Fast Inference with Dynamic Tree Speculative Decoding

Vrushank Gunjur
Department of Computer Science
Stanford University
vrushank@cs.stanford.edu

Alexander Waitz
Department of Computer Science
Stanford University
waitz@cs.stanford.edu

Abstract

Speculative decoding uses a small "draft" model to speculate a larger "oracle" model and increase generation speed. In this context, once approach to address the issue of compounding draft-oracle mismatch rates is to scale the number of speculative samples at each sequence position. However, prior work scales speculation at a fixed rate, which is not necessarily optimal. We introduce an entropy-informed technique that scales sampling according to draft model uncertainty. This method uses logit entropy as a proxy for model uncertainty, with high entropies corresponding to high token-level uncertainty. With this knowledge, we can dynamically shape the speculation tree to maximize draft-oracle match rates. We demonstrate that, with relatively low overhead, we can better allocate speculative tokens, leading to an overall decrease in generation time of ~7% when compared to vanilla speculative decoding.

1 Introduction

Scaling inference-time compute of large language models (LLMs) has been shown to significantly improve model accuracy. Some existing approaches using this concept include Tree of Thought [12] and Speculative Decoding [5]. In these approaches, smaller models are used to speed up overall generation time. Quantifying or estimating the success of these small models is crucial for efficiently implementing these approaches. However, it is currently handled by static heuristics or expensive forward passes.

To address these limitations, we propose using the Shannon entropy of model output logits to measure uncertainty. Increased entropy indicates a greater degree of “surprise” in the next-token probability distribution. Besides reflecting the true statistical uncertainty of the next-token, this methodology is cheap since token logits are already generated during model generation.

In this work, we first demonstrate that there exists an actionable relationship between model accuracy and logit entropy. Second, we operationalize this principle by implementing an accelerated variant of tree speculative decoding, building on the ideas proposed in Staged Speculative Decoding [9]. Rather than constructing a speculative tree batch of fixed topology, we dynamically adjust the branching factor as a function of entropy. The exponential characteristic of a tree batch is well-suited for exponential decrease in draft-oracle match rates, while entropy-informed branching “grows” the tree along optimal paths.

With this approach, we demonstrate a significant generation speed improvement compared to naive and fixed-tree speculative decoding.

2 Related Work

Repeated sampling of small models to meet the accuracy of larger models or boost their performance is a concept that’s recently gained significant traction. We present some selected techniques that are of relevance to our approach.

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

2.1 Shannon Entropy

In this work, we posit that entropy can be used as a measure of model uncertainty at the token level. Intuitively, Shannon entropy is a measure of uncertainty or "surprise" in a probability distribution [8]. Let $\mathcal{M}$ be some model. Let $\mathcal{Y}$ be a token prefix. Let $\mathcal{X}$ be a random discrete variable representing the next token prediction where $\mathcal{X}$ is distributed by the model probabilities (i.e. post-softmax logits). Thus, we can define $p_{\mathcal{M}}(x|\mathcal{P})$, for some $x \in \mathcal{X}$, as the probability that token x is the next token, given some prefix $\mathcal{P}$. Given this formulation, we can formally define the Shannon entropy for the next-token logits:

$$H(\mathcal{X}|\mathcal{P}) = -\sum_{x\in\mathcal{X}} p_{\mathcal{M}}(x|\mathcal{P}) \log p_{\mathcal{M}}(x|\mathcal{P})$$

2.2 Model Reasoning

Model reasoning refers to the use of multi-step problem-solving strategies used to tackle more complex problems. Some existing approaches to this challenge include Chain-of-Thought prompting [10], or Tree-of-Thoughts [12]. The common theme in these approaches is to re-prompt the model with a more complex set of inputs which are aggregated to form a more accurate response. Using this approach, it is possible to significantly improve the accuracy of small models while working in a resource-constrained setting.

A recent open-source project called Entropix incorporates entropy into pre-existing model reasoning approaches [11]. Rather than re-prompt at a fixed rate or rely on value functions dependent on model voting, Entropix asserts that the mean and variance of generation entropy can be used to more optimally sample models at inference-time.

