Methods and Stopping Criteria for Neural Machine Translation. In *Conference on Empirical Methods in Natural Language Processing*, 2018.

Jinwei Yao, Kexun Zhang, Kaiqi Chen, Jiaxuan You, Zeke Wang, Binhang Yuan, and Tao Lin. DeFT: Flash Tree-attention with IO-Awareness for Efficient Tree-search-based LLM Inference. In *International Conference on Learning Represenation, Workshop: How Far Are We From AGI*, 2024.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in neural information processing systems*, 2023.

Jinghan Zhang, Xiting Wang, Fengran Mo, Yeyang Zhou, Wanfu Gao, and Kunpeng Liu. Entropy-based exploration conduction for multi-step reasoning. In *Findings of the Association for Computational Linguistics*, 2025.

Shimao Zhang, Yu Bao, and Shujian Huang. EDT: Improving Large Language Models' Generation by Entropy-based Dynamic Temperature Sampling. *arXiv preprint arXiv:2403.14541*, 2024.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. In *Advances in neural information processing systems*, 2024.

Jiace Zhu, Yingtao Shen, Jie Zhao, and An Zou. Path-consistency: Prefix enhancement for efficient inference in llm. *arXiv preprint arXiv:2409.01281*, 2024.

Yuxin Zuo, Kaiyan Zhang, Li Sheng, Shang Qu, Ganqu Cui, Xuekai Zhu, Haozhan Li, Yuchen Zhang, Xinwei Long, Ermo Hua, Biqing Qi, Youbang Sun, Zhiyuan Ma, Lifan Yuan, Ning Ding, and Bowen Zhou. TTRL: Test-Time Reinforcement Learning. *arXiv preprint arXiv:2504.16084*, 2025.

A Related Work

Our work targets a specific inefficiency in test-time scaling for constrained reasoning: repeated sampling produces many duplicate prefixes and often re-discovers the same high-probability traces. ELE addresses this by (i) enumerating *distinct* leaves in a truncated sampling tree and (ii) exploiting prefix reuse across branches.

Test-time ensembling and self-consistency. Self-consistency aggregates multiple reasoning traces to improve accuracy, typically via majority voting on extracted answers (Wang et al., 2023). Follow-up work extends aggregation beyond simple answer extraction to free-form settings, e.g., by letting an LLM compare, synthesize, or select among sampled traces (Wang et al., 2024a; Chen et al., 2024; Wang et al., 2024b). Orthogonally, several methods reweight traces using confidence or selection criteria (e.g., self-certainty, filtering) to improve best-of-N style inference (Kang et al., 2025; Fu et al., 2025). In contrast, ELE focuses on making the *sampling stage* itself non-redundant, guaranteeing a set of distinct answer traces.

Reducing redundancy in aggregation. A line of work reduces compute by stopping early or pruning unpromising traces. Early-stopping self-consistency terminates sampling once answers stabilize (Li et al., 2024), and adaptive-consistency allocates variable budgets across prompts/samples (Aggarwal et al., 2023). Path-based strategy reuses partial computation by expanding from promising prefixes, improving latency while maintaining accuracy (Zhu et al., 2024). ELE is complementary with a different mechanism: rather than adapting the number of i.i.d. samples, it deterministically enumerates *previously unexplored* branches of the same truncated tree. This also connects to broader work on sampling sequences without replacement that aim to avoid duplicates while preserving distributional guarantees (Kool et al., 2019; Shi et al., 2021).

Search-based decoding. Beam search (Lowerre, 1976; Sutskever et al., 2014) makes the decoding tree explicit but suffers from degeneration and low diversity; diverse beam search addresses this by adding diversity-promoting terms (Vijayakumar et al., 2016). Other approaches explicitly promote beam diversity, e.g., via arithmetic coding style constructions (Vilnis et al., 2023). ELE differs from per-step breadth-first search. It traverses the ε-pruned tree in a depth-first-like manner (greedy completions) and allocates a leaf budget to distinct trajectories, which is particularly well-matched to self-consistency-style voting in constrained domains.

Tree-based structures. Tree-of-thought style inference explores multiple reasoning paths with self-evaluation (Yao et al., 2023). However, it requires repeated model prompting for each branch. Entro-duction uses entropy/varentropy-style signals to balance computation and exploration in multi-step reasoning (Zhang et al., 2025), and ϕ-decoding uses simulated foresight paths for globally informed step selection but also requires expensive extra computation (Xu et al., 2025). ELE is cheaper by using only next-token probabilities and spends compute primarily on full leaf completions rather than partial rollouts plus re-ranking.

Efficiency via prefix reuse and decoding systems. Efficient implementations for multi-branch decoding leverage shared prefixes and specialized kernels or KV-cache data structures (Yao et al., 2024; Juravsky et al., 2024; Zheng et al., 2024). Trie-based decoding for beam search similarly shares KV cache across beams with common prefixes (Chan et al., 2025). ELE directly relates to these system optimizations because its exploration is explicitly tree-structured, which reuses the longest available prefix.

