SEQUOIA: Scalable, Robust, and Hardware-aware Speculative Decoding

Zhuoming Chen*†, Avner May*‡, Ruslan Svirschevski*§,
Yuhsun Huang†, Max Ryabinin‡, Zhihao Jia†, and Beidi Chen†♯

†Carnegie Mellon University
‡Together AI
§Yandex
♯Meta AI
{zhuominc,yuhsunh,zhihaoj2,beidic}@andrew.cmu.edu,
{avner,mryab}@together.ai, ruslansv@gmail.com

February 20, 2024

Abstract

As the usage of large language models (LLMs) grows, performing efficient inference with these models becomes increasingly important. While speculative decoding has recently emerged as a promising direction for speeding up inference, existing methods are limited in their ability to scale to larger speculation budgets, and adapt to different hyperparameters and hardware. This paper introduces SEQUOIA, a scalable, robust, and hardware-aware algorithm for speculative decoding. To attain better scalability, SEQUOIA introduces a dynamic programming algorithm to find the optimal tree structure for the speculated tokens. To achieve robust speculative performance, SEQUOIA uses a novel sampling and verification method that outperforms prior work across different decoding temperatures. Finally, SEQUOIA introduces a hardware-aware tree optimizer that maximizes speculative performance by automatically selecting the token tree size and depth for a given hardware platform. SEQUOIA improves the decoding speed of Llama2-7B, Llama2-13B, and Vicuna-33B on an A100 GPU by up to 4.04×, 3.84×, and 2.37×, and Llama2-70B offloading speed by up to 10.33× on an L40.

1 Introduction

As large language models (LLMs) gain widespread adoption [2, 6, 42], efficiently serving these LLMs becomes increasingly important. However, accelerating LLM inference is challenging since generating a single new token requires accessing all parameters of the LLM [33]. As a result of this I/O bottleneck, the hardware is poorly utilized during generation. This problem is exacerbated in both small-batch and offloading-based inference settings, where generating one token takes as much time as processing a prompt with hundreds or thousands of tokens on modern GPUs.

To address this challenge, recent work has introduced *speculative decoding* to accelerate LLM inference while preserving the LLM's output distribution [4, 24, 27, 39]. These approaches leverages one or multiple *draft models* to predict the LLM's output; the predictions are organized in a *token tree*, whose nodes represent different sequences of speculated tokens. The correctness of these speculated tokens are then *verified in parallel* through a single forward pass of the LLM. Using a token tree—instead of sequence—can increase the number of tokens accepted by the LLM by providing several options for each token position.

While there are substantial studies on tree-based speculative decoding methods [27, 39], we see in our experiments that they have important limitations. First, we observe that existing token tree construction algorithms perform well for small token trees but are sub-optimal for large tree sizes. For example, SpecInfer constructs a token tree using k independent sequences, a topology which is bounded in the expected number

*Equal contribution.

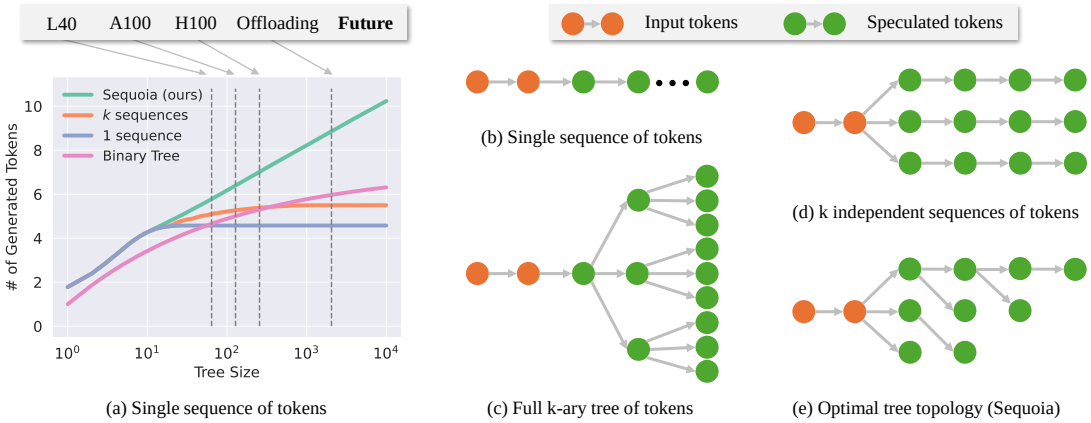


Figure 1: SEQUOIA is a scalable method for speculative decoding. Left: SEQUOIA tree construction algorithm is able to generate trees whose average number of generated tokens (after verification) continues to grow with the tree size, while existing tree structures asymptote. This allows SEQUOIA to perform much better than existing methods in very memory-bound regimes like offloading. Right: A visualization to contrast SEQUOIA tree structure with other common handcrafted ones.

of tokens it can accept, regardless of the tree size (Figure 1). Second, we observe that existing token tree sampling and verification algorithms are unable to perform well across inference hyperparameter configurations; for example, SpecInfer [27] and SpecTr [39] often perform poorly at low temperature (Figure 3), due to the fact that they can repeatedly sample an incorrect token with high draft model probability. Lastly, we observe that existing systems are unable to effectively optimize—for any hardware configuration—the size and shape of their speculated trees (Table 3). This is because existing models describing the speedup from speculative decoding [24, 39] assume verification time is constant, which does not hold for large speculated trees, thereby making these models ineffective for selecting the best tree dimensions.

In this paper, we aim to answer the following research question: *how can we design an optimal tree-based speculative decoding method to maximize speedups on modern hardware?* Realizing this goal requires addressing several technical challenges. First, for any tree size and depth, we must be able to efficiently search the exponentially large space of tree topologies to find the one that maximizes the expected number of generated tokens. Second, we must design a tree sampling and verification procedure which performs well across inference hyperparameters, and avoids repeatedly sampling incorrect tokens, while maintaining the correct output distribution. Third, for any hardware, we must be able to choose the tree size and depth that will provide the largest speedup, when paired with the optimal tree of those dimensions.

This paper introduces SEQUOIA, a scalable, robust, and hardware-aware speculative decoding algorithm. As shown in Figure 1, SEQUOIA can attain up to 10× speedups over incremental decoding and introduces several key techniques to address the aforementioned challenges.

- In Section 3.1, to solve the first challenge, we formulate tree construction as a constrained optimization problem and employ a dynamic programming algorithm to discover the *optimal* speculative token tree. We demonstrate, theoretically and empirically, that the number of tokens generated with this tree structure is unbounded, growing roughly logarithmically with the size of the tree.
- In Section 3.2, to address the second challenge, we build upon the SpecInfer [27] algorithm by performing sampling *without replacement* from the draft model—thereby preventing the draft model from making the same mistake twice, while maintaining the target model's output distribution. We prove that this new sampling and verification method is able to attain high acceptance rates at both high and low temperatures, and validate this claim empirically.
- In Section 3.3, to address the final challenge, we propose a hardware-aware tree optimizer, which treats the verification time as a hardware-dependent function of the number of tokens being verified, and uses this function to solve for the optimal tree shape and depth. We show this approach yields speedups relative to hardware-agnostic methods.

In Section 4, we perform extensive end-to-end experiments and ablation studies to demonstrate the effectiveness of SEQUOIA. We implement SEQUOIA on top of Hugging Face (and Accelerate) [15, 44] with CUDA Graphs [30, 31]. We show that SEQUOIA achieves up to 4.04× speedup for Llama2-7B on a single A100 GPU and 10.33× for Llama2-70B in the offloading setting on an L40 GPU. We also present ablation studies to show that: (1) the SEQUOIA tree structure can generate up to 33% more tokens per decoding step compared to k independent sequences (tree size ≤ 512), demonstrating better scalability; (2) the SEQUOIA sampling and verification algorithm is robust to the choice of hyperparameters (temperature, top-p), providing up to 65% and 27% speedup compared to SpecInfer and top-k sampling and verification algorithms, respectively; (3) the SEQUOIA hardware-aware tree optimizer can automatically select the best tree size and depth for different hardware.

2 Background

Here, we review tree-based speculative decoding methods. In particular, we discuss the way existing methods choose the structure of the speculation tree (Section 2.1), the algorithms they use to sample and verify the token trees (Section 2.2), and the way these methods can automatically select the shape of the token tree (Section 2.3).

2.1 Tree construction

The primary tree structure used by existing methods is one composed of k independent sequences of length L that branch from the tree root (which corresponds to the current prefix $[x_1,x_2,...,x_{n-1}]$). The SpecTr paper additionally considers arbitrary branching patterns $(k_1,k_2,...,k_t)$, but says that this did not perform better in their experiments than independent sequences. Medusa constructs a full k-ary tree, which increases the success rate at each layer but cannot form a deep tree under moderate token budgets [3].

2.2 Tree sampling and verification

We now review how SpecInfer [27], SpecTr [39], naive sampling [27], and top-k sampling[1] perform token tree sampling and verification. With regard to sampling, SpecInfer, SpecTr, and naive sampling all perform i.i.d. sampling with replacement from the draft model, while top-k sampling selects the top-k highest probability tokens from the draft model. In terms of verification, SpecInfer and SpecTr compare the draft and target model probabilities for the sampled tokens to decide which (if any) to accept; naive and top-k sampling, on the other hand, sample a token from the *target model* distribution and accept it if it corresponds to one of the tokens from the speculated tree. These methods all verify a speculated token tree in a recursive manner—starting at the root of the tree—differing only in the verification algorithm they apply at each node.

SpecInfer: The SpecInfer method iteratively verifies tokens that were sampled from one or more draft models. Like the original speculative decoding method [24], it compares the draft model probabilities to those from the target model to decide if to accept. Note that while the SpecInfer method allows sampling from k different draft models to generate k children for a node, in this work we consider the more common setting where only one draft model is available. Therefore, we compare with the version of SpecInfer which samples from a single draft model k times instead. We present pseudocode for SpecInfer in Appendix B.2.

SpecTr: The SpecTr algorithm is similar in spirit to the SpecInfer algorithm. It iterates through the children of a node, and uses a sampling procedure to decide if to accept a child, in such a way that the output distribution is unchanged. One important property of this algorithm is that it is within a factor of $(1-1/e)$ of the best possible verification algorithm (meaning, the one with highest possible acceptance rate). For brevity, we refer users to Algorithm 3 in the SpecTr paper for the exact pseudocode for this algorithm.

Naive sampling and top-k sampling: Given a node in a token tree, the verification algorithm for naive sampling and top-k sampling first samples from the *target model's* distribution $\mathcal{P}(\cdot \mid x_{<n})$ at that node, and then accepts this sample if it is equal to one of the children of that node. This verification algorithm trivially maintains the target model output distribution—regardless of how the token tree was generated—given that one always samples

[1] Top-k sampling is an improved version of naive sampling which we introduce as a baseline in this work.

