Decoding Speculative Decoding

Minghao Yan[*], **Saurabh Agarwal**, **Shivaram Venkataraman**
Department of Computer Sciences
University of Wisconsin-Madison

Abstract

Speculative Decoding is a widely used technique to speed up inference for Large Language Models (LLMs) without sacrificing quality. When performing inference, speculative decoding uses a smaller *draft* model to generate speculative tokens and then uses the *target* LLM to verify those draft tokens. The speedup provided by speculative decoding heavily depends on the choice of the draft model. In this work, we perform a detailed study comprising over 350 experiments with LLaMA-65B and OPT-66B using speculative decoding and delineate the factors that affect the performance gain provided by speculative decoding. Our experiments indicate that the performance of speculative decoding depends heavily on the latency of the draft model, and the draft model's capability in language modeling does not correlate strongly with its performance in speculative decoding. Based on these insights we explore a new design space for draft models and design hardware-efficient draft models for speculative decoding. Our newly designed draft model can provide 111% higher throughput than existing draft models and our approach generalizes further to all LLaMA models (1/2/3.1) and supervised fine-tuned models.

1 Introduction

In recent years, Large Language Models (LLMs) have emerged as a cornerstone of modern computational linguistics, offering unprecedented capabilities in generating and interpreting human language. As the demand for faster and more efficient language processing grows, understanding and optimizing the inference throughput of these models becomes increasingly crucial. Decoder-only LLMs (Brown et al., 2020; Touvron et al., 2023a,b) use autoregressive decoding to perform inference. Autoregressive decoding is known to be hardware inefficient (Miao et al., 2023; Liu et al., 2023a), leading to poor resource utilization and low throughput during inference.

Several methods (Yu et al., 2022; Wang et al., 2020; Kwon et al., 2023; Dao et al., 2023; Hong et al., 2023) have been studied to optimize the serving of LLMs. One promising approach to improve the throughput for serving LLMs without accuracy loss is speculative decoding (Stern et al., 2018; Xia et al., 2023a; Leviathan et al., 2023). When using speculative decoding to serve an LLM (usually 10s to 100s of billion parameters), a draft model (a significantly smaller LLM) is used to generate speculative tokens. The target LLM model then verifies the output of the draft model and only outputs tokens that match its output. In the case of speculative decoding, the target LLM for inference acts as a *verifier* for the draft model. By leveraging faster inference of smaller draft models, speculative decoding turns autoregressive decoding on the target LLM into a more hardware-friendly batched operation (similar to "prefill"), thereby increasing throughput while preserving accuracy.

Given the promised benefits of speculative decoding, this paper first focuses on understanding the key factors that dictate the throughput improvements that can be obtained. We perform a comprehensive benchmarking study and profile speculative decoding to characterize bottlenecks. We perform over 350 experiments, using LLMs like LLaMA-65B, OPT-66B, and fine-tuned chat models such as Vicuna-33B (Chiang et al., 2023) as target models and LLaMA and OPT families as draft models, ranging from $\approx 5\times$ to $528\times$ fewer parameters than the target models. Our findings show that the key bottleneck in speculative decoding is the *draft model's latency* (time to generate candidate tokens from the draft model), highlighting the need to study the design of draft models for speculative decoding.

While studying the design of draft models, we observed two interesting phenomena: First we ob-

[*] Correspondence: Minghao Yan <myan@cs.wisc.edu>

Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), pages 6460–6473
April 29 - May 4, 2025 ©2025 Association for Computational Linguistics

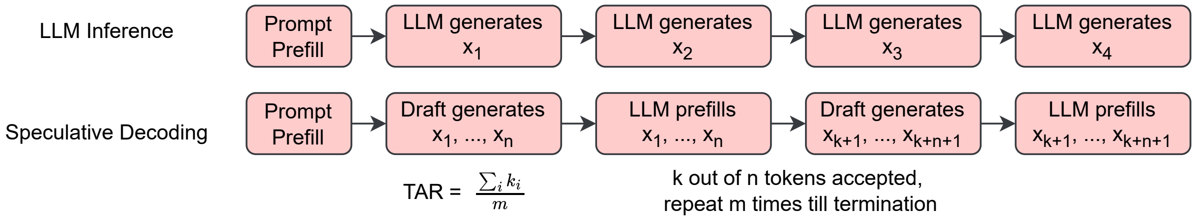

LLM Inference
Prompt Prefill
LLM generates x_1
LLM generates x_2
LLM generates x_3
LLM generates x_4
Speculative Decoding
Prompt Prefill
Draft generates $x_1, ..., x_n$
LLM prefills $x_1, ..., x_n$
Draft generates $x_{k+1}, ..., x_{k+n+1}$
LLM prefills $x_{k+1}, ..., x_{k+n+1}$
TAR = $\frac{\sum_i k_i}{m}$
k out of n tokens accepted, repeat m times till termination