B DATASETS AND PARAMETERS

MMLU-Pro (Wang et al., 2024c) metrics are weighted by the number of questions in each category.

All plotted diagrams and tables use Qwen2.5-0.5B-Instruct unless stated otherwise. All experiments are run on B200 or H100 GPUs using vLLM. The default benchmark for diagrams is GSM8K unless stated otherwise.

Figure 3 **(left)** uses $k = 32$. The number of new tokens generated is summed over the full dataset, and averaged per question per k. Since at high ε, ELE does not need to use the full 32-sequences budget to explore the tree fully, its averaged number of new tokens is low.

Figure 3 **(right)** uses $\varepsilon = 0.3$ and a fixed token budget for each question. For each question, sequences are generated one by one (not in parallel) until all tokens are used up.

C Ablations of the Proposed Algorithm

C.1 Search Algorithms

In addition to DivFirst and ProbFirst, which we discussed in the main body, we test three additional search algorithms

- RandBranch chooses branching points by *sampling* from the distribution defined by the ProbFirst weights. We define a categorical distribution proportional to the probability mass under ε-sampling and sample it to get the next branching point. This is no longer deterministic, unlike ProbFirst.
- GlobalProb chooses branches that have the highest global edge weight. Instead of choosing branches that have the highest global path weight so far (from accumulated edge probabilities), it chooses to explore the alternative token that has the highest probability among all the branches explored so far.
- DFS is Depth-First-Search that starts with greedy generation, but deepest branches are explored first. This prioritizes late branching.

Table 3 shows that these alternative methods perform worse than the methods in the main paper.

Table 3: Adding an early-stopping condition to ELE baselines generally improves performance. Alternative branching algorithms DFS and GlobalProb yield worse performance.

Method	GSM8K			Humaneval		
	maj@2	maj@4	maj@8	pass@2	pass@4	pass@8
ELE-DivFirst	35.41	40.41	44.35	39.02	46.34	51.22
w/o early stopping	34.95	40.49	45.11	39.02	46.34	50.61
ELE-ProbFirst	34.57	40.64	44.05	38.41	45.12	52.44
w/o early stopping	34.27	39.95	43.29	38.41	45.12	51.22
ELE-RandBranch	34.72	39.27	44.43	34.76	46.34	53.05
w/o early stopping	33.97	39.04	42.68	35.98	42.68	48.78
ELE-DFS	33.81	33.81	34.12	31.10	31.71	31.71
ELE-GlobalProb	33.97	34.72	35.25	34.76	37.8	40.24

C.2 Early Stopping

We also provide an ablation that removes the early stopping condition. Table 3 shows that adding early stopping generally results in better performance as it adds 1. more coverage 2. more diversity with minimal cost of additional tokens. When using early stopping, we treat a terminated sibling branch as a covered leaf mass (not a completed leaf) because it deterministically merges into an already-generated suffix. In plots, 'total coverage' includes completed leaves + early-stopped covered mass.

In Figure 6, we find that the early-stopping condition helps us achieve more coverage, especially when ε is small. By assuming that identical continuations other than the first branch token will result in the same sequence, we do not spend more tokens on them, but still count their probabilities as covered. This helps the model to achieve better performance by not wasting more tokens on identical continuations, with minimal additional "wasted" tokens.

As Figure 5 shows, the percentage of wasted tokens is around 1% of the total number of generated tokens, which adds minimal compute overhead. As n increases, the early-stopping condition is triggered less, resulting in fewer wasted tokens. The best n is around 10–15 and we use $n = 10$ in the main paper.

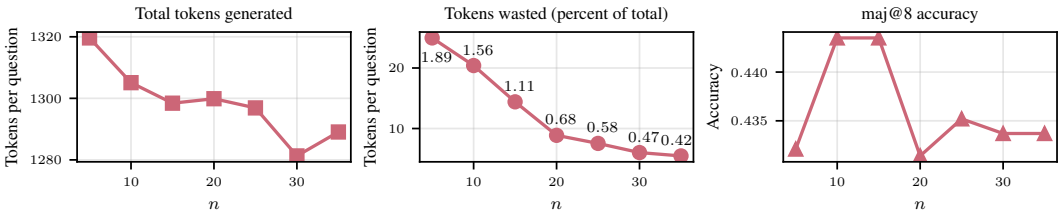


Figure 5: The x-axis shows n, the number of repeated token generated from a branch point, after which this branch will be terminated. As n increases, the early-stopping condition is triggered less frequently, resulting in fewer wasted tokens.

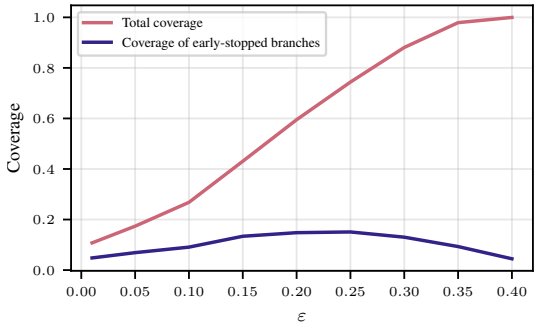


Figure 6: Total coverage includes the mass covered by completed leaves and early-stopped branches. Coverage of early-stopped branches only measures the coverage of branches that do not reach EOS but are stopped early due to overlap in their last 10-gram with previous branches.