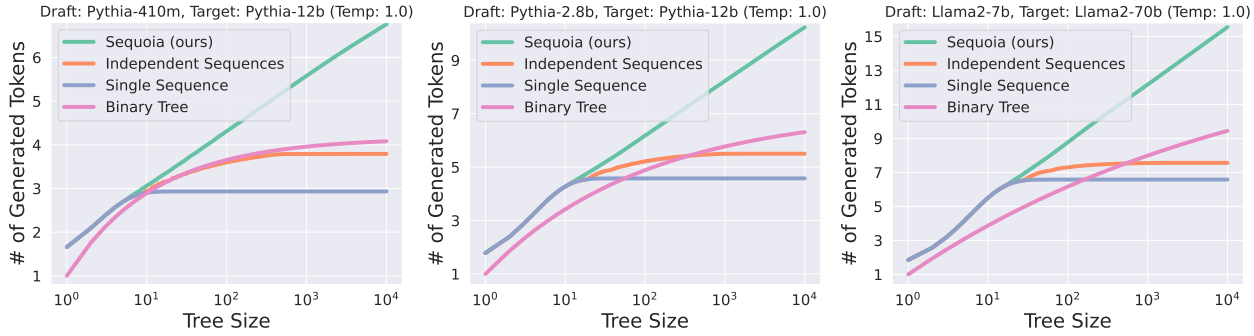


Figure 2: **Number of generated tokens vs. tree size**: We plot the average number of tokens generated for different tree structures per decoding step of the target model, as a function of the tree size, for different draft and target model pairs. The number of generated tokens for SEQUOIA trees continues to grow with the tree size, while other tree structures asymptote.

from the target model in this algorithm (as opposed to from the draft model, like in SpecTr and SpecInfer). This observation motivates our choice—for the top-k sampling method—to populate the tree by taking the top-k children of each node, instead of the naive sampling approach of taking k i.i.d. samples (with replacement). We use the top-k sampling method in our experiments in Section 3.2, to better understand the limits of this verification algorithm.

2.3 Tree optimizer

The original speculative decoding paper [24] proposed picking the number of tokens d to speculate by maximizing the following expression for speedup: $\texttt{Speedup}(n) = \frac{G(n)}{1+c\cdot n}$. Here, $G(n) = \frac{1-\alpha^{n+1}}{1-\alpha}$ is the expected number of generated tokens in one iteration of speculative decoding (if n tokens are speculated), c denotes the time to perform one forward pass of the draft model relative to the target model forward pass, and 1 represents the time to verify the token tree. This approach can be easily extended to the tree-based setting, by replacing $G(n)$ with $G(n,d)$, the expected number of generated tokens for the best tree of size n and depth d. It's important to note that regardless of the value of n, the verification time is treated as a constant $\mathcal{O}(1)$ by prior work (e.g., the computation model in SpecTr [39]).

3 Sequoia

We now present SEQUOIA, a scalable, robust, and hardware-aware speculative decoding algorithm.

- In Section 3.1, we present our scalable tree construction algorithm, which uses dynamic programming to solve for the optimal tree structure. We demonstrate both theoretically and empirically that the number of tokens generated by verifying SEQUOIA trees scales nearly logarithmically in the size of the tree, while existing tree structures asymptote in the number of tokens they can generate.
- In Section 3.2, we present our robust tree verification algorithm, which modifies the SpecInfer algorithm by sampling *without replacement* from the draft model. We show both theoretically and empirically that SEQUOIA is robust, performing well across temperature values, while existing verification methods are not.
- In Section 3.3, we present how we can make SEQUOIA hardware-aware by optimizing the size and depth of the speculated tree based on the hardware being used. We demonstrate theoretically and empirically that choosing these hyperparameters in this manner can yield increases in speed relative to choosing a fixed tree size.

3.1 Tree construction

We now present the SEQUOIA tree construction algorithm, and prove that the expected number of tokens generated by verifying these SEQUOIA trees scales well with the tree size.

3.1.1 Algorithm

To derive the SEQUOIA tree construction algorithm, we first express the tree construction problem as a constrained optimization problem, and then use dynamic programming to solve this problem optimally and

efficiently. In this optimization problem, we aim to maximize the expected number of tokens $F(\mathcal{T})$ generated by verifying a token tree $\mathcal{T}$, under a constraint on the size of $\mathcal{T}$. We begin by presenting a closed form expression for $F(\mathcal{T})$ (Proposition 3.4). We then present our tree construction algorithm, which uses dynamic programming to find the tree of size n which maximizes this expression (for any value of the speculation budget n).

We first present a number of important definitions:

Definition 3.1. Under the *positional acceptance assumption*, the probability of a verification algorithm accepting a token t which is the k^{th} child of an already accepted token depends only on the value of k.

Definition 3.2. The *acceptance vector* is the vector $p=(p_1,p_2,...,p_k,...)$ containing the probabilities p_k that the verification algorithm accepts a token at child position k. Under the positional acceptance assumption, the acceptance dynamics of a verification algorithm can be completely described by the acceptance vector.

Definition 3.3. Given an acceptance vector p and a tree $\mathcal{T}$, we define the *score function* $f(v)$ for a node $v\in\mathcal{T}$ as $f(v)=\prod_{i\in\texttt{Path}(v)} p_i$. where $\texttt{Path}(v)$ is equal to the list of child indices along the path from the root to a node $v\in\mathcal{T}$. For example, if v is the 3^{rd} child of the root's 2^{nd} child, then $\texttt{Path}(v)=[2,3]$. We define $f(root)=1$.

We are now ready to present Proposition 3.4 (proof in Appendix C.2), which shows the closed form solution for the expected number of tokens generated by verifying a token tree $\mathcal{T}$, under the positional acceptance assumption.

Proposition 3.4. *Let $\mathcal{T}$ be a token tree that is verified with the positional acceptance assumption, and let $f(v)$ denote the score function for a node $v\in\mathcal{T}$. Then the the expected number of tokens $F(\mathcal{T})$ generated by verifying $\mathcal{T}$ equals*

$$F(\mathcal{T})=\sum_{v\in\mathcal{T}} f(v)\,.$$

The SEQUOIA tree construction algorithm then simply corresponds to finding the tree $\mathcal{T}$ of size n which maximizes $F(\mathcal{T})$, using dynamic programming. Letting

$$c(n)=\max_{\mathcal{T},|\mathcal{T}|=n} F(\mathcal{T}),$$

we can express $c(n)$ in terms of $c(n')$ values for $n'<n$:

$$c(n)=\max_{a_i,\sum_{i=1}^{n-1}a_i=n-1} \quad 1+\sum_{i=1}^{n-1} p_i\cdot c(a_i)\,.$$

The recursive sub-structure of this optimization problem allows us to solve this problem using dynamic programming. We note that because the time to *speculate* the token tree is proportional to the depth of the tree, it is also important to be able to find the tree $\mathcal{T}$ of size n, and depth at most d, that maximizes $F(\mathcal{T})$. In Appendix C.2, we provide details on the dynamic programming formulations and solutions for both versions of the problem (bounded/unbounded depth).

3.1.2 Theoretical Results

We now prove that the SEQUOIA tree construction algorithm scales well with the size of the speculated tree. In particular, we show that under certain assumptions on the acceptance rates of the verification algorithm, the number of generated tokens is lower-bounded by a function which is (roughly) logarithmic in the size of the tree. This is in contrast to existing tree construction algorithms, which we show (Table 1) are upper bounded in the expected number of tokens they generate.

We first define what it means for a verification algorithm to have a b *power-law acceptance rate*, and then present our theorem on the scalability of SEQUOIA trees, under the assumption that the verification algorithm has a b power-law acceptance rate.

Definition 3.5. We say that a tree verification algorithm has a b *power-law acceptance rate* if the chance r_k of the tree verification algorithm rejecting all k speculated children of a node in a tree is upper bounded by a power-law of k with exponent b—meaning, $r_k\leq 1/k^b\ \forall k\in\mathbb{N}$, for $b>0\in\mathbb{R}$.

The above definition is motivated by our observation (Figure 3) that the SEQUOIA sampling and verification algorithm satisfies power-law acceptance rates in practice. We now state the theorem (proof in Appendix C.4).

Theorem 3.6. *Assume a tree verification algorithm has a b power-law acceptance rate. Then the expected number of tokens $G(n)$ generated by verifying the* SEQUOIA *tree of size n with this algorithm is in $\Omega\big(b\cdot\log(n)/\log(\log(n))\big)$.*

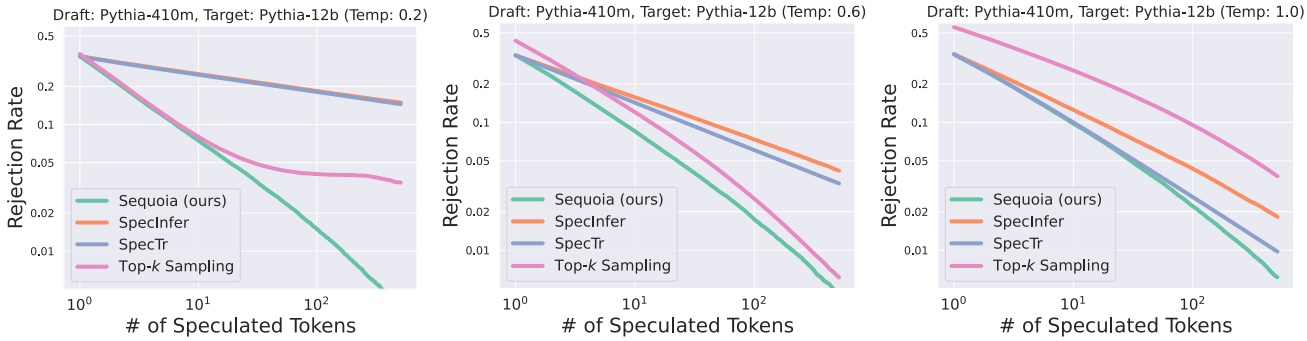


Figure 3: **Rejection rate vs. number speculated tokens**: We plot the average rejection rate ($1 - acceptance\_rate$) for the different verification algorithms, as a function of the number of speculated tokens k. We can see that across temperature settings ($\{0.2,0.6,1.0\}$, left to right), the SEQUOIA verification algorithm attains the lowest rejection rates, and consistently has a *power-law acceptance rate* (Definition 3.5).

Table 1: We present upper bounds on the number of tokens generated in each iteration of a tree-based speculative decoding algorithm, assuming that P_k denotes the probability that if a node has k children, that one of those children will be accepted.

TREE STRUCTURE	UPPER BOUND
SEQUENCE	$\frac{1}{1-P_1}$
k INDEPENDENT SEQUENCES	$1+\frac{P_k}{1-P_1}$
BINARY TREE	$\frac{1}{1-P_2}$
k-ARY TREE	$\frac{1}{1-P_k}$
SEQUOIA (OURS)	∞

3.1.3 Empirical Validation

In Figure 2, we plot the average number of tokens generated by SEQUOIA trees relative to the various tree structures presented in Table 1, as a function of the number of tokens n in the tree, for several draft and target model pairs, using data from WikiText-103. We see that the number of generated tokens for SEQUOIA trees is unbounded—scaling roughly logarithmically with the tree size—whereas the other tree structures asymptote.

3.2 Tree sampling and verification

We present our token tree sampling and verification algorithm, and prove that it is the first such algorithm to satisfy two important robustness properties, while maintaining the output distribution of the target model.

3.2.1 Algorithm

We present the pseudocode for the SEQUOIA Tree sampling and verification algorithm in Algorithm 1. As discussed in Section 2, an important motivation for designing the SEQUOIA verification algorithm was the observation that SpecInfer and SpecTr both perform poorly at low temperatures, due to the fact that they can repeatedly sample (and then reject) a low-quality token that the draft model is confident in. Thus, we wanted to design an algorithm that would never make the same mistake twice—meaning, once a token was rejected, it would never propose that token again. Toward this end, SEQUOIA introduces two changes to the SpecInfer algorithm: First, it performs sampling *without replacement* using the draft model distribution. Second, if all the tokens with non-zero draft model probability have already been sampled and rejected, it uses the uniform distribution over all tokens that have not yet been sampled as the new draft model distribution. These changes significantly improve the robustness of SEQUOIA relative to SpecInfer, while maintaining the guarantee that the output distribution is identical to that of the target model (proof in Appendix C.1).