Figure 1: This figure shows the speculative decoding process. In vanilla LLM inference, after the prompt is processed into KV caches (Prefill), LLM generates the output token by token (Autoregressive generation). In speculative decoding, a draft model is first used to generate n candidate tokens at each step (Draft token generation). The LLM verifies the candidate tokens and accepts k ($k \leq n$) tokens (LLM verification).

serve that the draft model latency is bottlenecked by model depth, and higher model depth leads to increased latency (Section 3.2) even with the same number of parameters. Secondly, we observe that draft model accuracy on language modeling tasks does not correlate strongly with its performance in speculative decoding (Section 3.3), *i.e.*, a draft model with higher accuracy on language modeling task can have similar TAR to a model with lower accuracy. These two phenomena show that existing draft models used for speculative decoding were primarily designed only to achieve maximum accuracy for a given parameter budget and are sub-optimal for maximizing the throughput with speculative decoding.

Based on these two insights, we propose designing new draft models that trade increased depth for width (thus retaining the same parameter count) and show that our new draft models can boost inference throughput using speculative decoding by over 60%. Finally, we show how pruning methods like Sheared-LLAMA (Xia et al., 2023b) can be used to generate smaller draft models with favorable configurations on three different families of models OPT, LLAMA, and LLAMA-3.1.

Our Contributions:

- To the best of our knowledge, we are the first work to conduct comprehensive experiments on serving the open source LLAMA-65B and OPT-66B models utilizing speculative decoding, conducting more than 352 experiments to elucidate the factors one needs to consider while selecting and designing a draft model.

- We show a systematic redesign of draft models used for speculative decoding is needed to maximize the efficiency of speculative decoding. We demonstrate that using accuracy on language modeling tasks to choose the draft model for speculative decoding leads to suboptimal choices. By redesigning draft models, we improved speculative decoding throughput by up to 111%.

- We show that our design leads to a 37% reduction in KV-Caches, enabling larger batch sizes, and also outperforms other popular methods like self-speculative decoding (Zhang et al., 2023) in several different setups.

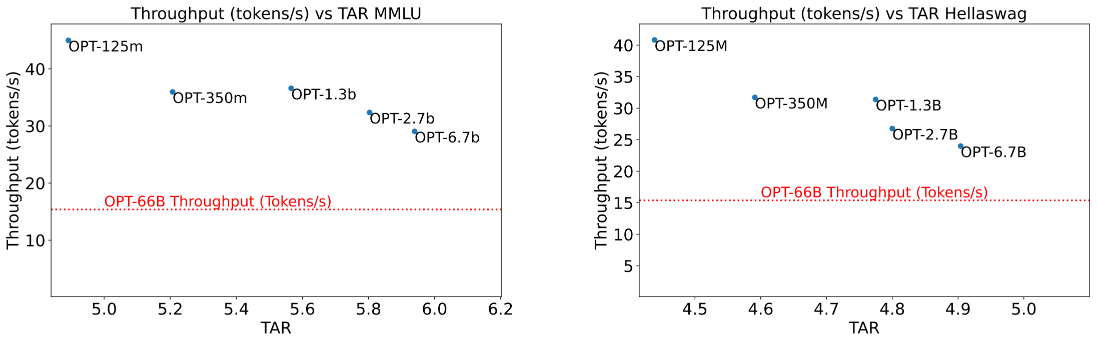
2 Background and Related Work

First, we provide a high-level overview of LLM inference and the use of speculative decoding.

2.1 Background

A decoder-only LLM performs inference in two phases: prefill and autoregressive decoding. In the prefill phase, the LLM is initialized with a context or prompt, formulated as $C = \{c_1, c_2, ..., c_n\}$, where C represents the input context and n the length of the prefill. In the prefill phase, the model processes the whole input context in parallel and performs next-word prediction. During the autoregressive decoding, the model generates new text sequentially, a token at a time, building upon the context provided in the prefill phase. Due to its sequential nature, the autoregressive decoding phase is widely known to be memory bandwidth bound on modern GPUs (Leviathan et al., 2023).

To improve hardware utilization and throughput, (Leviathan et al., 2023) and (Chen et al., 2023) proposed *speculative decoding*, where a smaller *draft* model generates multiple tokens, and the *target* LLM verifies the generated tokens in parallel. The verification is akin to *prefill* stage in LLM inference. As long as more than one token is accepted on average, speculative decoding can potentially provide speedups. Figure 1 shows how inference



(a) OPT Series on MMLU

(b) OPT Series on Hellaswag

Figure 2: This figure shows the throughput of different draft models from the OPT series. As model size increases, throughput decreases due to higher inference latency despite consistent increases in TAR.

using speculative decoding differs from autoregressive decoding. It is widely reported (Miao et al., 2023; Liu et al., 2023a) that the number of tokens accepted by the target model affects the speedup provided by speculative decoding.