C.3 Answer Aggregation Methods

Since we can obtain per-sequence probabilities during generation, we can aggregate answers by weighting them according to sequence-wise probabilities. However, we find that this method does not result in systematic improvement in performance, compared to weighting all final answers equally, as seen in Table 4.

Table 4: Weighing final answers by their leaf probabilities do not result in systematic increase in performance.

	Probability weighted	Equally weighted
Qwen2.5-0.5B-Instruct		
DivFirst	0.4170	0.4435
ProbFirst	0.4177	0.4405
RandBranch	0.4109	0.4443
Qwen2.5-7B-Instruct		
DivFirst	0.8878	0.8992
ProbFirst	0.8916	0.8886
RandBranch	0.8908	0.8893

D More Context on ε-Leaf Enumeration

Appendix D.1 shows that the expected coverage of ε-sampling leads to diminishing returns when increasing k, while the motivation of ELE is to explore more *new* probability mass. Appendix D.3 shows that our greedy choice is locally optimal and minimizes computational cost. Appendix E.1 shows an extension to Figure 3 **(left)**.

D.1 Expected Coverage of ε-Sampling

Let $\mathcal{L}$ denote the set of all leaves in the ε-pruned tree and let $q(\mathbf{x}) \coloneqq Q_\varepsilon(\mathbf{x})$ for $\mathbf{x} \in \mathcal{L}$. Let $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(k)} \sim q$ be the k leaves produced by ε-sampling, and let $S_k \coloneqq \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(k)}\}$ be the set of unique leaves.

Proposition D.1 (Expected coverage of ε-sampling)**.** *The expected coverage of ε-sampling after k samples is*

$$\mathbb{E}\left[m(S_k)\right] = \sum_{\mathbf{x}\in\mathcal{L}} q(\mathbf{x}) \left(1 - (1 - q(\mathbf{x}))^k\right). \tag{2}$$

The expected marginal gain is

$$\mathbb{E}\left[m(S_{k+1}) - m(S_k)\right] = \sum_{\mathbf{x}\in\mathcal{L}} q(\mathbf{x})^2 (1 - q(\mathbf{x}))^k, \tag{3}$$

which is non-increasing in k.

Proof. Let $Y_j, j \in \{1, \ldots, k\}$ be the distinct leaves $\in S_k$. Then,

$$\begin{aligned}
\mathbb{E}\left[m(S_k)\right] &= \mathbb{E}\left[\sum_{y\in S_k} q(y)\right] && (4)\\
&= \mathbb{E}\left[\sum_{y\in\mathcal{L}} q(y)\mathbb{1}\{y \in S_k\}\right] && (5)\\
&= \sum_{y\in\mathcal{L}} q(y)\mathbb{E}[\mathbb{1}\{y \in S_k\}] && (6)\\
&= \sum_{y\in\mathcal{L}} q(y)\mathbb{P}(y \in S_k) && (7)\\
&= \sum_{y\in\mathcal{L}} q(y)(1 - \mathbb{P}(y \notin S_k)) && (8)\\
&= \sum_{y\in\mathcal{L}} q(y)\left(1 - \mathbb{P}(\forall j \in \{1, \ldots, k\} : Y_j \neq y)\right) && (9)\\
&= \sum_{y\in\mathcal{L}} q(y)\left(1 - \prod_{j=1}^{k} \mathbb{P}(Y_j \neq y)\right) && (10)\\
&= \sum_{y\in\mathcal{L}} q(y)\left(1 - (1 - q(y))^k\right). && (11)
\end{aligned}$$

The marginal gain follows by the difference of two of the terms. □

Therefore, while $\mathbb{E}\left[m(S_k)\right]$ is increasing in k, $\mathbb{E}\left[m(S_{k+1}) - m(S_k)\right]$ is decreasing. This shows diminishing returns for the expected coverage of ε sampling. Intuitively, this shows that the "waste" of ε-sampling comes from duplicate leaves since ε-sampling samples full leaves *i.i.d. with replacement*.

Connection to ELE. In contrast, ELE deterministically expands previously unexplored branches of the same ε-pruned tree which *largely avoids generating duplicate leaves*. Thus, for the same leaf budget k, ELE can allocate more compute towards *uncovering new probability mass*.