Algorithm 1 SEQUOIA Sampling and Verification

Input: Prefix $[x_1,x_2,...,x_{n-1}]$, target model probabilities $\mathcal{P}(\cdot\,|\,x_{<n})$, draft model probabilities $\mathcal{Q}(\cdot\,|\,x_{<n})$, and number of branches $k \leq vocab\_size$.
Output: A token x sampled using SEQUOIA.
Initialize residual R with $\mathcal{P}$, draft D with $\mathcal{Q}$, and the set of rejected tokens S with $\varnothing$
for $i = 1 \to k$ **do**
  sample $x_i \sim R$, $r_i \sim \text{Uniform}(0,1)$
  if $r_i < \frac{R[x_i]}{D[x_i]}$ **then** ▷ Accept x_i
    Return x_i
  else ▷ Reject x_i
    $R \leftarrow \text{norm}(\max(R-D,0))$
    $D[x_i] \leftarrow 0$
    $S.\text{add}(x_i)$
    if $\text{sum}(D) = 0$ **then**
      # Let D be uniform over non-rejected set
      $D[t] \leftarrow 0$ if $t \in S$, else 1
    end if
    $D \leftarrow \text{norm}(D)$
  end if
end for
Return $x \sim R$

3.2.2 Theoretical Results

We now prove that the SEQUOIA verification algorithm is robust, in the sense that it satisfies both of the properties below, while existing verification algorithms do not.

- **The *optimal transport* property**: When $k=1$, the acceptance rate is equal to $1 - \frac{\|P-Q\|_1}{2}$.[2]
- **The *cover* property**: If the support of the draft model probability distribution Q is of size k and is a superset of the support of the target model probability distribution P, at most k speculations will be needed to attain an acceptance rate of 1. Furthermore, if k is equal to the vocabulary size, the acceptance rate should always be 1 as well, regardless of the draft model used.

Intuitively, satisfying the *optimal transport* property results in strong performance at high temperatures (because P and Q will approach uniform distributions), while satisfying the *cover* property results in strong performance at low temperatures (assuming the top target model token is in the top-k draft model tokens).

We now present our main robustness result (proof in Appendix C.4):

Theorem 3.7. *The* SEQUOIA *verification algorithm satisfies both the optimal transport and the cover properties, while SpecInfer and SpecTr only satisfy the optimal transport property, and top-k sampling only satisfies the cover property.*

3.2.3 Empirical Validation

In Figure 3, we plot the average rejection rates (equal to 1−acceptance rates) for the different verification algorithms, as a function of the number of speculated child tokens for a fixed token prefix, for various temperatures (0.2, 0.6, 1.0), measured on WikiText-103. We can see that across all temperature settings, the rejection rates for SEQUOIA decay faster than for the other algorithms. In general, we observe that the rejection rates r_k for SEQUOIA follow a power-law, where $r_k \approx 1/k^b$ for some $b > 0$. We can also see that while SpecTr and SpecInfer perform relatively well at high temperatures, they struggle at lower temperatures, and that the opposite is true for the top-k sampling method.

3.3 Hardware-aware Tree Optimizer

We present our proposed method for selecting the SEQUOIA tree size and depth in a hardware-aware manner to optimize the actual speedup attained by SEQUOIA on different hardware. Theoretically, we show that while prior work can get away with optimizing the size of the speculation budget in a hardware-agnostic manner, this is

[2]The SpecTr paper [39] showed that $1 - \frac{\|P-Q\|_1}{2}$ is the acceptance rate attained by the optimal verification algorithm for $k=1$.

Table 2: The components of the speedup equation (Equation 1) for SEQUOIA, as well as for prior work.

	Sequoia	**1 sequence**	k **ind. sequences**
$G(n,d)$	$\Omega\left(\frac{b\log(n)}{\log(\log(n))}\right)$	$\frac{1-P_1^{n+1}}{1-P_1}$	$\leq 1+\frac{1}{1-P_k}$
$t(n)$	$\Omega(n)$	1	1
$d(n)$	d	n	$\lfloor n/k \rfloor$

Table 3: Total speedup (number of tokens generated in parentheses) on different hardware. We compare our hardware-aware tree optimizer with using fixed tree sizes. Our optimizer can yield up to 38% speedups.

Hardware	**Target**	**Draft**	**Hardware-aware** n	**fixed** $n=128$	**fixed** $n=256$
L40	Llama2-13B	JF68M	3.26×(4.40)	3.16×(4.77)	2.51×(5.09)
A100	Vicuna-33B	SL1.3B	2.37×(4.41)	2.09×(4.99)	1.72× (5.30)

not possible in SEQUOIA, due to the fact that for large trees the verification time can no longer be approximated as a constant, independent of the tree size. Empirically, we demonstrate that selecting the tree size and depth in this hardware-aware manner yields up to 40% speedups relative to baselines.

3.3.1 Algorithm

We now show how we can select the tree size and depth optimally, depending on the hardware being used for inference. Letting $G(n,d)$ denote the expected number of tokens generated by verifying the SEQUOIA tree of size n and depth d (computed via dynamic programming), $t(n)$ denote the (hardware-dependent) amount of time it takes the target model to verify n tokens divided by the time to verify 1 token, and c denote the (hardware-dependent) time to draft 1 token divided by the time to verify 1 token, the speedup attained by SEQUOIA can be expressed as:

$$\texttt{Speedup}(n,d) = \frac{G(n,d)}{t(n)+d\cdot c}. \tag{1}$$

We propose optimizing this equation by measuring $t(n)$ and c empirically on the inference hardware being used, and then doing a grid search over possible values of n and d to find the combination that gives the largest speedup. Importantly, for batch size $b>1$, the denominator can be updated to $t(b\cdot n)+d\cdot c$, since the verifier must verify a tree of size n for each element of batch.

3.3.2 Theoretical Results

We now study the properties of the SEQUOIA speedup equation, to better understand the optimal tree size and depth that are selected by this equation, and why it is critical to choose these parameters in a hardware-aware manner.

In Table 2 we break down the speedup equation in terms of its components, for SEQUOIA and prior work. We first observe that for SEQUOIA, because the numerator for the speedup equation grows roughly logarithmically in n, while the denominator grows linearly with n, there must exist an optimal tree size $n<\infty$ that maximizes this speedup expression.

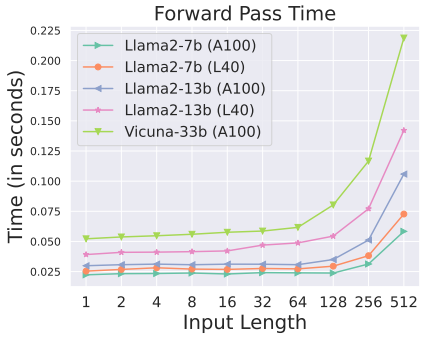


Figure 4: Forward pass times for different model/hardware combinations as a function of the number of tokens n being processed. We use these values to choose the optimal tree.

Additionally, we can observe that prior can get away with setting $t(n)=1$, because the optimal choice of n typically occurs well before $t(n)$ starts growing meaningfully larger than 1. However, this assumption is not always true with SEQUOIA. In SEQUOIA, the fact that we have improved $G(n,d)$ so that it grows (slight sub-)logarithmically in n, necessitates hardware-aware modeling of $t(n)$.

3.3.3 Empirical Validation

In Table 3 we show the speedups attained on several types of hardware when we select the tree size and depth using the correct $t(n)$ function (estimated empirically, see Figure 4) vs. simply choosing a fixed value of n. We can see that using the hardware-aware $t(n)$ function can yield meaningful speedups of up to 40% relative to using fixed values of n across different hardware.

4 Evaluation

In this section, we aim to demonstrate that SEQUOIA can speed up LLM inference by a large margin in wall-clock time. We first present our end-to-end system results showing total speedup, followed by validating our three claims that SEQUOIA is scalable, robust, and hardware-aware.

- In Section 4.1, we demonstrate SEQUOIA's superior end-to-end performance. Specifically, SEQUOIA achieves up-to 4.04× speed-up for Llama2-7B on A100 and 10.33× for Llama2-70B on L40 offloading.
- In Section 4.2.1, we show that the SEQUOIA tree can generate on average 33% more tokens than a tree of 16 independent sequences (tree size 512).
- In Section 4.2.2, we present that SEQUOIA's sampling and verification algorithm is robust to temperature and top-p, consistently outperforming SpecInfer (by up to 1.65×) and top-k sampling (by up to 1.27×).
- In Section 4.2.3, we show that SEQUOIA's hardware-aware tree optimizer can select the best tree size and depth for different hardware settings to maximize speedup.

4.1 End-to-end Results

We now demonstrate that SEQUOIA speeds up LLM decoding in the on-chip setting by up 4.04× on an A100 GPU, and up to 10.33× with offloading on an L40 GPU.

Setup. Our experiments are based on Llama and Vicuna models. For the on-chip setting, we use JackFram/Llama-68m (JF68m) [27] and princeton-nlp/Sheared-Llama-1.3B (SL1.3B) [45] as the draft models, and Llama2-7B [42], Llama2-13B, and Vicuna-33B [5] as the target models. For the offloading setting, we use Llama2-7B as the draft model and Llama2-70B as the target model. We evaluate our results on C4(en) [34] validation dataset, OpenWebText [12] and CNN DailyMail [35]. We sample 200 examples for each experiment. The prompt length and generation length are both set to 128 tokens. We evaluate SEQUOIA on different hardware including on-chip experiments on L40 and A100 GPUs, as well as offloading experiments on an L40 GPU. We also compare SEQUOIA with SpecInfer [27] with 5×8 trees (5 independent sequences of length 8, the tree structure used in [27] for batch size 1) and 8×8 trees for the on-chip setting, and 16×48 trees for the offloading setting.

Implementation Details. We implement the draft and target models using Transformers [44] for the on-chip setting, and using Accelerate [15] (**cpu offload** API) for the offloading setting. Because we determine the optimal tree structure in advance, we are able to use PyTorch CUDA graphs [30, 31] to reduce the overhead of kernel launching during speculative decoding. To accelerate sampling without replacement—which is not efficient in PyTorch 2.1 [31]—we use the exponential-sort algorithm [43], combined with PyTorch CUDA graphs [30, 31].

Main Results. We evaluate SEQUOIA using different temperatures, draft and target model pairs, and hardware configurations. Results are shown in Table 4 (A100 on-chip) and Table 5 (L40 offloading). We observe that SEQUOIA consistently speeds up LLM decoding in a wide range of settings. SEQUOIA reaches up to 4.04× speedup for the on-chip setting, and up to 10.33× speedup for the offloading setting, as a result of the huge gap between computation capacity and memory bandwidth. We present additional on-chip results (L40 GPU) in Appendix D.