We preliminarily validated the relationship between model uncertainty and logit entropy. We ran one-shot inference with Llama3.2-1B [1] on a subset of GSM8K [4], collecting data on output logit entropy and answer accuracy. We observed a clear correlation between the entropy generated by correct and incorrect outputs, as illustrated in Figure 3. This validates the notion that entropy can be used to meaningfully discern between desirable and undesirable outputs, at least for this domain.

2.3 Speculative Decoding

Speculative decoding is a two step process in which a small draft model auto-regressively generates a series of "speculated" tokens and a larger oracle model verifies them. In Staged Speculative Decoding, as proposed by Spector & Ré [9], a fixed tree of speculated tokens is generated. Thus, the number of speculated candidate tokens at each subsequent sequence position is exponentially greater than the previous. This approach effectively addresses the issue of compounding draft-oracle disagreement with minimal additional overhead. This paper also proposes other optimizations, but we focus on expanding on the idea of providing multiple options at each index with a tree batch.

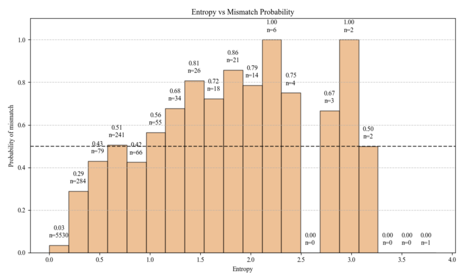
Combining the tree batch mechanism and entropy-based classification of "good" and "bad" model outputs, we propose using dynamically shaped trees instead of the static branching factors and depths from this paper.

3 Methods and Technical Approach

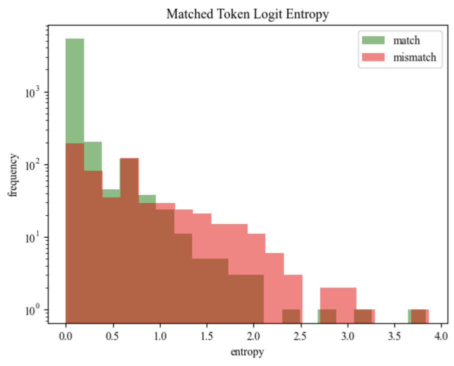
3.1 Validation and Naive Approaches

To validate that entropy is actionable in the context of speculative decoding, we first ran some preliminary experiments examining the entropy of logits from the speculative (draft) model. We recorded the logits of all matched and mismatched tokens to calculate their respective entropies. We found that the mismatch probability of a given speculated logit increases with entropy, even though the number of mismatched tokens is much smaller than the number of matched tokens 1.

With this knowledge, we implemented an initial version of speculative decoding where the number of speculated tokens is decided on the fly. If the probability of mismatch is higher than the probability of a match (corresponding to entropies higher than 1), we return early without continuing to speculate. We call this method "early stopping". An extension of this approach naively provides multiple options for



(a) Probability of token mismatch



(b) Match and mismatch entropies

Figure 1: Speculated entropy frequencies according to match and mismatched labels

the final index of our speculated sequence by copying the entire sequence down the batch dimension and replacing the last token with the top-k probable tokens. We then return early from the speculation. We duplicate the final index because the final sequence element typically contains the token of greatest entropy in the "early stopping" approach. We call this approach "naive width".

3.2 Efficient Tree Batches

To improve upon the naive method of providing multiple options at an index, we must allow for tree batches to efficiently be passed into the model. Although there are several ways to potentially implement tree batches, one must be careful to pick one that is sufficiently efficient; otherwise the speedups produced by an increased draft-oracle match rate will be lost to the additional overhead.

The naive width approach suffers from massive redundancy in the final speculated batch. To remedy this, We instead use the same set of keys and values for the entire shared prefix and only generate additional ones for the new options we propose at each index. This eliminates unnecessary replication of work and data. To achieve this, we implement tree batches by flattening the speculation tree into a single sequence and constructing an attention mask such that each token only attends to itself and its parents in the tree. We also simultaneously construct positional embeddings that reflects each token's true position in the sequence, as opposed to its position in the flattened tree sequence. An illustration of the attention mask constructed by this process is found in Figure 4.