In this work, we conduct a comprehensive empirical study to identify the performance bottleneck of speculative decoding and identify strategies to design the best draft model for a given LLM.

2.2 Related Work

LLM Inference: There has been significant amount of work on improving LLM serving including work in Orca (Yu et al., 2022), LightSeq (Wang et al., 2020), DeepSpeed Inference (Aminabadi et al., 2022), PagedAttention (Kwon et al., 2023), FlashDecoding (Dao et al., 2023) and FlashDecoding++ (Hong et al., 2023). These works seek to improve LLM inference by better utilization of hardware. Other lines of work have looked at pruning LLMs based on input context to speed up inference (Liu et al., 2023b) or using shallower and wider neural networks for machine translation (Kasai et al., 2020). However, in this work, we focus on speculative decoding (Leviathan et al., 2023; Chen et al., 2023; Santilli et al., 2023), which has been inspired by speculative execution in hardware (Hennessy and Patterson, 2011).

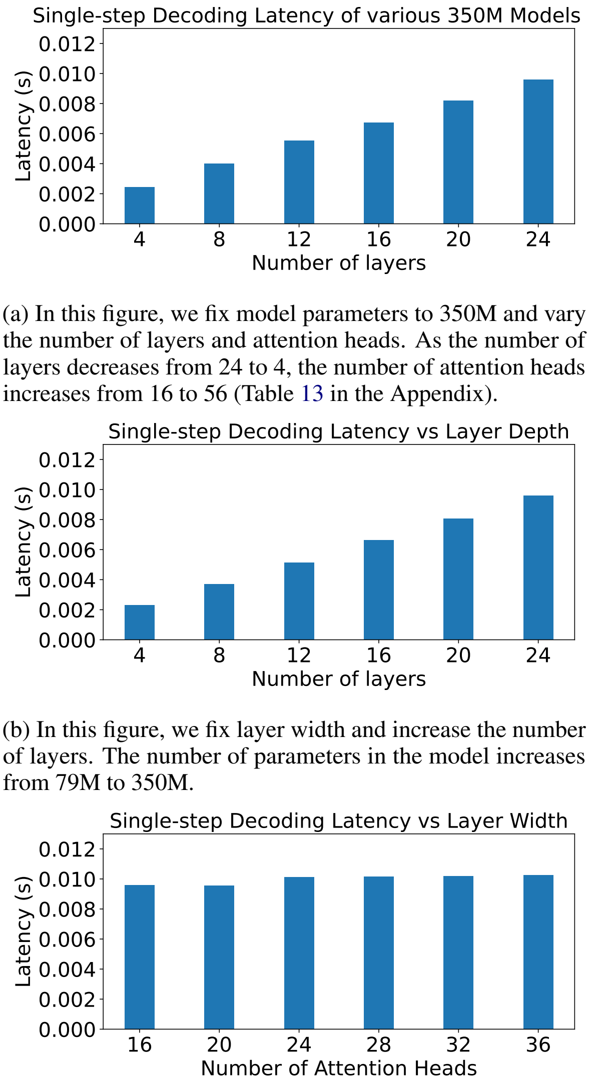
Speculative Decoding: Several prior works have studied ways to improve speculative decoding. Liu et al. (2023a) seeks to continuously train the draft model on the output of the target model to improve the token acceptance rate. However, training on the same hardware during inference can be challenging. Predictive Pipeline Decoding (PPD) (Yang et al., 2023) introduced the use of early exit (Schuster et al., 2022) from the target model to obtain draft tokens. Similar to PPD, Draft&Verify (Zhang et al., 2023) seeks to combine the use of early exit with speculative decoding, where the early exit (Schuster et al., 2022; Bae et al., 2023) from the target model acts as a draft token. A drawback of these methods is that the maximum benefit in latency is capped. For example, in speculative decoding, we can use draft models that are orders of magnitude (e.g., ≈100x-1000x) smaller than the target model, while early exit methods usually exit after performing inference over at least a fourth of the model (Schuster et al., 2022), thus, limiting the gain in throughput. Other lines of work, such as Medusa (Cai et al., 2024), propose fine-tuning multiple generation heads within the LLM that do not match the LLM output distribution exactly but maintain the generation quality. In addition, other works have looked at improving speculative decoding via learning an encoder-decoder-based draft model (Xia et al., 2023a), generating multiple draft tokens (Sun et al., 2024b; Yang et al., 2024), draft model distillation (Zhou et al., 2024), or focusing on long-context scenarios (Sun et al., 2024a; Chen et al., 2024).

In this work, we aim to understand how the choice of draft model affects the throughput provided by speculative decoding. We use insights from benchmarking to design draft models that maximize speculative decoding throughput.

3 Understanding Speculative Decoding

To study the effects of the choice of the draft model, we first perform a detailed study on serving OPT-65B and LLaMA-65B using speculative decoding.