D.2 Support of Greedy Sampling and ELE

Remark D.2 (If the number of leaves, k, is arbitrarily large, the support of ε-sampling and ELE is the same)**.** *Note that if we explore all branches in the ε-pruned tree, we enumerate all possible sequences $\mathbf{x}$ of ε-sampling with $Q_\varepsilon(\mathbf{x}) > 0$. Thus, their support is the same.*

D.3 Greedy Sampling is Locally Optimal and Compute Optimal

After picking an alternative token at a branch point, we always complete the sequence greedily. For a prefix $\mathbf{x}$ and a horizon T, its best continuation is $V^*(\mathbf{x}, T) = \max_{\mathbf{x}'_{1:T} \in \mathcal{V}^T} \log p(\mathbf{x}'_{1:T} \mid \mathbf{x})$, with Bellman optimality recursion (Bellman, 1966)

$$V^*(\mathbf{x}, T) = \max_{v \in A_\varepsilon(\mathbf{x})} [\log p(v \mid \mathbf{x}) + V^*(\mathbf{x} \circ v, T-1)], \tag{12}$$

where $V^*(\mathbf{x}, 0) = 0$. Greedy sampling replaces the term $V^*(\mathbf{x} \circ v, T-1)$ by 0.

Remark D.3 (Local and compute optimality of greedy sampling)**.** *Greedy sampling $v = g(\mathbf{x}_{<t})$ chooses, by definition, the maximum next-token probability $p(v \mid \mathbf{x}_{<t}) \geq p(v' \mid \mathbf{x}_{<t}) \forall v' \in \mathcal{V}$, and it generates a sample in T forward passes.*

D.4 Implementation of ELE

In this section, we discuss leaf aggregation rules and the practical implementation of ELE in inference engines.

Leaf Aggregation Rules. Regardless of the expansion rules defined in Section 2, each generated leaf $\mathbf{x} \in \overline{\mathcal{L}}$ comes with its exact ε-sampling probability $Q_\varepsilon(\mathbf{x})$. This provides us with sequence probabilities for weighted aggregation of all leaves without any additional model calls. However, in contrast to Wang et al. (2024a), we find that equally weighting each answer is practically more robust—we ablate this in Appendix C.3.

We highlight that ELE always expands *prefixes* which do not need to be recomputed. ELE explores multiple leaves, i.e., full generations, that share long prefixes, which allows convenient key-value (KV) reuse. At any branching point, the child leaf *reuses* all tokens up to the divergence point and continues decoding from there. This is in contrast to standard self-consistency that runs all sequences independently and in parallel, where shared prefixes are generated repeatedly.

vLLM (Kwon et al., 2023) uses *automatic prefix caching* (APC), reusing cached KV blocks from prior requests when needed. This lets any new request skip computing shared prefixes if they have been cached before.

SGLang (Zheng et al., 2024) is built around KV cache reuse across multiple generation calls that is explicitly optimized for tree-based generations, using RadixAttention, which stores KV caches in a radix tree keyed by token prefixes.

ELE benefits directly from these inference engines by avoiding recomputing shared prefixes. While vLLM provides a lightweight implicit reuse path, SGLang has a native tree structure with an explicit reuse path that aligns closely with the structure of our branching algorithm. All of the implementation details can be found in our code[1]. We verify the performance on each of the inference engines in Section 3.

D.5 Evaluation Metrics

In this section, we define the metrics used to evaluate our experimental results in Section 3.

Coverage. We define coverage $m(\overline{\mathcal{L}}) \in [0, 1]$ in Definition 2.1 over a set of leaves $\overline{\mathcal{L}}$ quantifying how much of the ε-pruned tree ELE has explored.

[1]We will release our code upon publication of the paper.

N-gram diversity. The n-gram diversity of a string $\mathbf{s}$ is defined as the fraction of unique n-grams among all n-grams in $\mathbf{s}$ (Li et al., 2016). Specifically,

$$\text{Distinct-}n(\mathbf{s}) = \frac{|\text{unique}(n\text{-grams}(\mathbf{s}))|}{|n\text{-grams}(\mathbf{s})|}. \tag{13}$$

In this paper, we use $n = 10$ for 10-gram diversity and early stopping condition.

Cache hit rate. We measure cache reuse over the *flattened tree* of the k enumerated leaves. Each leaf corresponds to one generation stream, and we concatenate all streams to obtain a single measure L_{flat} that includes all repeated prompt and generated tokens. We report two cache hit rates:

- **Actual cache hit rate** is the total number of cached tokens reported by SGLang (via `cached_tokens`) over the total flat length: $\frac{C_{\text{act}}}{L_{\text{flat}}}$.
- **Theoretical cache hit rate** is the maximum number of tokens that *could* be reused given the enumerated tree structure, i.e., the length of the longest prefix of a leaf that matches any previously generated leaf (including the prompt): $\frac{C_{\text{th}}}{L_{\text{flat}}}$.

E Additional Results

E.1 Accuracy, Number of New Tokens Generated, Token Budget

In this section we give more context on our proposed method. As an extension to Figure 3 **(left)** which only shows when $k = 32$, we additionally provide results for when $k = 8$. Similarly, ELE uses fewer tokens for the same accuracy.