With this methodology, we only keep a single copy of the sequence prefix at any given position, irrespective of the number of samples drawn, to avoid unnecessary redundancy. Thus, each call to the draft model to speculate a new set of tokens creates an empty tree rooted at the last token of the conditioning. We then recursively create the attention mask and position embedding matrices together and perform a forward pass with them. The resulting logits are then used to grow the tree by dynamically deciding a branching factor b at each leaf and selecting b top tokens to include. We can think of this as a generalization of naive speculative decoding, where the naive approach is a special case with a fixed branching factor of 1. Selecting more than one candidate at a new branch is essentially free since we are simply performing a top-k operation on the logits we already calculated. We also provide an avenue to control the depth of the tree as a function of entropy, which is equivalent to setting the branching factor to 0.

3.3 Applications to Speculative Decoding

The dynamic trees that are enabled by our approach can have both dynamic depth and width. Intuitively, we would want to increase depth and keep a narrow tree if the speculative model is more confident, and we would want a shallow but wide tree if the model is less confident. Thus, we must choose when to stop growing the tree, and when we grow the speculative tree to a new index, we must choose the branching factor for each leaf. These are hyperparameters that we searched through with some intuition about the distribution of entropies on matched and mismatched tokens. We provided scaffolding to specify drop-in functions of entropy to control both the depth and width of our tree that would make it easy to run experiments on these hyperparameters.

Algorithm 1 Dynamic Tree Speculative Decoding

```
Require: $\mathbf{x}_{\text{input}}$ (input sequence), $L$ (generation length), $M_d$ (draft model), $M_o$ (oracle model)
Ensure: $\mathbf{x}_{\text{output}}$ (output sequence)
 1: $\mathbf{x}_{\text{output}} \leftarrow \mathbf{x}_{\text{input}}$
 2: while $|\mathbf{x}_{\text{output}}| - |\mathbf{x}_{\text{input}}| < L$ do
 3:     // Speculative tree from draft model
 4:     $\mathcal{T} \leftarrow M_d$.speculate$(\mathbf{x}_{output})$
 5:     // Sequence, mask, position matrix
 6:     $(\mathbf{s}, \mathbf{m}, \mathbf{P}) \leftarrow$ FlattenTree$(\mathcal{T})$
 7:     // Logits via forward pass of oracle model
 8:     $\mathbf{z} \leftarrow M_o(\mathbf{s}, \mathbf{m}, \mathbf{P})$
 9:     // Position index for initial token in $\mathbf{x}_{\text{input}}$
10:     $p \leftarrow |\mathbf{x}_{\text{input}}|$
11:     for $d = 1$ to Depth$(\mathcal{T})$ do
12:         $t_{\text{sampled}} \sim$ Sample$(\mathbf{z}[p])$
13:         // Match with candidate at depth $d$
14:         if $t_{\text{sampled}} \in \mathcal{T}$.keys() then
15:             $\mathbf{x}_{\text{output}} \leftarrow \mathbf{x}_{\text{output}} \cup \{t_{\text{sampled}}\}$
16:             // Update $p$ to matched token position
17:             $p \leftarrow$ Position$(t_{\text{sampled}}, \mathbf{s})$
18:             // Update tree to child of matched token
19:             $\mathcal{T} \leftarrow \mathcal{T}[t_{\text{sampled}}]$
20:             if $\mathcal{T} = \emptyset$ then
21:                 $t_{\text{sampled}} \sim$ Sample$(\mathbf{z}[p])$
22:                 $\mathbf{x}_{\text{output}} \leftarrow \mathbf{x}_{\text{output}} \cup \{t_{\text{sampled}}\}$
23:                 break
24:             end if
25:         else
26:             $\mathbf{x}_{\text{output}} \leftarrow \mathbf{x}_{\text{output}} \cup \{t_{\text{sampled}}\}$
27:             break
28:         end if
29:     end for
30: end while
31: return $\mathbf{x}_{\text{output}}$
```

Using these functions, we created many version of our system. These versions employed various strategies, such as being aggressive with the branching factor but remaining static in depth, or varying both dynamically.

4 Experiments and Results

We formulate our experiments by running inference on the humaneval dataset [2] using speculative decoding with a temperature of 0.4. Each approach is evaluated on 45 prompts, with the relevant metrics being captured on a per-prompt basis. GPT2-large [7] is used as the oracle model and GPT2 as the draft model. We built ontop of the boilerplate code provided in CS 229S Assignment 2 [6].