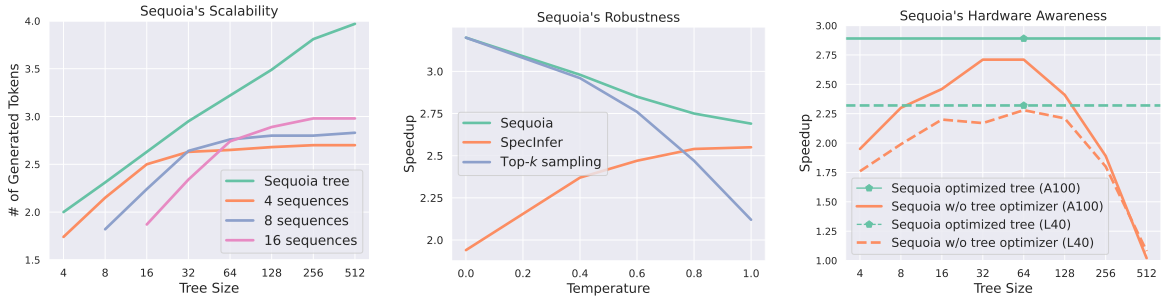
Table 4: **On-chip results (A100)**: The optimal tree configuration and speedup for different pairs of draft and target models, and different temperatures, for SEQUOIA vs. SpecInfer. We specify the average number of generated tokens per decoding step in parentheses, next to the speedup factor. SEQUOIA attains up to 4.04× speedup on an A100.

Target LLM	Draft Model	T	Dataset	Tree Config. (size, depth)	Speedup	SpecInfer 5×8	SpecInfer 8×8
Llama2-7B	JF68M	0	C4	(128,10)	**4.04 ×(5.04)**	3.66×(3.98)	3.62×(4.04)
Llama2-7B	JF68M	0.6	C4	(128,7)	**3.15×(3.88)**	2.47×(2.96)	2.43×(3.03)
Llama2-7B	JF68M	0	OpenWebText	(128,7)	**3.29×(3.87)**	2.99×(3.17)	2.99×(3.29)
Llama2-7B	JF68M	0.6	OpenWebText	(128,6)	**2.74×(3.33)**	2.03×(2.50)	2.09×(2.59)
Llama2-7B	JF68M	0	CNN Daily	(128,7)	**3.24×(3.83)**	2.80×(3.03)	2.85×(3.15)
Llama2-7B	JF68M	0.6	CNN Daily	(128,6)	**2.83×(3.45)**	2.03×(2.54)	2.07×(2.56)
Llama2-13B	JF68M	0	C4	(64,9)	**3.84×(4.36)**	3.40×(3.76)	3.15×(3.85)
Llama2-13B	JF68M	0.6	C4	(64,7)	**3.21×(3.58)**	2.47×(2.83)	2.40×(2.98)
Llama2-13B	JF68M	0	OpenWebText	(64,7)	**3.08×(3.39)**	2.71×(2.99)	2.54×(3.07)
Llama2-13B	JF68M	0.6	OpenWebText	(64,6)	**2.77×(3.02)**	2.15×(2.49)	1.99×(2.49)
Llama2-13B	JF68M	0	CNN Daily	(64,7)	**3.20×(3.51)**	2.74×(3.00)	2.54×(3.10)
Llama2-13B	JF68M	0.6	CNN Daily	(64,6)	**2.89×(3.15)**	2.19×(2.49)	2.03×(2.54)
Llama2-13B	JF160M	0	C4	(64,7)	**3.12×(4.73)**	2.73×(4.47)	2.63×(4.53)
Llama2-13B	JF160M	0.6	C4	(64,6)	**2.83×(4.06)**	2.08×(3.44)	2.05×(3.56)
Llama2-13B	JF160M	0	OpenWebText	(64,6)	**2.65×(3.79)**	2.16×(3.52)	2.10×(3.59)
Llama2-13B	JF160M	0.6	OpenWebText	(64,5)	**2.51×(3.40)**	1.76×(2.90)	1.73×(2.99)
Llama2-13B	JF160M	0	CNN Daily	(64,6)	**2.80×(4.00)**	2.20×(3.57)	2.14×(3.67)
Llama2-13B	JF160M	0.6	CNN Daily	(64,5)	**2.60×(3.51)**	1.87×(3.06)	1.81×(3.14)
Vicuna-33B	SL1.3B	0	C4	(64,6)	**2.37×(4.44)**	1.94×(4.09)	1.82×(4.20)
Vicuna-33B	SL1.3B	0.6	C4	(64,6)	**2.20×(4.19)**	1.67×(3.60)	1.62×(3.79)
Vicuna-33B	SL1.3B	0	OpenWebText	(64,5)	**2.20×(3.88)**	1.72×(3.64)	1.63×(3.75)
Vicuna-33B	SL1.3B	0.6	OpenWebText	(64,5)	**2.12×(3.80)**	1.56×(3.34)	1.44×(3.36)
Vicuna-33B	SL1.3B	0	CNN Daily	(64,5)	**2.15×(3.81)**	1.66×(3.49)	1.56×(3.58)
Vicuna-33B	SL1.3B	0.6	CNN Daily	(64,5)	**2.10×(3.75)**	1.51×(3.23)	1.43×(3.33)

Table 5: **Offloading results (L40)**: The optimal tree configuration and speedup for different pairs of draft and target models, and different temperatures, for SEQUOIA vs. SpecInfer. We specify the average number of generated tokens per decoding step in parentheses, next to the speedup factor. SEQUOIA attains up to 10.33× speedup in the offloading setting on an L40.

Target LLM	Draft Model	T	Dataset	Tree Config. (size, depth)	Speedup	SpecInfer 16×48
Llama2-70B	Llama2-7B	0	C4	(768,25)	**9.83 ×(11.19)**	7.28×(8.35)
Llama2-70B	Llama2-7B	0.6	C4	(768,26)	**10.33×(11.79)**	7.32×(8.70)
Llama2-70B	Llama2-7B	0	OpenWebText	(768,19)	**9.15 ×(9.77)**	5.51×(6.00)
Llama2-70B	Llama2-7B	0.6	OpenWebText	(768,21)	**7.85×(8.95)**	5.78×(6.50)
Llama2-70B	Llama2-7B	0	CNN Daily	(768,19)	**8.48 ×(9.09)**	5.98×(6.44)
Llama2-70B	Llama2-7B	0.6	CNN Daily	(768,20)	**8.57×(9.12)**	4.31×(5.29)

Analysis. We made several interesting observations on the interplay between SEQUOIA tree construction, sampling and verification, and hardware-aware optimizer. (1) SEQUOIA selects much larger trees in the offloading setting (768 tokens) than in the on-chip setting (64 to 128 tokens). (2) In general, the average number of generated tokens is close to the wall-clock time speedup (especially when JF68M is used as the draft) as a result of the hardware-aware tree optimizer. (3) The optimal trees found by SEQUOIA for slightly different configurations—e.g., different temperatures and model pairs—can be very different from one another. (4) SEQUOIA chooses deeper trees at low temperature than high temperature, due to the acceptance rates being higher for low temperature. (5) The optimal tree size for larger models such as 33B or 13B is generally smaller



(a) (b) (c)

Figure 5: **Left**: We compare the number of tokens generated on average by SEQUOIA trees vs. k independent sequences, where we use SEQUOIA sampling and verification for both tree structures. The gap between SEQUOIA trees and the baselines demonstrates the improved scalability of the SEQUOIA tree construction algorithm. **Middle**: We compare the speedups attained by the SEQUOIA sampling and verification algorithm relative to SpecInfer and top-k sampling, across various temperatures, holding the tree structure fixed. We can see SEQUOIA is robust to the choice of temperature, performing well across the different settings. **Right**: We compare the wall-clock time speedup of SEQUOIA trees of various sizes (orange lines)—chosen to maximize the # generated tokens—with the speedup of the trees selected by the hardware-aware tree optimizer (horizontal green lines)—chosen to maximize speedup—on A100 and L40 GPUs. The optimizer can select the optimal tree size and depth for each type of hardware; by limiting the depth of the tree it can make speculation faster and thus attain larger speedups than the trees with unconstrained depth (orange lines).

than for smaller models because the verification time increases faster with the number of candidate tokens, as illustrated in Figure 4.

4.2 Ablations

We present our ablation experiments validating the scalability of the SEQUOIA tree construction algorithm (Section 4.2.1), the robustness of SEQUOIA tree sampling and verification algorithm (Section 4.2.2), and the hardware-awareness of the SEQUOIA tree optimizer (Section 4.2.3). For each of these experiments, we only vary one element at a time (e.g., the tree structure for Section 4.2.1) to study the gains attained by each component of SEQUOIA.

4.2.1 Tree Construction

In this section, we evaluate the scalability of the SEQUOIA tree construction method in terms of the average number of generated tokens at different budget sizes, relative to baselines. In Figure 5a we compare the SEQUOIA tree to k independent sequences, where we use SEQUOIA's sampling and verification algorithm for both tree structures. The SEQUOIA tree is able to generate up to 33% more tokens per decoding step, demonstrating the effectiveness of SEQUOIA's tree construction algorithm. For these experiments, we use JackFram/Llama-68m as the draft model, Llama2-13B as the target model, 0.6 as the temperature, and CNN Daily Mail as the dataset.

4.2.2 Tree Sampling and Verification

Here, we compare the SEQUOIA sampling and verification algorithm to SpecInfer and top-k sampling across different inference hyperparameters (temperature and top-p [19]), holding the tree structure fixed. In Figure 5b we present the results for these methods as we vary the temperature (for top-p=1), while in Table 6 we present results varying the top-p parameter (for temp=1). We can see that SEQUOIA achieves the largest speedups across all temperature and top-p settings, attaining up to 1.65× and 1.27× speedup relative to SpecInfer and top-k sampling, respectively. In addition, we can observe that the SEQUOIA sampling and verification algorithm performs well across different temperature and top-p settings, whereas top-k sampling and SpecInfer excel in different regimes[3]. For these experiments, we use JackFram/Llama-68m as the draft model, Llama2-7B as the

[3]Top-p is not efficiently supported in current implementation, which influences the total speedup.

Table 6: We compare the robustness of the SEQUOIA sampling and verification algorithm to the top-p hyperparameter, relative to SpecInfer and top-k sampling. We present total speedups on an A100 GPU for the different methods (number of generated tokens in parentheses). We hold the tree structure fixed across methods, use JF68M as the draft model, and Llama2-7B as the target model.

Top-p	**Sequoia (Ours)**	**SpecInfer**	**top-k sampling**
0.8	2.54×(3.18)	2.35×(2.93)	2.43×(2.90)
0.9	2.61×(3.27)	2.42×(3.01)	2.27×(2.71)
1.0	2.69×(3.26)	2.55×(3.10)	2.12×(2.44)

target model, 0.6 as the temperature, CNN Daily Mail as the dataset, and the corresponding SEQUOIA tree from Table 4 as the tree structure.

4.2.3 Hardware-aware Tree Optimizer

In this section, we demonstrate the effectiveness of the SEQUOIA hardware-aware tree optimizer. In Figure 5c, we compare the speedups attained by the SEQUOIA trees of various sizes from Figure 5a to the trees selected by the hardware-aware tree-optimizer. Because the tree optimizer is able to limit the tree depth to make speculation faster, it is able to attain larger end-to-end speedups than any of the SEQUOIA trees from Figure 5a, whose structures were chosen to maximize the expected number of generated tokens (not the speedup). The optimizer is also able to automatically find the tree size that produces the largest overall speedup.

5 Conclusion

We presented SEQUOIA, a scalable, robust, and hardware-aware speculative decoding method. By improving the topology of the token tree, the sampling algorithms, and the choice of tree size, SEQUOIA is able to speed up autoregressive LLM inference up to 4.04× on GPU and 10.33× with offloading. In addition to providing real speedups, we believe SEQUOIA also provides insight into both the large potential and fundamental limits of speculative decoding systems. We hope that this understanding inspires future work in this area, or even informs the design of custom chips for LLM inference.