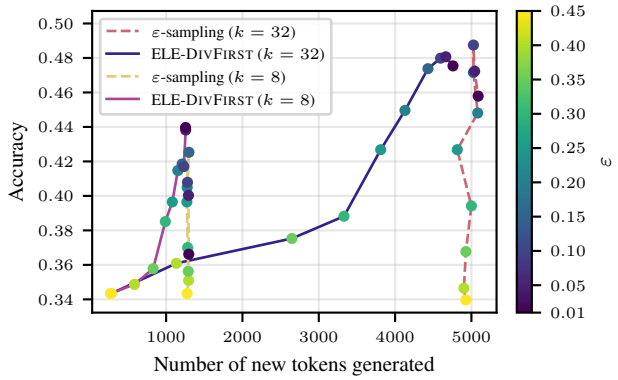


Figure 7: **ELE outperforms ε-sampling with far fewer new tokens generated.** Number of new tokens is counted on average for each question. ε is marked with colors indicating its value. ε-sampling generates many more tokens to achieve a similar performance as ELE. See Section 3.

As discussed in the main text, for a fixed token budget ELE achieves a higher accuracy and generates more sequences as compared to ε-sampling. See Figure 8.

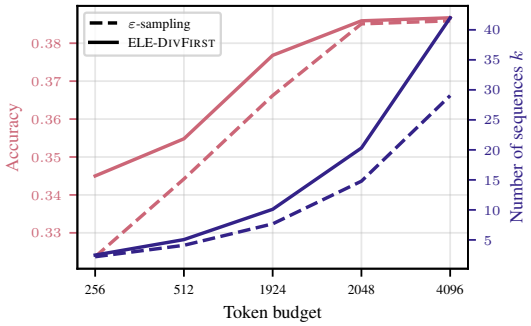


Figure 8: **For a fixed token budget, ELE achieves a higher accuracy and generates more sequences compared to ε-sampling.**

E.2 ELE COVERAGE OVER ε

Figure 9 shows that ELE has higher coverage than ε-sampling over all ε. Additionally, we find that ELE-PROBFIRST outperforms ELE-DIVFIRST in coverage. This is due to the fact that the former is designed to explore the highest probability branches, and, thus, covers the largest mass of the search tree.

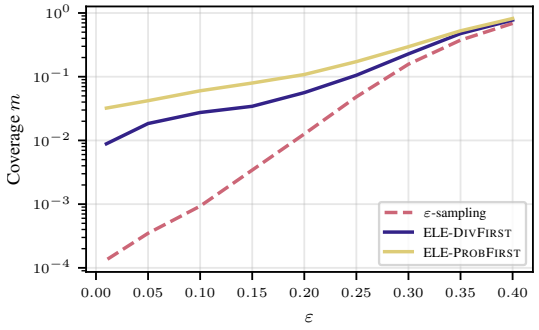


Figure 9: **ELE has higher coverage overall.** Coverage m (see Definition 2.1) plotted against ε at $k = 8$. For small ε, m is much larger for ELE than ε-sampling. See Section 3.

E.3 DIVERSITY

As discussed in Section 1, we investigate redundancy among generations (see Figure 1) by looking at the repetition rate among prefixes. In Figure 10 we find that for code generation, 20% of all prefix tokens are repeated across generated traces. Thinking of generated sequences in terms of a tree structure, this means that early branches are repeatedly traversed.

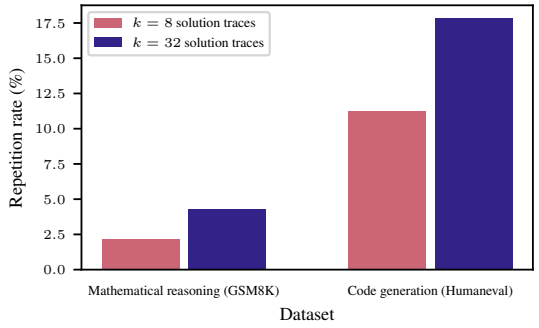


Figure 10: **The repetition rate of prefix tokens increases as more sequences are sampled for ε-sampling.** For mathematical reasoning, the rate nearly doubles from $\approx 2\%$ to $\approx 4\%$. For code generation, the rate moves from $\approx 11\%$ to $\approx 17\%$. See Section 1.

E.4 Inference Speed

In the main body, we discussed the inference speed benefit of using SGLang, which can output the cached token count for each generation request. However, vLLM does not output such data, so we cannot directly measure its cache retrieval rate. We can still see that there is some speed benefit from using ELE at high ε thresholds.

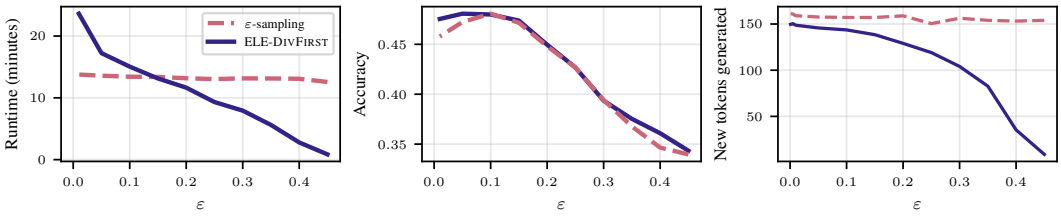


Figure 11: Inference time and performance are measured using vLLM. At higher ε, inference time is shorter than the baseline. The performance boost is also more obvious since far fewer tokens are generated.