Since we were only interested in examining the impact of tree batches and not hierarchical speculation as in Spector and Re, we compared our results to naive speculative decoding without tree batches. To provide an estimate of the improvement that dynamic trees provide over static ones, we also used our own implementation to record the performance of our dynamic approach against a static tree with fixed branching factor of 2, demonstrating the same exponential increase in tokens per index.

In figures 5 and 6, we visualize the time gains on the y-axis by subtracting the generation times on a per-prompt basis. The x-axis is a measure of reduction in wasted work, which we represent by subtracting the number of wasted speculated forward passes in the baseline which would have been thrown out due to an earlier mismatch with the number of wasted layers in the new approach. For both axes, a value greater than 0 is desirable (marked by the dashed black lines) as it indicates a performance increase compared to the baseline. The average time reductions are represented by the dashed red lines.

We see that the number of wasted passes is usually strongly correlated with the time savings (with the exception of the naive width approach, which introduces high overhead and disproportionately impacts long generations).

We report results for the most promising candidates of the depth and width functions we experimented with. We find that the best-performing approach was to use tree batches with dynamic width and a fixed depth of four. This approach saved an average of 2.07 seconds per prompt (Fig 5 part c). This result is summarized in Fig 2 below, along with the tokens per second improvement to isolate the length of the generation.

	Baseline	Static Tree	Simple Cutoff	Dyn Width + Depth	Dyn Width Only
Avg Time Savings	0%	2.3%	4.43%	6.65%	**7.16%**
Tokens/sec	10.078	10.351	10.556	10.83	**10.88**

Figure 2: Performance results from speculative decoding variants.

We see that using static tree batches (with a branching factor of 2, depth 4) is preferable to vanilla speculative decoding by providing exponentially more options at each index, saving 1.07 seconds per generation on average, Fig 6. More importantly, we demonstrate that using dynamic entropy-based strategies produced further improvement, boosting time savings an additional 4.86%.

While the reported results reflect the best-performing hyperparameters identified in our search, limited time and resources prevented us from conducting an exhaustive grid search. The best-performing "dynamic width" result from Fig 2 uses the following expression to calculate the branching factor at a given node, while keeping the depth fixed:

$$\mathcal{B}(\text{entropy}) = \begin{cases} 1 & \text{if entropy} < 0.02, \\ 2 & \text{if } 0.02 \leq \text{entropy} < 1, \\ \min(\lceil \text{entropy} \times 4 \rceil, 7) & \text{if entropy} \geq 1. \end{cases}$$

5 Discussion

From these results, it's clear that using tree batches with dynamic cutoffs provides performance benefits over vanilla speculative decoding. We also show that dynamically varying the branching factor performs better than keeping a static branching factor. While we weren't able to exhaustively search through static branching factors, we believe our chosen hyperparameters to be an approximate representation of the general performance of this method and minimally a lower bound on its potential.

We experimented with various formulations for branching factor and depth as a function of entropy: we built on the idea that entropies greater than 1 are more likely to be mismatched than matched, and that the number of options should scale with the magnitude of the entropy. Following this train of thought, we opted to not explicitly fix the number of nodes in the tree, inspired by the success of the early stopping approach we attempted at first which relied on a highly variable number of speculated tokens. However, a likely optimization is to set an explicit maximum to the total number of nodes (a speculation "budget") instead of a cap on the depth and width individually.

There are a few possibilities for why we saw a relative decline in results when varying both the depth and width dynamically. One (likely) possibility is that we simply didn't explore the hyperparameter space sufficiently due to time and resource constraints. Alternatively, our implementation might be skewed to perform better in the static depth setting. For example, when we grow the speculative tree and decide to stop growing deep on some paths, we still must repeatedly unflatten and flatten all "done" branches when continuing the speculation for the other paths. This means we also perform forward-passes along those paths despite not using the logits. We also suspect that using a dynamic depth prevents us from capitalizing on the benefits of mitigating mismatch by going "wide" at an index since we instead end speculation early. In the context of a defined speculation budget, we could reformulate dynamic depth as setting the branching factor to 0 when the desired branching factor exceeds our speculation budget; in other words, when the uncertainty is too great to reasonably account for by re-sampling.