Setup: We implement speculative decoding in the Microsoft Deepspeed library (Microsoft, 2023).



(a) In this figure, we fix model parameters to 350M and vary the number of layers and attention heads. As the number of layers decreases from 24 to 4, the number of attention heads increases from 16 to 56 (Table 13 in the Appendix).

(b) In this figure, we fix layer width and increase the number of layers. The number of parameters in the model increases from 79M to 350M.

(c) In this figure, we fix model depth and increase the number of attention heads in each layer. The number of parameters in the model increases from 350M to 1B.

Figure 3: This figure shows microbenchmarks on how model depth and width affect decoding latency.

We use the same setup as SpecInfer (Miao et al., 2023), first using the draft model to generate draft tokens and then using the target model to verify the output of the draft model. We set the batch size to 1 and use greedy decoding. For all our experiments, we use 4 Nvidia 80GB A100 GPUs. We perform our experiment on the OPT and LLaMA base models (Zhang et al., 2022; Touvron et al., 2023a) on MMLU (Hendrycks et al., 2020), Hellaswag (Zellers et al., 2019), and Chatbot Arena datasets (Zheng et al., 2023). For MMLU, we use the standard 5-shot setup. The remaining datasets were evaluated in a zero-shot setting. Note that since our goal is to test our draft model's ability to study the target model's behavior, we do not instruct the model to emit a single-letter answer to MMLU and Hellaswag questions, but instead opt for an open-ended generation approach where the model can provide as much as explanation as it sees fit, which aligns better with a real-world chatbot setting. For MMLU, we feed the model with the question and choices; for Hellaswag, we feed the model with incomplete sentences without the choices. We use OPT-66B and LLaMA-65B as the target LLM for OPT and LLaMA series and use OPT-125M, 350M, 1.3B, 2.7B, and 6.7B variants as draft models for OPT series, and LLaMA-7B and 13B as draft models for LLaMA series.

Metrics: To quantify the performance of different draft models when performing inference on a target model, we measure throughput (tokens generated per second) and TAR (Figure 2). We note that the primary goal of speculative decoding is to improve throughput.

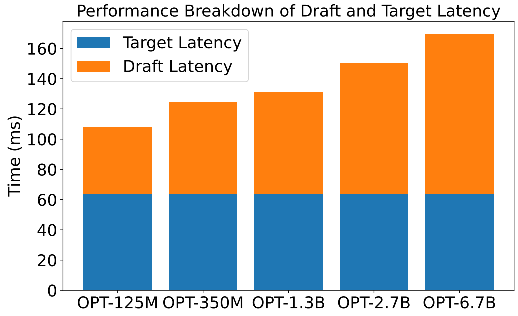
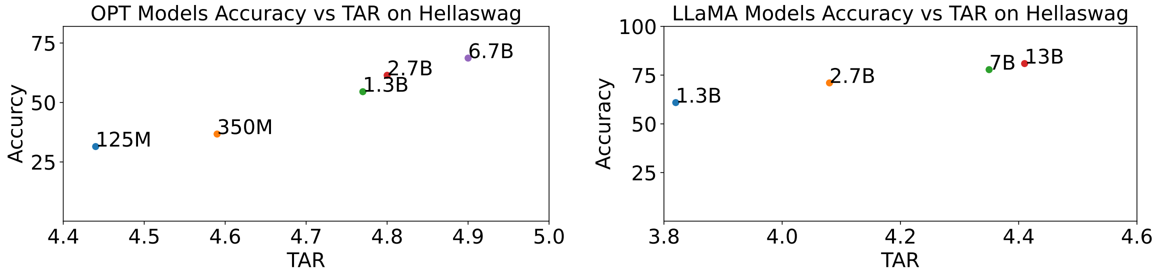


Figure 4: This figure shows the performance breakdown of speculative decoding on OPT models, look ahead length is set to be optimal for each draft model found empirically.

3.1 Bottlenecks in Speculative Decoding

To understand the throughput of LLMs, we first plot a latency breakdown of speculative decoding in Figure 4. We show the latency breakdown between the draft token generation phase and the target model verification phase for serving OPT-66B model when using various variants of OPT as the draft model. A similar figure for LLaMA models (Figure 9) can be found in the Appendix.