References

[1] Joshua Ainslie, Tao Lei, Michiel de Jong, Santiago Ontañón, Siddhartha Brahma, Yury Zemlyanskiy, David Uthus, Mandy Guo, James Lee-Thorp, Yi Tay, et al. Colt5: Faster long-range transformers with conditional computation. *arXiv preprint arXiv:2303.09752*, 2023.

[2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[3] Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads, 2024.

[4] Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *CoRR*, abs/2302.01318, 2023. doi: 10.48550/ARXIV.2302.01318. URL `https://doi.org/10.48550/arXiv.2302.01318`.

[5] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023. URL `https://lmsys.org/blog/2023-03-30-vicuna/`.

[6] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. PaLM: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[7] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *CoRR*, abs/2307.08691, 2023. doi: 10.48550/ARXIV.2307.08691. URL https://doi.org/10.48550/arXiv.2307.08691.

[8] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

[9] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm.int8(): 8-bit matrix multiplication for transformers at scale. *CoRR*, abs/2208.07339, 2022. doi: 10.48550/ARXIV.2208.07339. URL https://doi.org/10.48550/arXiv.2208.07339.

[10] Elias Frantar and Dan Alistarh. Massive language models can be accurately pruned in one-shot. *arXiv preprint arXiv:2301.00774*, 2023.

[11] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: accurate post-training quantization for generative pre-trained transformers. *CoRR*, abs/2210.17323, 2022. doi: 10.48550/ARXIV.2210.17323. URL https://doi.org/10.48550/arXiv.2210.17323.

[12] Aaron Gokaslan and Vanya Cohen. Openwebtext corpus, 2019.

[13] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *CoRR*, abs/2312.00752, 2023. doi: 10.48550/ARXIV.2312.00752. URL https://doi.org/10.48550/arXiv.2312.00752.

[14] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=uYLFoz1vlAC.

[15] Sylvain Gugger, Lysandre Debut, Thomas Wolf, Philipp Schmid, Zachary Mueller, Sourab Mangrulkar, Marc Sun, and Benjamin Bossan. Accelerate: Training and inference at scale made simple, efficient and adaptable. https://github.com/huggingface/accelerate, 2022.

[16] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[17] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015.

[18] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *J. Mach. Learn. Res.*, 22(241):1–124, 2021.

[19] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2019.

[20] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.

[21] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 5156–5165. PMLR, 2020. URL http://proceedings.mlr.press/v119/katharopoulos20a.html.

[22] Sehoon Kim, Karttikeya Mangalam, Jitendra Malik, Michael W. Mahoney, Amir Gholami, and Kurt Keutzer. Big little transformer decoder. *CoRR*, abs/2302.07863, 2023. doi: 10.48550/ARXIV.2302.07863. URL https://doi.org/10.48550/arXiv.2302.07863.

[23] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In Jason Flinn, Margo I. Seltzer, Peter Druschel, Antoine Kaufmann, and Jonathan Mace, editors, *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, pages 611–626. ACM, 2023. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

[24] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR, 2023.

[25] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han. AWQ: activation-aware weight quantization for LLM compression and acceleration. *CoRR*, abs/2306.00978, 2023. doi: 10.48550/ARXIV.2306.00978. URL https://doi.org/10.48550/arXiv.2306.00978.

[26] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

[27] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Rae Ying Yee Wong, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. Specinfer: Accelerating generative llm serving with speculative inference and token tree verification. *arXiv preprint arXiv:2305.09781*, 2023.

[28] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

[29] Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling. Data-free quantization through weight equalization and bias correction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1325–1334, 2019.

[30] NVIDIA, Péter Vingelmann, and Frank H.P. Fitzek. Cuda, release: 10.2.89, 2020. URL https://developer.nvidia.com/cuda-toolkit.

[31] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[32] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. *ArXiv*, abs/2211.05102, 2022. URL https://api.semanticscholar.org/CorpusID:253420623.

[33] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. *arXiv preprint arXiv:2211.05102*, 2022.

[34] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv e-prints*, 2019.

[35] Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1099. URL https://www.aclweb.org/anthology/P17-1099.

[36] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. Flexgen: High-throughput generative inference of large language models with a single GPU. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 31094–31116. PMLR, 2023. URL `https://proceedings.mlr.press/v202/sheng23a.html`.

[37] Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. *Advances in Neural Information Processing Systems*, 31, 2018.

[38] Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023.

[39] Ziteng Sun, Ananda Theertha Suresh, Jae Hun Ro, Ahmad Beirami, Himanshu Jain, and Felix Yu. Spectr: Fast speculative decoding via optimal transport. *arXiv preprint arXiv:2310.15141*, 2023.

[40] Raphael Tang, Yao Lu, Linqing Liu, Lili Mou, Olga Vechtomova, and Jimmy Lin. Distilling task-specific knowledge from bert into simple neural networks. *arXiv preprint arXiv:1903.12136*, 2019.

[41] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, pages 10347–10357. PMLR, 2021.

[42] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.

[43] Tim Vieira. Gumbel-max trick and weighted reservoir sampling, 2014.

[44] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

[45] Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. Sheared llama: Accelerating language model pre-training via structured pruning. *arXiv preprint arXiv:2310.06694*, 2023.

[46] Daliang Xu, Wangsong Yin, Xin Jin, Ying Zhang, Shiyun Wei, Mengwei Xu, and Xuanzhe Liu. Llmcad: Fast and scalable on-device large language model inference. *arXiv preprint arXiv:2309.04255*, 2023.

[47] Lu Yin, You Wu, Zhenyu Zhang, Cheng-Yu Hsieh, Yaqing Wang, Yiling Jia, Mykola Pechenizkiy, Yi Liang, Zhangyang Wang, and Shiwei Liu. Outlier weighed layerwise sparsity (owl): A missing secret sauce for pruning llms to high sparsity. *arXiv preprint arXiv:2310.05175*, 2023.

[48] Gyeong-In Yu and Joo Seong Jeong. Orca: A distributed serving system for transformer-based generative models. In *USENIX Symposium on Operating Systems Design and Implementation*, 2022. URL `https://api.semanticscholar.org/CorpusID:251734964`.

[49] Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft & verify: Lossless large language model acceleration via self-speculative decoding. *CoRR*, abs/2309.08168, 2023. doi: 10.48550/ARXIV.2309.08168. URL `https://doi.org/10.48550/arXiv.2309.08168`.

[50] Ritchie Zhao, Yuwei Hu, Jordan Dotzel, Chris De Sa, and Zhiru Zhang. Improving neural network quantization without retraining using outlier channel splitting. In *International conference on machine learning*, pages 7543–7552. PMLR, 2019.

[51] Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. Distillspec: Improving speculative decoding via knowledge distillation. *arXiv preprint arXiv:2310.08461*, 2023.

A Related Work

This work introduces a new algorithm in the family of speculative decoding methods that aims to maintain the exact output distribution of the target model by improving the structure and sampling/verification algorithm for the speculated token tree. There exist many other directions within this line of work—for example, methods which introduce leniency into the speculative decoding algorithm to attain increased speed at the cost of accuracy [22, 37], methods that reuse layers or representations from the target model as the draft model [3, 49], etc. Alternatively, the draft model can be distilled to better approximate the target model; DistillSpec [17, 40, 41, 51] improves that process by using model-generated data and adjusting the objective depending on the task and the decoding strategy. Finally, LLMCad [46] proposes an advanced algorithm for token tree generation and verification in the context of on-device LLM inference.

In addition to speculative decoding, there exist many other methods aimed at improving the speed of LLM inference. For example, model quantization is another very promising way of dealing with the I/O bottleneck during inference, by reducing the number of bits per parameter. However, unlike speculative decoding, these methods generally deteriorate the quality of the model to some degree, depending on the amount of quantization [9, 11, 16, 20, 25, 29, 50] or sparsity [18, 26, 28].

Meanwhile, various works [1, 10, 38, 47] have studied ways to improve LLM serving throughput. Pope et al. [32] investigated the batching effect in scaling up LLM. Orca [48] proposed a distributed LLM serving system that uses a finegrained scheduling policy to improve GPU utilization under various request lengths. vLLM [23] used page tables to manage GPU memory to increase memory utilization, which significantly boosts inference throughput. FlexGen [36] proposed an offloading mechanism to support larger batches to achieve high throughput.

FlashAttention [7, 8] is another algorithm that aims to improve the speed of LLMs (at both training and inference time) by considering the I/O cost of different operations.

Another promising approaching to speeding up inference is to change the fundamental building blocks of the model. Recently, numerous sub-quadratic architectures—including SSMs [13, 14] and linear attention models [21]—have been proposed. These models are particularly beneficial for long inputs.

B Background

B.1 Sequence-based speculative decoding

The original speculative decoding method [4, 24] proposes using a small "draft model" to speculate γ tokens into the future, and then using the "target model" to in parallel process these tokens and decide which of the tokens to "accept", in such a way that the output distribution of the target model is unchanged. This algorithm is presented in Algorithm 2.

Leviathan et al. [24] analyze the performance of this algorithm, presenting equations for the expected number of accepted tokens from one run of the algorithm, and the expected wall-clock speed up from using speculative decoding (relative to standard autoregressive inference with the target model). In this analysis, they introduce the acceptance rate $\alpha \in [0,1]$, corresponding to the probability that a token x_i is accepted by Algorithm 2, under the simplifying assumption that the acceptance decisions are i.i.d.[4] Under this assumption, they show that the expected number of generated tokens in each run of Algorithm 2 is $\frac{1-\alpha^{\gamma+1}}{1-\alpha}$. Additionally, letting c denote the ratio between the time to run the draft model and the time to run the target model, they show that the expected wall-clock speed-up from using this algorithm is $\frac{1-\alpha^{\gamma+1}}{(1-\alpha)(\gamma c+1)}$.

B.2 SpecInfer

Here we present the pseudocode for the SpecInfer algorithm in Algorithm 3.

[4] One can think of α as the average acceptance rate over many runs of this algorithm on a representative dataset.

Algorithm 2 Sequence-based Speculative Decoding

Input: Prefix $[x_1,x_2,...,x_{n-1}]$, Target model M_p, draft model M_q, and number of tokens γ to speculate.
Output: A sequence of tokens generated using speculative decoding.
for $i=n\rightarrow n+\gamma$ - 1 **do** ▷ Sample sequence of γ tokens from draft model
  $q_i(x)\leftarrow M_q([x_1,...,x_{i-1}])$
  $x_i\sim q_i(x)$
end for
for $i=n\rightarrow n+\gamma$ **do** ▷ For loop below can be run in parallel with a single forward pass of M_p
  $p_i(x)\leftarrow M_q([x_1,...,x_{i-1}])$
end for
$s\leftarrow n-1$ ▷ Choose how many tokens n to accept
for $i=n\rightarrow n+\gamma$ - 1 **do**
  $r_i\sim$ Uniform(0,1)
  if $r_i<\frac{p_i(x_i)}{q_i(x_i)}$ **then**
    $s\leftarrow s+1$
  else
    break
  end if
end for
$p'(x)\leftarrow p_{s+1}(x)$
if $t<n+\gamma-1$ **then**
  $p'(x)\leftarrow$ norm(max$(0,p_{s+1}(x)-q_{s+1}(x))$)
end if
$t\sim p'(x)$ ▷ Sample a final token from $p'(x)$
Return $x_1,...,x_s,t$

Algorithm 3 SpecInfer Sampling and Verification

Input: Prefix $[x_1,x_2,...,x_{n-1}]$, target model probabilities $\mathcal{P}(\cdot\,|\,x_{<n})$, draft model probabilities $\mathcal{Q}(\cdot\,|\,x_{<n})$, and number of branches k.
Output: A token x sampled using SpecInfer.
Initialize residual R with $\mathcal{P}$ and draft D with $\mathcal{Q}$
for $i=1\rightarrow k$ **do**
  sample $x_i\sim D$, $r_i\sim$ Uniform(0,1)
  if $r_i<\frac{R[x_i]}{D[x_i]}$ **then** ▷ Accept x_i
    Return x_i
  else ▷ Reject x_i
    $R\leftarrow$ norm(max$(R-D,0)$)
  end if
end for
sample $x\sim R$
Return x

C Theoretical results

C.1 Correctness of Sequoia verification algorithm

We prove now that the SEQUOIA verification algorithm maintains the output distribution of the target model. We assume we have a target model t, and a list of draft models $(d_1,...d_n,d_{n+1},...)$, where d_i in this case depends on the previously rejected samples $x_1,...,x_{i-1}$, and where $d_i(u)$ and $t(u)$ denote the probabilities of sampling token $u\in V$ from d_i or t respectively (where V is the token vocabulary). We let t_i denote the residual at iteration i of SEQUOIA loop, (after $i-1$ nodes have been rejected (so $t_1=t$, as can be seen in Algorithm 1)

We will prove by induction on the number of proposed tokens n that the SEQUOIA verification algorithm is correct.