More broadly, there were several key implementation details that are relevant to contextualizing our results. As an alternative to performing entirely isolated self-attention on a copy of the main sequence in speculative generation, we could have conducted cross-attention into the main KV cache corresponding to the shared prefix and self-attention on a temporary cache for the tree batch itself, as is done in Spector and Re [9]. Since we run our baselines with KV caching disabled for the speculative model, we skip this optimization. Moreover, when constructing the flattened tree, there were several traversal orders we could have opted for. We chose for in-order traversal for ease of implementation. However, there were alternative traversal orders, such as level-order traversal, which would have enabled us to merge identical tokens at a given index position to avoid repeated work.

Another important conclusion from the results is that even simple inclusions of entropy-based techniques can have significant impact on performance. Simply choosing speculation length based on an appropriate entropy cutoff provided the most consistent improvement across all prompts (Fig 5a), with only one outlier prompt out of 45 actually performing significantly worse than baseline.

Overall, we find that using entropy-based approaches to speculative decoding affords a performance benefit, even without using the most optimized implementation.

6 Conclusion

We demonstrate that speculative decoding with dynamic width outperforms naive speculative decoding and other entropy-driven variants. We believe that this approach succeeds because it allows us to capitalize on the increased match rate afforded by tree batches while being responsive to the model's confidence. Overall, this results in a ~7% reduction in average response generation time / increase in tokens per second. This result demonstrates entropy's viability as a tractable and useful uncertainty measure in inference scaling.

6.1 Future Work

In the future, we would like to explore additional algorithmic and hardware accelerations. This includes a more sophisticated implementation of tree batching, potentially incorporating entropy variance into our uncertainty metric [11], and using hierarchical draft models as in [9]. Projects such as Sequoia [3] have developed tree-batch implementation with decreased redundancy and hardware-aware configuration. Moreover, it implements an overall "token budget" for speculative tree generation, which we believe is helpful for hardware-aware optimizations.

Moreover, we would like to gather more extensive data on the relationship between draft-model agreement and entropy. The data in our paper was generated from benchmarks that are limited in scope. Moreover, we used GPT2 and GPT2-large, models that are relatively behind state-of-the-art technology. With more data, we believe that we can significantly improve our technique by developing a branching factor equation that is more empirically driven (or perhaps one that is itself learned). Additionally, we neglect to incorporate oracle logit entropy in our methods. If both the draft and oracle entropies are high at a sequence position, we can interpret this as both models being uncertain about the next token. In this case, we believe that we can loosen the draft-oracle match criteria by either defining a confidence interval or taking multiple samples from the oracle logits. Since the oracle model is also uncertain, any of its top-k predictions are reasonable selections, for some empirically determined k. With an appropriate function from entropy to k, we believe we could improve inference with little cost to output correctness.

Finally, we would also like to perform a more complete hyperparameter sweep to better understand the optimal behavior of our algorithm when applied to more optimized models and implementations.

References

[1] Meta AI. Llama 3.2-1b: Multilingual large language model. `https://huggingface.co/meta-llama/Llama-3.2-1B`. Accessed: 2024-12-09.

[2] 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, 2021.

[3] Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. Sequoia: Scalable, robust, and hardware-aware speculative decoding, 2024.

[4] 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, 2021.

[5] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding, 2023.

[6] Azalia Mirhoseini. Assignment 2 boilerplate code. CS 229S, Department of Computer Science, Stanford University. Accessed: 2024-12-09.

[7] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019. Accessed: 2024-12-09.

[8] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 623–656, 1948. Accessed: 2024-12-09.

[9] Benjamin Spector and Chris Re. Accelerating llm inference with staged speculative decoding, 2023.

[10] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

[11] xjdr alt. Entropix: Entropy based sampling and parallel cot decoding. `https://github.com/xjdr-alt/entropix`. Accessed: 2024-12-09.

[12] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023.