E.5 Cache Hit Rate With SGLang

Figure 12 details the cache hit rate for DivFirst and ε-sampling. We find that ELE allows a higher cache rate because answer prefixes can be reused. In contrast, baseline ε-sampling can only reuse $k-1$ of all k question prompts. All answer tokens have to be generated. Further, the actual cache hit rate (solid) is very close to the theoretical cache hit rate (dashed) as SGLang effectively retrieves the relevant KV cache from memory.

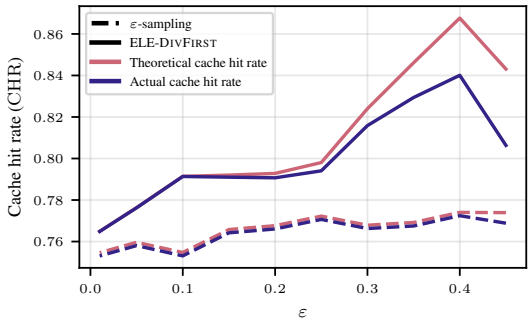


Figure 12: **Cache rate is larger for ELE than ε-sampling.** SGLang retrieves prefix cache efficiently such that the actual cache hit rate is close to the theoretical one. Tested using SGLang with $k = 8$. See Section 3 for additional details.

E.6 Results for Other Models

Table 5 shows results for Qwen2.5-7B-Instruct, Table 6 shows results for Qwen2.5-14B-Instruct, Table 7 shows results for Llama3.2-1B-Instruct, and Table 8 shows results for Llama3.2-3B-Instruct.

Table 5: Performance (maj@k and pass@k) of various methods on GSM8K, Humaneval and MMLU-Pro with the Qwen2.5-7B-Instruct model.

Method	GSM8K			Humaneval			MMLU-Pro		
	maj@2	maj@4	maj@8	pass@2	pass@4	pass@8	maj@2	maj@4	maj@8
Self-consistency	78.77	86.05	89.16	64.02	77.44	85.98	45.79	48.43	51.75
ε-sampling	81.43	87.87	89.92	72.56	81.10	**88.41**	46.19	48.31	**51.89**
ELE-DivFirst	84.91	**88.48**	**89.92**	**81.10**	**86.59**	87.80	**46.89**	**48.90**	50.71
ELE-ProbFirst	**84.15**	87.95	88.86	79.88	84.76	87.20	46.88	48.72	50.62
ELE-RandBranch	83.78	87.11	88.78	76.83	83.54	**88.41**	46.70	48.72	50.69

Table 6: Performance (maj@k and pass@k) of various methods on GSM8K, Humaneval and MMLU-Pro with the Qwen2.5-14B-Instruct model.

Method	GSM8K			Humaneval			MMLU-Pro		
	maj@2	maj@4	maj@8	pass@2	pass@4	pass@8	maj@2	maj@4	maj@8
Self-consistency	88.48	91.51	92.49	67.68	78.05	88.41	52.01	55.50	58.39
ε-sampling	89.23	91.36	**92.72**	73.17	83.54	88.41	52.94	56.48	**58.56**
ELE-DivFirst	**90.60**	**91.51**	92.34	76.22	**85.98**	88.41	54.66	56.40	58.18
ELE-ProbFirst	90.07	91.43	92.19	**76.83**	**85.98**	**89.02**	54.33	56.38	58.30
ELE-RandBranch	89.92	91.51	92.42	78.66	84.76	88.41	**54.88**	**56.52**	58.29

Table 7: Performance (maj@k and pass@k) of various methods on GSM8K, Humaneval and MMLU-Pro with the Llama3.2-1B-Instruct.

Method	GSM8K			Humaneval			MMLU-Pro		
	maj@2	maj@4	maj@8	pass@2	pass@4	pass@8	maj@2	maj@4	maj@8
Self-consistency	18.50	24.64	32.60	25.00	37.20	46.95	13.73	16.71	19.02
ε-sampling	27.75	33.89	**39.95**	37.80	48.78	54.88	18.87	20.69	21.82
ELE-DivFirst	31.39	34.65	36.85	31.71	39.63	46.95	**19.76**	**21.23**	22.15
ELE-ProbFirst	**33.21**	**36.01**	38.06	41.46	**52.44**	**59.76**	19.61	21.13	**22.95**
ELE-RandBranch	33.13	34.80	38.44	**43.29**	48.17	57.32	19.75	20.90	21.97

Table 8: Performance (maj@k and pass@k) of various methods on GSM8K, Humaneval and MMLU-Pro with the Llama3.2-3B-Instruct model.