In Figure 4, the time taken by the draft model for each token generation step increases with an increase in model sizes, going from 6.23 ms for OPT-125M to 18.56 ms for OPT-6.7B (Table 11 in the Appendix). However, even the smallest draft model, OPT-125M, still takes significant time in a speculative decoding iteration to perform draft



(a) Model accuracy vs TAR for OPT models

(b) Model accuracy vs TAR for LLaMA models

Figure 5: This figure shows the task accuracy versus TAR for OPT and LLaMA models on Hellaswag. The accuracy numbers are obtained from OpenLLM Leaderboard (HuggingFace, 2023).

model autoregressive decoding. Though the target LLM has a higher latency in each decoding iteration, it only has to perform one prefill operation on the entire candidate token sequence. In contrast, the draft model has to perform multi-step autoregressive decoding sequentially, creating a bottleneck. This highlights why draft model latency is one of the key bottlenecks in speculative decoding performance. We note that while Figure 4 uses look ahead values (the number of tokens generated by the draft model) from 6 to 8, depending on the draft model, even if we scale look ahead length to hundreds of tokens, the target model verification time stays constant. The draft model latency remains the bottleneck due to the difference in efficiency between prefill and autoregressive decoding. Next, we investigate how to reduce draft model latency.

3.2 Understanding Draft Model Latency

When studying the breakdown in latencies for speculative decoding in the previous section, we observed something intriguing in Figure 4. We see that OPT-350M has a similar draft-model latency as OPT-1.3B, a model almost four times its size. This indicates that OPT-350M is inefficient, and we can design better models.

We perform three microbenchmarks to validate our hypothesis and analyze decoding throughput: First, we fix the total model parameters at 350M and see how changing layer width and depth would affect decoding latency. Then, we fix either the layer width or depth to be the same as in OPT-350M and modify the other to see how latency scales with wider layers or shallower models.

Figure 3 shows the results of these three benchmarks. In the first benchmark (Figure 3a), we vary the number of attention heads, feed-forward dimension, and layers in a model to keep the model parameters at around 350M. The detailed configuration for each model can be found in Table 13 in the Appendix. The plot shows that autoregressive decoding latency scales linearly with layer depth despite similar total model parameters.

The same is true for the second benchmark (Figure 3b). The original OPT-350M model has 24 layers. As we reduce the number of layers while keeping all other configurations the same, the autoregressive decoding latency decreases linearly. On the other hand, the third benchmark (Figure 3c) shows that as we scale the number of attention heads up from the original OPT-350's 16 heads to 36 heads, the decoding latency stays almost constant even if layer width has doubled.

These experiments indicate more latency-efficient model architectures with the same parameter budget exist. Changing the number of layers and attention heads not only changes the throughput but also affects the quality of predictions made by the model. We will next study how changes in model depth and width affect model accuracy and TAR and the correlation between them.

3.3 Understanding Draft Model TAR

In prior work (Leviathan et al., 2023), speculative decoding throughput is modeled by $\frac{1-\alpha^{\gamma+1}}{(1-\alpha)(\gamma c+1)}$, where $\frac{1-\alpha^{\gamma+1}}{1-\alpha}$ represents the improvement factor (expected number of tokens matched in each iteration) and $\gamma c + 1$ represents the combined latency of draft and target models. Therefore, tokens accepted per iteration (also known as TAR) have a linear effect on speculative decoding throughput.

In this section, we perform experiments to understand the correlation between the accuracy of a model on popular NLP tasks and its TAR. We plot the accuracy of a model against the TAR it achieves in Figure 5. Surprisingly, we find that

TAR correlates little to the model's accuracy on a task. We believe this lack of correlation is due to the majority of tokens in a sentence not being content words (Chen et al., 2023), which do not affect the model's accuracy on a specific task. Results on more datasets can be found in the Appendix (Figure 8).

Combining insights from these experiments, we observe that current draft models are not designed to maximize speculative decoding throughput. Next, we will show how to design new draft models that outperform existing models.

4 Draft Model Design for Speculative Decoding

The above results indicate that to improve the throughput of speculative decoding, it is necessary to improve the latency of draft models, *i.e.*, can we design a model that provides a similar TAR at a lower inference cost? In the next section, we study the possibility of such a design.

Table 1: This table shows the model configuration of the two pruned models. Here l represents the number of layers, h is the number of attention heads, d_{inter} is intermediate size, and d_{model} is model dimension.

Model	l	h	d_{inter}	d_{model}
NoFT-1.3B	24	16	5504	2048
NoFT-Wide-1.3B	12	20	9280	2560
NoFT-Wide-796M	5	32	11008	4096
NoFT-Wide-543M	3	32	11008	4096
NoFT-Wide-290M	1	32	11008	4096

4.1 Draft Model Design

In section 3.1, we show that model depth bottlenecks draft model latency, while in section 3.3, we show that a draft model's performance in speculative decoding is largely irrelevant to its accuracy on language modeling. These two insights prompted us to test if we can build a wider and shallower network and study how it affects latency and TAR.