7 Supplementary Material / Appendix

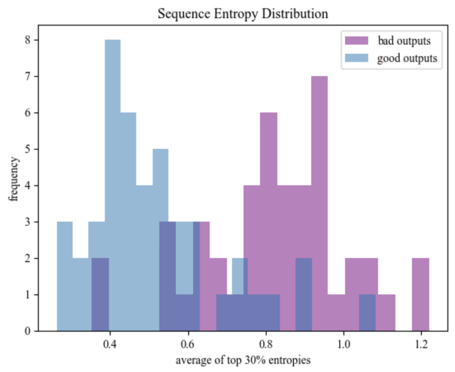
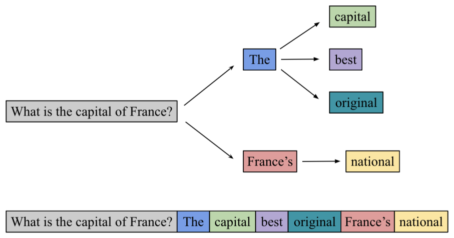
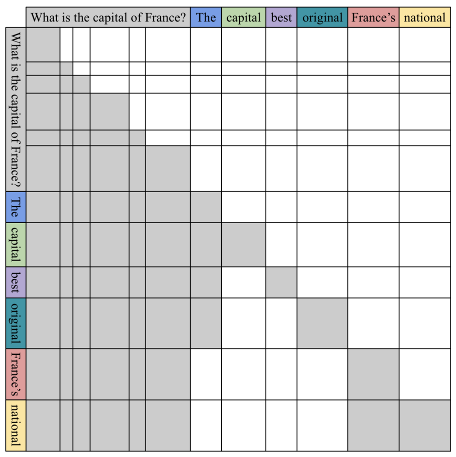


Figure 3: Logit entropy distributions for correct and incorrect model generations

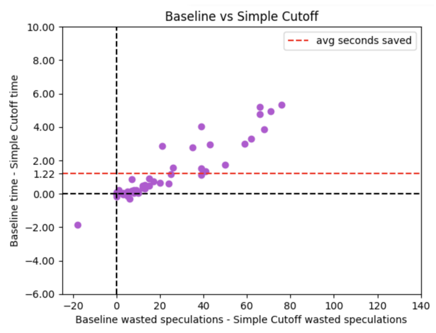


(a) Example tree batch

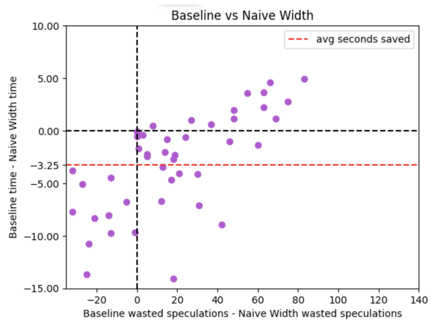


(b) Tree batch attention mask

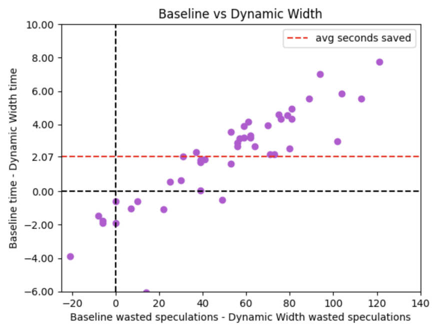
Figure 4: Tree batch and corresponding attention mask



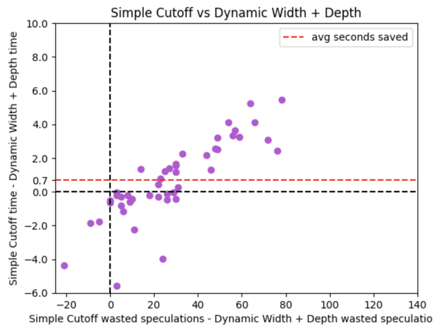
(a) Baseline vs returning early if entropy > 1



(b) Baseline vs copying down the batch dimension for more options at singular high entropy index



(c) Baseline vs tree batches with dynamic branching factor and fixed depth of 4



(d) Baseline vs tree batches with dynamic branching factor and depth

Figure 5: Baseline (vanilla speculative decoding with 4 speculated tokens) vs entropy-based dynamic approaches

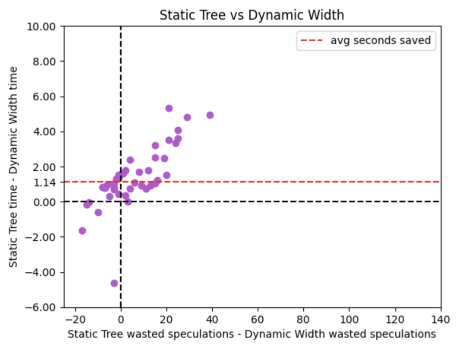


Figure 6: Static tree with fixed branching factor of 2 vs dynamic branching factor tree