Base case ($n=0$): SEQUOIA is trivially correct, as it will simply sample from the residual t_1, which is equal to t.

Recursive case: We assume SEQUOIA is correct for $n-1$ proposed samples and prove it is correct for n proposed samples.

We first show that at stage i in the speculative decoding algorithm, the chance of SEQUOIA choosing to reject the proposed sample is equal to $\sum_x \max\Big(0, t_i(x) - d_i(x)\Big)$:

Lemma C.1. *P(No token accepted at iteration i)* $= \sum_x \max\Big(0, t_i(x) - d_i(x)\Big)$.

Proof.

$$\begin{aligned}
P(\text{No token accepted at iteration } i) &= \sum_x P(\text{sample } x) \cdot P(\text{reject } x \mid x \text{ is sampled}) \\
&= \sum_x d_i(x) \cdot \left(1 - \min\Big(\frac{t_i(x)}{d_i(x)}, 1\Big)\right) \\
&= \sum_x d_i(x) - \sum_x \min\Big(t_i(x), d_i(x)\Big) \\
&= \sum_x t_i(x) - \sum_x \min\Big(t_i(x), d_i(x)\Big) \\
&= \sum_x t_i(x) + \max\Big(-t_i(x), -d_i(x)\Big) \\
&= \sum_x t_i(x) - t_i(x) + \max\Big(0, t_i(x) - d_i(x)\Big) \\
&= \sum_x \max\Big(0, t_i(x) - d_i(x)\Big)
\end{aligned}$$

□

We are now ready to prove the recursive case of the SEQUOIA algorithm. By the inductive hypothesis, we know that for all $u \in V$,

$$t(u) = P(u \text{ accepted in first } n-1 \text{ iterations}) + P(\text{No token accepted in first } n-1 \text{ iterations}) \cdot t_n(u)$$

What this means is that in the case where we run SEQUOIA for $n-1$ iterations (and if no token is accepted we sample from the residual t_n), this is equivalent to sampling from the target distribution t directly. We would like to show that this output distribution is equivalent to the one we would get if we run SEQUOIA for n iterations (and if no token is accepted we sample from the residual t_{n+1}). The output distribution of this scenario can be written as follows:

$$\begin{aligned}
&P(u \text{ accepted in first } n-1 \text{ iterations}) + P(\text{No token accepted in first } n-1 \text{ iterations}) \cdot \\
&\Big(d_n(u) \cdot P(u \text{ accepted at iteration } n) + P(\text{No token accepted in iteration } n) \cdot t_{n+1}(u) \Big)
\end{aligned}$$

Thus, all we must show is that

$$t_n(u) = d_n(u) \cdot P(u \text{ accepted at iteration } n) + P(\text{No token accepted in iteration } n) \cdot t_{n+1}(u)$$

We now show this desired result. We will use Lemma C.1, and the fact that by definition of Algorithm B.2, we know that $t_{n+1}(u) = \frac{\max\Big(0, t_n(u) - d_n(u)\Big)}{\sum_x \max\Big(0, t_n(x) - d_n(x)\Big)}$.

$$
\begin{aligned}
&d_n(u)\cdot P(u \text{ accepted at iteration } n) + P(\text{No token accepted in iteration } n)\cdot t_{n+1}(u) \\
&= d_n(u)\cdot \min\left(1, \frac{t_n(u)}{d_n(u)}\right) + \left(\sum_x \max\Big(0, t_n(x) - d_n(x)\Big)\right) t_{n+1}(u) \\
&= \min\Big(d_n(u), t_n(u)\Big) + \left(\sum_x \max\Big(0, t_n(x) - d_n(x)\Big)\right) \cdot \left(\frac{\max\big(0, t_n(u) - d_n(u)\big)}{\sum_x \max\big(0, t_n(x) - d_n(x)\big)}\right) \\
&= \min\Big(d_n(u), t_n(u)\Big) + \max\Big(0, t_n(u) - d_n(u)\Big) \\
&= t_n(u)
\end{aligned}
$$

To see that this last equality holds, we consider two cases:

1. Case 1 $\Big(t_n(u) \geq d_n(u)\Big)$: $\min(d_n(u), t_n(u)) + \max(0, t_n(u) - d_n(u)) = d_n(u) + t_n(u) - d_n(u) = t_n(u)$.
2. Case 1 $\Big(t_n(u) < d_n(u)\Big)$: $\min(d_n(u), t_n(u)) + \max(0, t_n(u) - d_n(u)) = t_n(u) + 0 = t_n(u)$.

This completes the proof.

C.2 Sequoia tree construction details

We now prove Proposition 3.4 by deriving the closed-form expression for $F(\mathcal{T})$ (the expected number of tokens generated by verifying tree $\mathcal{T}$), and show how to use dynamic programming to find the optimal tree $\mathcal{T}$ under a tree budget size.

Proposition C.2. *Let $\mathcal{T}$ be a token tree that is verified with the positional acceptance assumption, and let $f(v)$ denote the score function for a node $v \in \mathcal{T}$. Then the the expected number of tokens $F(\mathcal{T})$ generated by verifying $\mathcal{T}$ is equal to*

$$
F(\mathcal{T}) = \sum_{v \in \mathcal{T}} f(v)\,.
$$

Proof. Let $D(\mathcal{T})$ denote the expected number of tokens generated by verifying tree $\mathcal{T}$. We would like to prove that $D(\mathcal{T}) = F(\mathcal{T})\ \forall \mathcal{T}$. We will prove this by induction on the size of $\mathcal{T}$.

Base case ($n = 1$): A tree of size 1 is composed solely of the root node. By definition of the score function $f(v)$ (Definition 3.3), we know that $f(v) = 1$ for the root node, so $F(\mathcal{T}) = 1$. $D(\mathcal{T}) = 1$ also, because verifying a tree composed of a root node with no children will simply sample from the target model, and generate 1 token.

Inductive step ($n > 1$): For $|\mathcal{T}| = n > 1$, let v be a leaf of $\mathcal{T}$ at child index i_v of depth d with parent v_p and sibling $\mathcal{S}_v$ (set of sibling indices). We can then consider the tree $\mathcal{T}' = \mathcal{T} - \{v\}$. Based on the inductive assumption, we know that $g(\mathcal{T}') = D(\mathcal{T}')$. Using this assumption, we can express $D(\mathcal{T})$ in terms of $D(\mathcal{T}')$:

$$
\begin{aligned}
D(\mathcal{T}) &= D(\mathcal{T}') - (d-1)\cdot f(v_p)\cdot\left(1 - \sum_{i \in \mathcal{S}_v} p_i\right) + (d-1)\cdot f(v_p)\cdot\left(1 - \sum_{i \in \mathcal{S}_v \cup \{i_v\}} p_i\right) + d\cdot f(v) \\
&= D(\mathcal{T}') - (d-1) f(v_p) p_{i_v} + d\cdot f(v) \\
&= \sum_{v' \in \mathcal{T}'} f(v') - (d-1) f(v) + d\cdot f(v) \\
&= F(\mathcal{T}') + f(v) \\
&= F(\mathcal{T})
\end{aligned}
$$

Note that we use the inductive hypothesis, along with the fact the $f(v_p)\cdot p_{i_v} = f(v)$ (by definition of $f(v)$). $\square$

C.2.1 Dynamic Programming (Unbounded Tree Depth)

We will now describe our dynamic programming solution to find the tree $\mathcal{T}$ of size n with the highest value of $F(\mathcal{T})$. Recall that we defined

$$c(n) = \max_{\mathcal{T}, |\mathcal{T}|=n} F(\mathcal{T}),$$

and that we showed that we can express $c(n)$ in terms of $c(n')$ values for $n' < n$:

$$c(n) = \max_{a_i, \sum_{i=1}^{n-1} a_i = n-1} 1 + \sum_{i=1}^{n-1} p_i c(a_i).$$

Using this recursive sub-structure, we show that $c(n)$ can be calculated by dynamic programming. To make problem simple, we suppose that the number of candidates (i.e. the successor set of any vertices) has an upper bound k. Then consider the branch in the first layer, we can get

$$c(n+1) = \max 1 + p_1 c(a_1) + p_2 c(a_2) + ... + p_k c(a_k) \quad s.t. \quad \sum_{i=1}^{k} a_i = n, \mathbf{I}(a_i > 0) \geq \mathbf{I}(a_{i+1} > 0) \quad i \in [k-1]$$

We assume that $p_1 \geq p_2 \geq ... \geq p_k$ (this is reasonable due to the mechanism of SEQUOIA and SpecInfer). Then we can cancel the last constraints, and the function we are optimizing becomes:

$$c(n+1) = \max 1 + p_1 c(a_1) + p_2 c(a_2) + ... + p_k c(a_k) \quad s.t. \quad \sum_{i=1}^{k} a_i = n$$

Based on the assumption that $p_1 \geq p_2 \geq ... \geq p_k$, we can reach that in optimal solution $a_1 \geq a_2 ... \geq a_k$.

To simplify the optimization objective, we define a new variable for $n \geq L \geq 1$

$$c_L(n+1) = \max 1 + p_1 c(a_1) + p_2 c(a_2) + ... + p_k c(a_k) \quad s.t.$$

$$\sum_{i=1}^{k} a_i = n, a_1, a_2, ..., a_L > 0, a_{L+1} = a_{L+2} = ... = a_k = 0$$

and $c_0(1) = 1$. Then apparently,

$$c(n) = \max_{L \in [n-1]} c_L(n)$$

Then we consider the transfer function,

$$\begin{aligned} c_{L+1}(n) &= \max 1 + p_1 c(a_1) + p_2 c(a_2) + ... + p_k c(a_k) \\ &= \max 1 + p_1 c(a_1) + p_2 c(a_2) + ... + p_{L+1} c(a_{L+1}) \\ &= \max_{x \in \{L+1, L+2, ..., n-1\}} c_L(x) + p_{L+1} c(n-x) \end{aligned}$$

$$c_1(n) = 1 + p_1 c(n-1)$$

The computing order is visualized in Figure 6.