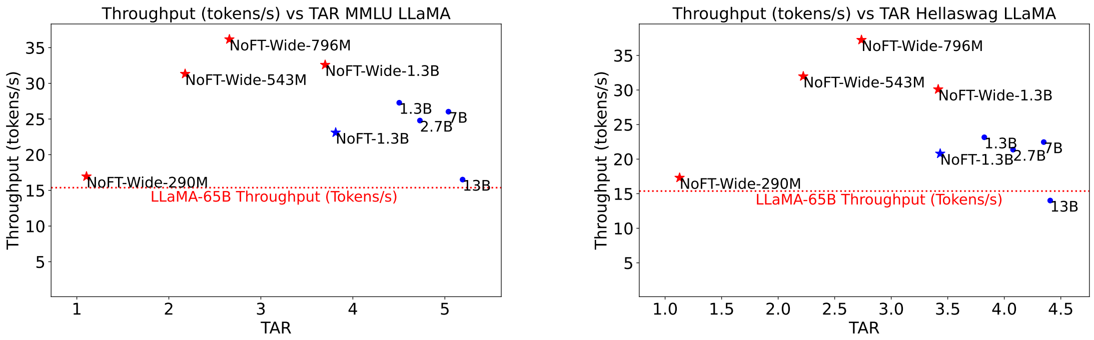
Method: We leverage recent advances in structured LLM pruning, Sheared-LLaMA (Xia et al., 2023b), which provides a framework to prune larger models to a specified smaller configuration. Sheared-LLaMA (Xia et al., 2023b) learns layers, attention heads, and neurons to mask from the large model to prune it into the specified small model. The flexibility enables us to prune LLaMA-7B into desirable model configurations. In our experiments, we pruned our models from LLaMA-7B using 0.4B tokens sampled from the RedPajama Dataset (Computer, 2023) following Xia et al. (2023b) but skipped the expensive fine-tuning step on 50B more tokens (and hence the name NoFT). We find that this is sufficient to achieve a significantly higher throughput.

Deep vs wide model comparison: Our goal is to start with LLaMA-7B and produce a wider version of Sheared-LLaMA-1.3B while keeping the number of parameters the same as in Sheared-LLaMA-1.3B. We choose Sheared-LLaMA-1.3B since it achieves the highest throughput in our benchmark among existing models (blue dots in Figure 6). We use two configurations: the first configuration was provided by the Sheared-LLaMA authors (NoFT-1.3B), and we designed the second configuration (NoFT-Wide-1.3B) to optimize for better speculative decoding throughput. Table 1 shows the detailed configuration of the two models. We slash the number of layers by half, from 24 to 12, and keep the total parameter count roughly the same by increasing the intermediate size from 5504 to 9280, the number of attention heads from 16 to 20, and the corresponding model dimension from 2048 to 2560. Figure 6a, 6b, and 7b show that we can achieve up to 30% higher speculative decoding throughput using only 0.8% of tokens used to train Sheared-LLaMA-1.3B.

Table 3 also shows the latency and TAR of the two sheared models on MMLU. The deep variant (NoFT-1.3B) can achieve 3% higher TAR, but the wide variant (NoFT-Wide-1.3B) reduces draft latency by 49%, improving overall throughput by 41%. We found results are very similar for other datasets, such as Chatbot Arena (Figure 7b in the Appendix) and Hellaswag (Figure 6b). This experiment shows a need to rethink the model design space for speculative decoding, where we should specifically design models for higher throughput.

Draft model scaling: To understand the limitation of draft model depth-width tradeoff in speculative decoding, we created three configurations, NoFT-Wide-796M, 543M, and 290M, that use the same number of attention heads, intermediate size, and model dimension as LLaMA-7B, but reduce the number of layers to 5, 3, and 1, respectively. This is the widest configuration possible using the Sheared-LLaMA pruning scheme.

Figure 6 shows that the NoFT-Wide-796M model provides another 20% improvement in throughput over NoFT-Wide-1.3B and up to 60% throughput improvement over the existing Sheared-



(a) LLaMA Series on MMLU

(b) LLaMA Series on Hellaswag

Figure 6: This figure shows the throughput scaling of different draft models from the LLaMA series on MMLU and Hellaswag. Asterisks represent models that are pruned but not fine-tuned. The red asterisks represent model configurations that we designed.

Table 2: This table shows the throughput of speculative decoding (tokens/s) with temperature sampling on various datasets and models. Our NoFT-Wide-796M model achieved 97.7% to 111.8% higher throughput compared to the existing Sheared-LLaMA-1.3B model while only using 0.8% of its training tokens.

	Temperature = 1.0			Temperature = 0.5		
Draft Model	MMLU	Hellaswag	Chatbot Arena	MMLU	Hellaswag	Chatbot Arena
NoFT-Wide-543M	20.54	24.08	23.77	26.08	25.77	25.31
NoFT-Wide-796M	**24.32**	**24.32**	**24.18**	**29.64**	**26.91**	**25.76**
NoFT-Wide-1.3B	22.99	24.08	**24.22**	28.43	26.39	**25.85**
Sheared-LLaMA-1.3B	12.06	12.30	11.45	14.60	13.45	12.66
Speedup	101.6%	97.7%	111.8%	103.0%	100.0%	103.5%

Table 3: This table shows the TAR, per-step decoding latency, and speculative decoding throughput using the two pruned draft models.