Method	GSM8K			Humaneval			MMLU-Pro		
	maj@2	maj@4	maj@8	pass@2	pass@4	pass@8	maj@2	maj@4	maj@8
Self-consistency	43.29	54.36	65.58	46.34	57.93	70.12	24.88	29.31	33.04
ε-sampling	58.38	62.33	69.14	55.49	64.46	75.61	30.98	35.37	37.36
ELE-DIVFIRST	62.62	67.40	69.52	55.49	65.24	71.95	33.19	35.65	37.23
ELE-PROBFIRST	64.44	**68.92**	71.65	**59.15**	**69.51**	**78.66**	33.18	**35.88**	**37.53**
ELE-RANDBRANCH	**64.52**	68.31	**72.33**	57.93	65.24	73.17	**33.22**	35.60	37.49

E.7 DEEP THINK WITH CONFIDENCE (DEEPCONF) (FU ET AL., 2025)

DeepConf (Fu et al., 2025) is a test-time method that filters out reasoning traces of low quality when doing inference or after inference. We only investigate the offline version of DeepConf.

Fu et al. (2025) defines the token confidence as the negative average log-probability of the top-n tokens at a given position i

$$C_i = -\frac{1}{n}\sum_{j=1}^{n} \log P_i^{\text{top}}(j), \tag{14}$$

where n denotes the number of tokens considered and P_i^{top} returns the probabilities of top tokens ordered decreasingly at position i. The average trace confidence is simply defined as the mean over all token confidences.

Additionally, DeepConf introduces group confidence for groups of tokens, bottom 10% group confidence, lowest group confidence, and tail confidence. For voting, DeepConf introduces confidence-weighted majority voting that weights each answer by its confidence, and confidence filtering that filters the top-$p\%$ of traces.

In our experiments, we set the group size to 64 tokens (as opposed to 2048 for the original work) as we are working with reasoning problems that require fewer tokens. We experiment with $p \in \{10, 25\}\%$.

Table 9 shows results for all DeepConf experiments (Fu et al., 2025) on GSM8K (Cobbe et al., 2021) with Qwen2.5-0.5B-Instruct. Table 10 shows results for all DeepConf experiments (Fu et al., 2025) on MMLU-Pro (Wang et al., 2024c) with Qwen2.5-0.5B-Instruct. We include the best results achieved in Table 1.

Table 9: Results for DeepConf (Fu et al., 2025) on GSM8K (Cobbe et al., 2021) with Qwen2.5-0.5B-Instruct. See Appendix E.7 for details.

Method	2	4	8
Mean Confidence			
Weighted	0.2002	0.2578	0.3298
Top Percent Weighted (10%)	0.1948	0.2532	0.2616
Top Percent Weighted (25%)	0.2153	0.2199	0.2570
Top Percent (10%)	0.2115	0.2343	0.2509
Top Percent (25%)	0.2055	0.2153	0.2464
Tail Mean Conf (64 Tokens)			
Weighted	0.2085	0.2472	0.3222
Top Percent Weighted (10%)	0.2002	0.2199	0.2350
Top Percent Weighted (25%)	0.2032	0.2077	0.2456
Top Percent (10%)	0.1941	0.2146	0.2358
Top Percent (25%)	0.1903	0.2161	0.2525
Tail Mean Conf (10%)			
Weighted	0.1782	0.2252	0.3086
Top Percent Weighted (10%)	0.1865	0.2077	0.1971
Top Percent Weighted (25%)	0.1676	0.1956	0.2055
Top Percent (10%)	0.1638	0.1850	0.2077
Top Percent (25%)	0.1804	0.1933	0.2039
Min Sliding Mean Conf (64 Tokens)			
Weighted	0.1895	0.2563	0.3283
Top Percent Weighted (10%)	0.1873	0.2252	0.2464
Top Percent Weighted (25%)	0.2092	0.2199	0.2418
Top Percent (10%)	0.1842	0.2206	0.2593
Top Percent (25%)	0.2077	0.2244	0.2214
Bottom Sliding Mean Conf (10%, 64 Tokens)			
Weighted	0.2077	0.2631	**0.3336**
Top Percent Weighted (10%)	0.1979	0.2206	0.2661
Top Percent Weighted (25%)	0.1774	0.2183	0.2487
Top Percent (10%)	0.1979	0.2237	0.2449
Top Percent (25%)	0.1676	0.2183	0.2441
Bottom Sliding Mean Conf (50%, 64 Tokens)			
Weighted	0.1895	**0.2691**	0.3139
Top Percent Weighted (10%)	0.1918	0.2206	0.2578
Top Percent Weighted (25%)	**0.2168**	0.2290	0.2638
Top Percent (10%)	0.1850	0.2115	0.2623
Top Percent (25%)	—	0.2199	0.2631

Table 10: Results for DeepConf (Fu et al., 2025) on MMLU-Pro (Wang et al., 2024c) with Qwen2.5-0.5B-Instruct. See Appendix E.7 for additional information.