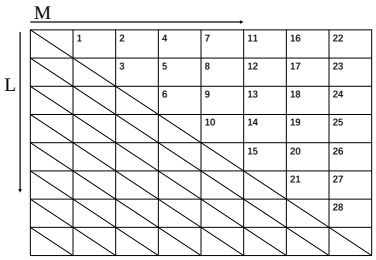


Figure 6: Computing Order

The complexity can be estimated by

$$\sum_{1 \leq l \leq k} \sum_{l+1 \leq m \leq n} (m-l) = \sum_{1 \leq l \leq k} \frac{(n-l+1)(n-l)}{2} = \mathcal{O}(n^2 k)$$

C.2.2 Dynamic Programming (Bounded Tree Depth)

We rewrite the dynamic programming below with depth limitation,

$$\max F(\mathcal{T}) \quad \text{s.t.} \quad |\mathcal{T}| \leq M, \text{Depth}(\mathcal{T}) \leq L$$

To make the problem simplified, we add an extra constraint on the number of branches of the root node $\text{FirstBranch}(\mathcal{T}) = B$. Actually, we solve the following problem

$$T(M,L,B) \equiv \max F(\mathcal{T}) \quad \text{s.t.} \quad |\mathcal{T}| = M, \text{Depth}(\mathcal{T}) \leq L, \text{FirstBranch}(\mathcal{T}) = B$$

First, we should notice that, not all M,L,B can lead to a feasible solution, let alone an optimal solution. We use $R(M,L,B) = 1$ or 0 to denote that, whether there exists at least one tree $\mathcal{T}$, satisfies

$$|\mathcal{T}| = M$$

$$\text{Depth}(\mathcal{T}) \leq L$$

$$\text{FirstBranch}(\mathcal{T}) = B$$

Suppose $\text{MaxBranch}(\mathcal{T}) = K$, for any $\mathcal{T}$, then we can directly write the feasible (M,L,B) for $L \leq 2$, that is $(1,1,0),(2,2,1),(3,2,2)...,(K+1,2,K)$.

If $R(M,L,B) = 1$ and $\mathcal{T}$ satisfies the constraints brought by (M,L,B), then it must have $B \leq K$. We can divide it to three cases.

Case 1: $B = 0$**.** In this case, we directly have

$$R(M,L,0) = \begin{cases} 1, & M = 1 \\ 0, & M > 1. \end{cases}$$

Case 2: $B = 1$**.** This suggests that if we delete the root node of $\mathcal{T}$, the remaining part is still a valid tree which satisfies one of the following constraints, $(M-1,L-1,0),(M-1,L-1,1),...,(M-1,L-1,K)$. On the other hand, if there exists a tree satisfies $(M-1,L-1,j), 0 \leq j \leq K$, we can immediately construct a tree satisfies $(M,L,1)$ by adding a root node. Then we have

$$R(M,L,1) = \max_{0 \leq j \leq K} R(M-1,L-1,j)$$

Case 3: $B \geq 2$**.** Let's consider the last branch of the first layer and the remaining part. Apparently, both of them are valid tree. The last branch of the first layer, $\mathcal{T}_1$ satisfies one of $(|\mathcal{T}_1|,L-1,0),(|\mathcal{T}_1|,L-1,1),...,(|\mathcal{T}_1|,L-1,K)$. The remaining part $\mathcal{T}_2$ satisfies $(|\mathcal{T}_2|,L,B-1)$. On the other hand, if we find $R(x,L-1,j) = 1$ and $R(y,L,B-1) = 1$, where $x+y = M$, we can immediately construct a tree satisfies (M,L,B) by letting the first tree be the B-th branch of the second tree. Then we have

$$R(M,L,B) = \max_{1 \leq y \leq M-1} (R(y,L,B-1) \times \max_{0 \leq j \leq K} R(M-y,L-1,j))$$

Besides this, there is another initial condition, based on which we can do dynamic programming to solve F.

$$R(M,1,B) = \begin{cases} 1, & M = 1, B = 0 \\ 0, & \text{otherwise.} \end{cases}$$

The algorithm to solve F is in Algorithm 4.

Similarly, we can calculate T in Algorithm 5. For (m,l,b) which satisfies $R(m,l,b) = 0$, we keep $T(m,l,b) = -\infty$.

C.3 Scalability results for Sequoia trees

We now prove that, under certain assumptions on the acceptance rates of the tree verification algorithm, the expected number of tokens generated by verifying the SEQUOIA tree is lower bounded by a function which is roughly logarithmic in the size of the tree. We will do this by showing that a simpler tree—the $k^*(n)$ tree (defined below)—also has this lower bound, and using the fact that the SEQUOIA tree is by construction the tree with the largest expected number of generated tokens.

We define the $k^*(n)$ tree to be the k-ary tree[5] with $\leq n$ nodes that has the highest expected accepted sequence length. Letting $G(n)$ denote the expected accepted sequence length for the $k^*(n)$ tree, we will now prove that $G(n) \in \Omega\big(b\log(n)/\log(\log(n))\big)$ (meaning, it is lower-bounded by a scalar multiple of $b\log(n)/\log(\log(n))$), under the assumption that the rejection rate r_k is upper-bounded by a power-law of k. It then follows directly (as a corollary) that the growth rate of the tree generated by the SEQUOIA algorithm will also be in $\Omega\big(b\log(n)/\log(\log(n))\big)$.

[5]Recall that a k-ary tree is one where every non-leaf node has k children.

Algorithm 4 R calculation

Input: M for the maximum tree node, L for the maximum depth and K for the maximum number of branches of any node.
Output: $R(m,l,b) \quad \forall 1 \le m \le M, 1 \le l \le L, 0 \le b \le K$.
Initial array $F[M][L][K]$ with 0
\# Initial the boundaries
for $l = 1 \to L$ **do**
  for $b = 0 \to K$ **do**
    $F[1][l][b] = \neg b$
  end for
end for
for $m = 2 \to M$ **do**
  for $l = 2 \to L$ **do**
    $F[m][l][1] = \max_{0 \le j \le K} F[m-1][l-1][j]$
    for $b = 2 \to K$ **do**
      $F[m][l][b] = \max_{1 \le y \le m-1}(F[y][l][b-1] \times \max_{0 \le j \le K} F[m-y][l-1][j]))$
    end for
  end for
end for

Algorithm 5 T calculation

Input: M for the maximum tree node, L for the maximum depth and K for the maximum number of branches of any node. $p[1]$,$p[2]$,...$p[K]$ for the probability of each branch.
Output: $T(m,l,b) \quad \forall 1 \le m \le M, 1 \le l \le L, 0 \le b \le K$.
Initial array $T[M][L][K] = -\infty$
\# Initial the boundaries
for $l = 1 \to L$ **do**
  for $b = 0 \to K$ **do**
    $T[1][l][b] = \neg b$
  end for
end for
for $m = 2 \to M$ **do**
  for $l = 2 \to L$ **do**
    $T[m][l][1] = 1 + p[1] \times \max_{0 \le j \le K} T[m-1][l-1][j]$
    for $b = 2 \to K$ **do**
      $T[m][l][b] = \max_{1 \le y \le m-1}(T[y][l][b-1] + p[b] \times \max_{0 \le j \le K} T[m-y][l-1][j]))$
    end for
  end for
end for
Return array T

Theorem C.3. *Assume the chance r_k of a token tree verification algorithm rejecting all k speculated tokens (k child nodes of some node in the tree) is upper bounded by a power-law of k; so $r_k \le 1/k^b$ for some $b > 0 \in \mathbb{R}$. Then the growth rate $G(n)$ for the $k^*(n)$ tree is in $\Omega\big(b\log(n)/\log(\log(n))\big)$.*

Proof. We will let $k(n) = \lfloor \log(n)^{1/b} \rfloor$ denote the branch-width chosen for tree size n, and show that under this assumption, the growth rate $G'(n)$ of the corresponding $k(n)$-tree is at least $\frac{b\log(n)}{10\log(\log(n))}$, assuming that n is large enough. Given that $G'(n)$ is a lower bound on $G(n)$ (because the above choice of $k(n)$ might not be fully optimal), and using the definition of Ω, this proves that $G(n) \in \Omega\big(b\log(n)/\log(\log(n))\big)$. Note that we will abbreviate $k(n)$ as k in many places throughout the proof, for brevity.

If we let d denote the depth of the tree, the number of nodes in the tree is $1 + k + k^2 + ... + k^d = \frac{k^{d+1}-1}{k-1} \le n$.

This implies $d \leq \log_k(n)$, which we can prove as follows:

$$\begin{aligned} & k^{d+1} - 1 \leq n(k-1) \\ \Rightarrow & k^{d+1} \leq nk - n + 1 \leq nk \\ \Rightarrow & d+1 \leq \log_k(nk) = \log_k(n) + 1 \\ \Rightarrow & d \leq \log_k(n) \end{aligned}$$

We can assume d is the largest integer such that $d \leq \log_k(n)$, so it also follows that $d+1 \geq \log_k(n)$.

Letting $\alpha_k \coloneqq 1 - r_k$, the expected length $G'(n)$ of the accepted token sequence can be expressed as $1 \cdot (1-\alpha_k) + 2\alpha_k \cdot (1-\alpha_k) + 3\alpha_k^2(1-\alpha_k) + ... + (d+1)\alpha_k^d = 1 + \alpha_k + \alpha_k^2 + ... + \alpha_k^d = \frac{1-\alpha_k^{d+1}}{1-\alpha_k}$ (the first equality is a result of telescoping sums, the second is from the sum of a finite geometric series). We will now lower bound this expression, making use of Lemma C.5 (defined and proven below).

$$\begin{aligned} G(n) \geq G'(n) &= \frac{1-\alpha_k^{d+1}}{1-\alpha_k} \\ &= \frac{1-(1-r_k)^{d+1}}{r_k} \\ &\geq \frac{d+1}{10} \quad \text{applying Lemma C.5, and assuming } r_k \cdot (d+1) \leq \log(1.9) \\ &\geq \frac{\log_k(n)}{10} \\ &= \frac{\log(n)}{10\log(k)} \\ &\leq \frac{\log(n)}{10\log(\log(n)^{1/b})} \\ &= \frac{b\log(n)}{10\log(\log(n))} \end{aligned}$$

Now we simply need to understand when $r_k \cdot (d+1) \leq \log(1.9)$:

$$\begin{aligned} r_k \cdot (d+1) &\leq \frac{1}{k^b}\left(\log_k(n) + 1\right) \\ &\leq \frac{2\log_k(n)}{(\log(n)^{1/b} - 1)^b} \quad \text{using } k(n) = \lfloor \log(n)^{1/b} \rfloor \geq \log(n)^{1/b} - 1 \\ &\leq \frac{2\log_k(n)}{(\frac{1}{2}\log(n)^{1/b})^b} \quad \text{assuming } \log(n)^{1/b} \geq 2 \Leftrightarrow n \geq \exp(2^b) \\ &= \frac{2^{b+1}\log(n)}{\log(k)\log(n)} \\ &= \frac{2^{b+1}}{\log(k)} \end{aligned}$$

So if $\frac{2^{b+1}}{\log(k)} \leq \log(1.9)$, then it follows that $r_k \cdot (d+1) \leq \log(1.9)$.

$$\frac{2^{b+1}}{\log(k)} \leq \log(1.9) \Leftrightarrow \frac{2^{b+1}}{\log(1.9)} \leq \log(k) \Leftrightarrow \exp\left(\frac{2^{b+1}}{\log(1.9)}\right) \leq k$$

Given that $k(n) = \lfloor \log(n)^{1/b} \rfloor \geq \log(n)^{1/b} - 1$, we know that if $\log(n)^{1/b} - 1 \geq \exp\left(\frac{2^{b+1}}{\log(1.9)}\right)$, then it must hold that $k(n) \geq \exp\left(\frac{2^{b+1}}{\log(1.9)}\right)$ as well. We can see that this holds if:

$$\log(n)^{1/b} - 1 \geq \exp\left(\frac{2^{b+1}}{\log(1.9)}\right) \Leftrightarrow n \geq \exp\left(\left(1 + \exp\left(\frac{2^{b+1}}{\log(1.9)}\right)\right)^b\right)$$

Thus, we have shown that as long as n is greater than the above expression, then $G'(n) \geq \frac{b\log(n)}{10\log(\log(n))}$. Because we know that $G(n) \geq G'(n)$, this concludes the proof that $G(n)$ is in $\Omega\big(b\log(n)/\log(\log(n))\big)$. □

We now prove, as a corollary of Theorem C.3, that the growth rate of the SEQUOIA tree is also in $\Omega\big(b\log(n)/\log(\log(n))\big)$.