Draft Model	TAR	Latency (ms)	Throughput (tokens/s)
NoFT-1.3B	**3.81**	13.13	23.10
NoFT-Wide-1.3B	3.70	**6.69**	**32.59**

LLaMA-1.3B model. Though the smaller NoFT-Wide-543M provides up to 40% throughput improvements over Sheared-LLaMA-1.3B, it has a lower throughput than NoFT-Wide-796M.

Results in Figure 6 show that reducing the layer count to less than 5 layers would cause the model's alignment capability to reduce dramatically. In addition, as we reduce models to 5 layers, target model latency takes more than 80% of the time in a decoding cycle. Therefore, further reducing the latency would only provide a marginal gain in overall decoding latency since the target model latency remains constant. In this case, the drop in TAR significantly outweighs the latency gain, causing decoding throughput to decrease.

We also verified our model's robustness under different temperatures. We performed experiments with temperature sampling by choosing temperatures of 0.5 and 1 following Leviathan et al. (2023). In Table 2, we show that our model achieves even higher speedup than existing models when stochasticity is introduced to the decoding process. On three datasets we achieved from 97% to 111% higher throughput using our NoFT-Wide-796M model compared to using an existing fine-tuned model. The speedups in different temperature settings show that our speedups generalize across different sampling methods.

Table 4: This table shows the memory usage of our draft model LLaMA-Wide-1.3B versus Sheared-LLaMA-1.3B. Our draft model reduces KV cache by 37%.

	KV cache		Activations	
Context Length	Sheared	Wide	Sheared	Wide
256	48MB	**30MB**	1MB	1.25MB
512	96MB	**60MB**	2MB	2.5MB
1024	192MB	**120MB**	4MB	5MB

Batch size and KV cache: By designing a shallower model with wider MLP layers, we reduce the KV-cache size required by the model. This

Table 5: This table shows the prefill and per-step autoregressive decoding latency for different batch sizes.

Batch size	Prefill (s)	Autoregressive decoding (s)	
Model	LLaMA-65B	LLaMA-NoFT Wide-1.3B	Sheared LLaMA-1.3B
1	0.060	**0.0097**	0.018
4	0.061	**0.0097**	0.019
16	0.063	**0.0099**	0.019
32	0.065	**0.0101**	0.019

allows us to increase the batch size we can accommodate. Table 4 shows that for the same sized 1.3B model, our wider design reduces KV-Cache by more than 37%. Table 5 demonstrates that increasing the batch size has minimal impact on autoregressive decoding latency, thereby increasing throughput.

LLaMA-3 Results We apply our approach to the newest LLaMA-3.2-1B model. All other experiment settings follow section 4.1. We evaluated on MMLU, Hellaswag, and Chatbot Arena prompts. We compare our Wide-829M model against LLaMA-3.2 and self-speculative decoding. Since LLaMA-3 is trained on more tokens than LLaMA-2 (15 trillion vs 2 trillion), pruning its layers had a larger impact on TAR. We follow prior work (Muralidharan et al., 2024) to recover the TAR after pruning by distilling it over 1 million tokens. Note that this process can be finished within 10 minutes. In this experiment, we use LLaMA-3.1-8B as the target model. Table 6 shows that our pruned Wide-829M model achieves 42.6% higher throughput compared to LLaMA-3.2-1B and 51.8% higher throughput compared to self-speculative decoding. This experiment shows that our approach works well on SoTA models.

Table 6: This table shows the speculative decoding throughput and the latencies of our Wide-829M models against LLaMA-3.2-1B model and self-speculative decoding.

	Throughput(tokens/s)		
Draft Model	Chat	MMLU	Hellaswag
Wide-829M	**53.35**	**61.77**	**60.44**
LLaMA-3.2-1B	37.41	44.89	39.03
Self-Speculative	35.14	40.41	40.02

4.2 Ablation Studies

In this section, we study if varying the decoding sampling methods or using a different or supervised fine-tuned target model would affect our draft model's performance.

Varying the target model: Prior experiments are performed with LLaMA-65B as the target model. As newer generations of models roll out, we would like to see if our conclusion holds on newer generations of models. In this ablation study, we evaluate our best NoFT-Wide-796M model against the LLaMA-2-70B model. Table 7 shows that though our NoFT-Wide-796M is distilled from LLaMA-7B, it can achieve a similar token acceptance rate when the target model is from LLaMA-2 family. This shows that our distilled model can be applied to various models based on similar training recipes and tokenizers. We also show more results on the newest LLaMA-3.1 and LLaMA-3.2 families in Appendix 4.1.

Table 7: This table shows the tokens accepted per iteration when we use different target models. The draft model we use is NoFT-Wide-796M.