Method	2	4	8
Mean Confidence			
Weighted	0.1406	**0.1558**	0.1607
Top Percent Weighted (10%)	0.1371	0.1451	0.1544
Top Percent Weighted (25%)	0.1363	0.1524	0.1439
Top Percent (10%)	0.1433	0.1493	0.1471
Top Percent (25%)	0.1419	0.1439	0.1503
Tail Mean Conf (64 Tokens)			
Weighted	0.1381	0.1518	0.1635
Top Percent Weighted (10%)	0.1362	—	0.1478
Top Percent Weighted (25%)	0.1410	0.1468	0.1516
Top Percent (10%)	0.1409	0.1435	0.1479
Top Percent (25%)	0.1388	0.1464	0.1470
Tail Mean Conf (10%)			
Weighted	0.1410	0.1503	0.1600
Top Percent Weighted (10%)	0.1377	0.1434	0.1410
Top Percent Weighted (25%)	0.1385	0.1376	0.1446
Top Percent (10%)	0.1341	0.1420	0.1497
Top Percent (25%)	0.1394	0.1385	0.1444
Min Sliding Mean Conf (64 Tokens)			
Weighted	0.1401	0.1497	**0.1676**
Top Percent Weighted (10%)	0.1418	0.1470	0.1469
Top Percent Weighted (25%)	0.1347	0.1465	0.1456
Top Percent (10%)	0.1411	0.1478	0.1499
Top Percent (25%)	0.1418	0.1457	0.1470
Bottom Sliding Mean Conf (10%, 64 Tokens)			
Weighted	0.1395	0.1547	0.1618
Top Percent Weighted (10%)	0.1389	0.1432	0.1490
Top Percent Weighted (25%)	0.1375	0.1463	0.1490
Top Percent (10%)	0.1426	0.1462	0.1495
Top Percent (25%)	0.1394	0.1418	0.1476
Bottom Sliding Mean Conf (50%, 64 Tokens)			
Weighted	**0.1441**	0.1501	0.1646
Top Percent Weighted (10%)	0.1419	0.1523	0.1526
Top Percent Weighted (25%)	0.1390	0.1467	0.1489
Top Percent (10%)	0.1365	0.1439	0.1511
Top Percent (25%)	0.1391	0.1462	0.1461

E.8 SELF-CERTAINTY (KANG ET AL., 2025)

Self-certainty (Kang et al., 2025) is a method for response evaluation and selection. More specifically, it is a metric that takes the probability distribution of the LLM into account to estimate response quality without additional compute. The general idea is that a larger self-certainty also corresponds to an improved response accuracy. Self-certainty is defined as the point-wise Kullback-Leibler (Kullback & Leibler, 1951) divergence between a uniform distribution and the next-token distribution, averaged over the sequence length,

$$\text{Self-Certainty}(\mathbf{x}_{1:T}) \coloneqq -\frac{1}{VT}\sum_{t=1}^{T}\sum_{v=1}^{V}\log(V \cdot p(v \mid \mathbf{x}_{<t})), \tag{15}$$

where $V = |\mathcal{V}|$ is the size of the vocabulary.

Kang et al. (2025) also identifies that score-based voting methods suffer from sensitivity to score scaling. Therefore, they propose an approach inspired by Borda-counts. First, they rank N outputs of models by confidence, obtaining a ranking $(r_1, r_2, \ldots, r_N)$. Then, they assign votes to these ranked outputs using the formula

$$v(r) = (N - r + 1)^p, \tag{16}$$

where r is the rank of the output and p is the voting power.

Table 11 shows results for all self-certainty experiments on GSM8K (Cobbe et al., 2021) with Qwen2.5-0.5B-Instruct. Table 12 shows results for all self-certainty experiments on MMLU-Pro (Wang et al., 2024c) with Qwen2.5-0.5B-Instruct.

Table 11: Results for self-certainty (Kang et al., 2025) on GSM8K (Cobbe et al., 2021) with Qwen2.5-0.5B-Instruct. $\arg\max, p = 0.0$ denotes the case *without voting* and choosing the $\arg\max$ sequence for self-certainty. See Appendix E.8 for additional information.

Voting power p	$\arg\max, p = 0.0$	$p = 0.1$	$p = 0.3$	$p = 0.5$	$p = 0.7$	$p = 0.9$
Voting $k = 2$	**20.09**	19.94	19.94	20.02	19.94	19.94
Voting $k = 4$	23.81	25.09	24.87	24.41	24.64	**25.32**
Voting $k = 8$	28.73	29.80	**32.22**	31.01	30.17	31.54

Table 12: Results for self-certainty (Kang et al., 2025) on MMLU-Pro (Wang et al., 2024c) with Qwen2.5-0.5B-Instruct. $\arg\max, p = 0.0$ denotes the case *without voting* and choosing the $\arg\max$ sequence for self-certainty. See Appendix E.8 for additional information.

Voting power p	$\arg\max, p = 0.0$	$p = 0.1$	$p = 0.3$	$p = 0.5$	$p = 0.7$	$p = 0.9$
Voting $k = 2$	14.42	14.26	14.21	14.11	14.45	**14.51**
Voting $k = 4$	15.13	15.40	**15.92**	15.90	15.10	15.91
Voting $k = 8$	13.48	14.25	14.66	**14.68**	14.28	14.14