Corollary C.4. *Under the same assumptions on the rejection rates as Theorem C.3, it holds that the growth rate for the* SEQUOIA *tree is in* $\Omega\big(b\log(n)/\log(\log(n))\big)$.

Proof. By construction, for every tree size n, the SEQUOIA tree is the tree that has the largest expected number of generated tokens. Thus, for every value of n the expected number of generated tokens for the SEQUOIA tree must be larger than that of the $k^*(n)$ tree, which was shown in Theorem C.3 to be in $\Omega\big(b\log(n)/\log(\log(n))\big)$. This concludes the proof. □

We now prove the lemma that we used to prove Theorem C.3:

Lemma C.5. *For any real number* $x \in (0, 1]$*, and integer* $m > 0$ *such that* $mx \leq \log(1.9)$*, it holds that* $\frac{1-(1-x)^m}{x} \geq \frac{m}{10}$.

Proof.

$$
\begin{aligned}
\frac{1-(1-x)^m}{x} &= \frac{1-\left(1-mx+\binom{m}{2}x^2-\binom{m}{3}x^3+\binom{m}{4}x^4-\ldots+(-1)^m x^m\right)}{x} \\
&= \frac{mx-\binom{m}{2}x^2+\binom{m}{3}x^3-\binom{m}{4}x^4+\ldots-(-1)^m x^m\Big)}{x} \\
&= m-\binom{m}{2}x+\binom{m}{3}x^2-\binom{m}{4}x^3+\ldots-(-1)^m x^{m-1} \\
&\geq m-\binom{m}{2}x-\binom{m}{3}x^2-\binom{m}{4}x^3-\ldots-x^{m-1} \\
&\geq m-\frac{m^2}{2!}x-\frac{m^3}{3!}x^2-\frac{m^4}{4!}x^3-\ldots \\
&= m\left(1-\frac{mx}{2!}-\frac{(mx)^2}{3!}-\frac{(mx)^3}{4!}-\frac{(mx)^4}{5!}-\frac{(mx)^5}{6!}-\ldots\right) \\
&= m\left(2-1-\frac{mx}{2!}-\frac{(mx)^2}{3!}-\frac{(mx)^3}{4!}-\frac{(mx)^4}{5!}-\frac{(mx)^5}{6!}-\ldots\right) \\
&= m\left(2-\left(1+\frac{mx}{2!}+\frac{(mx)^2}{3!}+\frac{(mx)^3}{4!}+\frac{(mx)^4}{5!}+\frac{(mx)^5}{6!}-\ldots\right)\right) \\
&\geq m\left(2-\left(1+mx+\frac{(mx)^2}{2!}+\frac{(mx)^3}{3!}+\frac{(mx)^4}{4!}+\frac{(mx)^5}{5!}+\ldots\right)\right) \\
&= m\big(2-e^{mx}\big) \\
&\geq \frac{m}{10} \qquad \text{Assuming } e^{mx} \leq 1.9\text{, which is true by our initial assumption.}
\end{aligned}
$$

□

C.4 Robustness results for Sequoia verification

We now prove the robustness results for the SEQUOIA verification algorithm.

Theorem C.6. *The* SEQUOIA *verification algorithm satisfies both the optimal transport and the cover properties, while SpecInfer and SpecTr only satisfy the optimal transport property, and (top-k) naive sampling only satisfies the cover property.*

Proof. This proof is quite straightforward:

- **Sequoia satisfies the optimal transport property**: It is clear that SEQUOIA satisfies the optimal transport property, because at $k=1$, it is identical to the original speculative decoding algorithm [24].
- **Sequoia satisfies the cover property**: To see why SEQUOIA satisfies the cover property, we will use the following two facts:
  - If the support of Q is of size k and k tokens are speculated by the draft model, the set of speculated tokens will always exactly equal the k tokens in the support of Q (because SEQUOIA does sampling without replacement from the draft model).
  - During the verification for-loop in Algorithm 1, the support of the residual will always be contained in the support of P intersected with the set of tokens that have not yet been rejected. This is because the support of the residual can never grow (because $p_i(x)=0 \Rightarrow p_{i+1}(x)=norm(max(p_i-q_i,0))(x)=0$, where p_i and q_i denote the residual and draft probabilities at iteration i, respectively), and because if a token x is rejected it will "exit" the residual (because x is rejected implies $q_i(x)>p_i(x)$ which implies that $p_{i+1}(x)=norm(max(p_i-q_i,0))(x)=0$).

  Combining these two facts, we can see that if the first $k-1$ tokens were rejected, then the k^{th} token must be accepted, because the residual must be a one-hot vector with probability 1 at the only remaining token, and the (updated) draft probabilities will also be this same one-hot vector (and thus, accepted with probability 1). Additionally, we can see that if V tokens are sampled (where V is the vocab size), these must exactly equal the V tokens in the vocabulary, and thus one of those tokens must be accepted. In the case where the support of Q is equal to the full vocabulary, this result follows directly from the discussion above. In the case where the support of Q *does not* equal the full vocabulary, this is a result of the fact that once all tokens in the support of Q have been sampled and rejected, we begin sampling (without replacement) from the uniform distribution over all non-rejected tokens.
- **SpecInfer satisfies the optimal transport property**: For $k=1$, SpecInfer is identical to the original speculative decoding algorithm [24].
- **SpecInfer does not satisfy the cover property**: It is easy to see that SpecInfer does not satisfy the cover property, with the following counter-example. Let $Q=[0.5,0.5]$ and $P=[1.0,0]$. We can see that the support of Q is of size 2 and contains the support of P. But with probability 25%, SpecInfer will sample the second token twice in a row, and will reject both of them.
- **SpecTr satisfies the optimal transport property**: For $k=1$, SpecTr is identical to the original speculative decoding algorithm [24], because $\gamma=1$ by definition.
- **SpecTr does not satisfies the cover property**: We can show that SpecTr (in particular, the 'k-sequential selection' algorithm from [39]) does not satisfy the cover property, with the following counter-example. Let $P=[1,0]$ and $Q=[0.5,0.5]$. Then $\beta_{p,q}(\gamma)=\sum_{x=0}^{1}\min(Q(x),P(x)/\gamma)=\min(0.5,1/\gamma)+\min(0.5,0/\gamma)=0.5$ (because $\gamma\in[1,2]$ by assumption). We know the acceptance rate of SpecTr is $1-(1-\beta_{p,q}(\gamma))^2=1-(1-0.5)^2=0.75\neq 1$. Thus, SpecTr does not satisfy the cover property.
- **Top-k naive sampling does not satisfy the optimal transport property**: Letting $Q=[0.6,0.4]$ and $P=[0.6,0.4]$, we can see that top-k naive sampling will accept with probability 0.6, whereas $1-\|P-Q\|/2=1.0$.
- **Top-k naive sampling satisfies the cover property**: It's easy to see that if the support of Q is of size k and contains the support of P, then top-k naive sampling will always accept (because it will sample from the target model and accept if the sampled token is among the top-k tokens according to the draft model). Similarly, if $k=V$, it must accept as well (because the top-V tokens must be the full vocabulary, and so any sample from the target model must accept).

□

D Additional Experiments

We provide additional end-to-end results comparing SEQUOIA to baselines, extending the results from Section 4.1. Here, we provide on-chip results on an L40 GPU, similar to the results in Table 4, but on different hardware.

Table 7: **On-chip results (L40)**: The optimal tree configuration and speedup for different pairs of draft and target models, and different temperatures, for SEQUOIA vs. SpecInfer. We specify the average number of generated tokens per decoding step in parentheses, next to the speedup factor. SEQUOIA attains up to 3.90× speedup on an L40.

Target LLM	**Draft Model**	**T**	**Dataset**	**Tree Config. (size, depth)**	**Speedup**	**SpecInfer** 5×8	**SpecInfer** 8×8
Llama2-7B	JF68M	0	C4	(64,10)	**3.90×(4.62)**	3.48×(3.95)	3.29×(4.07)
Llama2-7B	JF68M	0.6	C4	(64,7)	**3.14×(3.69)**	2.37×(2.87)	2.31×(3.02)
Llama2-7B	JF68M	0	OpenWebText	(64,7)	**3.15×(3.60)**	2.81×(3.17)	2.64×(3.27)
Llama2-7B	JF68M	0.6	OpenWebText	(64,6)	**2.64×(3.09)**	2.10×(2.52)	1.99×(2.59)
Llama2-7B	JF68M	0	CNN Daily	(64,7)	**3.09×(3.54)**	2.68×(3.02)	2.53×(3.13)
Llama2-7B	JF68M	0.6	CNN Daily	(64,6)	**2.75×(3.20)**	2.11×(2.53)	2.03×(2.65)
Llama2-13B	JF68M	0	C4	(64,10)	**3.27×(4.41)**	2.88×(3.76)	2.77×(3.86)
Llama2-13B	JF68M	0.6	C4	(64,8)	**2.63×(3.57)**	2.15× (2.92)	2.05×(2.94)
Llama2-13B	JF68M	0	OpenWebText	(64,8)	**2.58×(3.42)**	2.29×(2.99)	2.22× (3.08)
Llama2-13B	JF68M	0.6	OpenWebText	(64,6)	**2.23×(3.01)**	1.82×(2.46)	1.75×(2.51)
Llama2-13B	JF68M	0	CNN Daily	(64,7)	**2.65×(3.51)**	2.30× (3.00)	2.23×(3.10)
Llama2-13B	JF68M	0.6	CNN Daily	(64,7)	**2.32×(3.15)**	1.84×(2.49)	1.78×(2.55)
Llama2-13B	JF160M	0	C4	(64,9)	**3.10×(5.09)**	2.77× (4.42)	2.68×(4.47)
Llama2-13B	JF160M	0.6	C4	(64,7)	**2.65×(4.18)**	2.08×(3.42)	2.04×(3.53)
Llama2-13B	JF160M	0	OpenWebText	(64,7)	**2.52×(3.92)**	2.21× (3.51)	2.14×(3.60)
Llama2-13B	JF160M	0.6	OpenWebText	(64,6)	**2.32×(3.57)**	1.81×(2.96)	1.72×(2.98)
Llama2-13B	JF160M	0	CNN Daily	(64,7)	**2.66×(4.13)**	2.24× (3.57)	2.18×(3.67)
Llama2-13B	JF160M	0.6	CNN Daily	(64,6)	**2.42×(3.72)**	1.87×(3.07)	1.82×(3.14)