Target Model	MMLU	Hellaswag	Chatbot Arena
LLaMA-65B	2.66	2.74	2.61
LLaMA-2-70B	2.55	2.68	2.64

Supervised fine-tuned models: Prior experiments are performed on base models to study the scaling of draft models. In practice, supervised fine-tuned models are adopted for their better instruction-following capabilities. In this section, we compare our best NoFT-Wide-796M model to Tiny-LLaMA-1.1B with Vicuna 33B as the target model. Note that our NoFT-Wide-796M is pruned from the base version of LLaMA-7B without fine-tuning. Table 8 shows that NoFT-Wide-796M outperforms Tiny-LLaMA-1.1B in all cases by up to 45%. While Tiny-LLaMA-1.1B has a TAR 35% and 32% higher than NoFT-Wide-796M on MMLU and Hellaswag, respectively, its latency is 4x higher due to having 22 layers in the model compared to NoFT-Wide-796M with merely 5 layers. This ablation study also demonstrates how speculative decoding is bottlenecked by draft model depth and that a draft model obtained from the non-fine-tuned base model, when appropriately designed, can outperform fine-tuned models.

Comparison against self-speculative decoding: Self-speculative decoding (Zhang et al., 2023) was proposed to leverage the base model to avoid the need to train a draft model. However, we show that we achieve significantly higher throughput with our proposed model architecture design. We pick the settings where self-speculative decoding

Table 8: This table shows the throughput of speculative decoding (tokens/s) with Vicuna 33B as the target model.

Draft Model	MMLU	Hellaswag	Chatbot Arena
Tiny-LLaMA-1.1B	20.78	18.25	18.73
NoFT-Wide-796M	**29.87**	**26.55**	**25.61**

achieves its highest speedup (LLaMA-2-70B as target model with greedy sampling on CNNDM and XSum datasets). However, our LLaMA-NoFT-Wide-796M outperforms self-speculative decoding by up to 53%. This is because our LLaMA-NoFT-Wide-796M model's autoregressive decoding latency is only 7% of the target 70B model's prefill latency. Self-speculative decoding suffers from a significant drop in drafting accuracy after dropping more than 42 layers out of 80 in LLaMA-2-13B (Zhang et al., 2023). Table 10 shows that on a 70B target model, they would still need to compute 33 attention layers and 43 MLP layers to obtain a draft response, while our LLaMA-NoFT-Wide-796M only has 5 attention and MLP layers each. Therefore, we obtain a much higher throughput compared to self-speculative decoding. More comparisons on LLaMA-3 are presented in Appendix 4.1.

Table 9: This table shows the number of layers used in self-speculative decoding and our draft model.

Model Setup		Layers	
		Attention	MLP
Self-Speculative	LLaMA-2-13B	16	30
	LLaMA-2-70B	33	43
Draft Model	NoFT-Wide-796M	5	5

Table 10: This table shows the speedup achieved by our LLaMA-NoFT-Wide-796M and self-speculative decoding, respectively.

Strategy	Speedup	
	CNNDM	XSum
Self-Speculative	99%	60%
NoFT-Wide-796M	**131%**	**145%**

5 Conclusion

In this work, we conduct a large-scale experimental study to understand how we can optimize the throughput of speculative decoding. Using our experiments, we outline the various factors that affect speculative decoding throughput. We observe that draft model accuracy on language modeling does not correlate strongly with its performance in speculative decoding. Further, we find that draft model latency is bottlenecked by model depth, and higher model depth increases latency. Based on these two insights, we propose new draft models pruned to align with the target model while trading model depth for width. Our proposed draft model can increase throughput by up to 111% over existing models. We find that the pruned models can be used for supervised fine-tuned target models without modification and our approach generalizes to state-of-the-art models.

Limitations

Our work aims to improve the inference efficiency of LLMs by designing better draft models for speculative decoding. One limitation of our work is that we focus on empirically studying the performance bottleneck and improving lossless speculative decoding throughput, where lossless refers to preserving the target LLM's output distribution. Broader studies on approximate efficient inference algorithms are left for future work.

Acknowledgement

We gratefully acknowledge the support of the NSF Diamond project OAC-2311767 (Democratizing Large Neural Network Model Training for Science). We thank Kangwook Lee for his helpful suggestions during the project. This research used resources from the National Energy Research Scientific Computing Center (NERSC), a Department of Energy Office of Science User Facility, using the NERSC award DDR-ERCAP0029980. This research also used computational resources from the NSF Cloudlab (Duplyakin et al., 2019) facility.

References

Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, et al. 2022. Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15. IEEE.

Sangmin Bae, Jongwoo Ko, Hwanjun Song, and Se-Young Yun. 2023. Fast and robust early-exiting framework for autoregressive language models with synchronized parallel decoding. *arXiv preprint arXiv:2310.05424*.

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

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. 2024. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*.

Jian Chen, Vashisth Tiwari, Ranajoy Sadhukhan, Zhuoming Chen, Jinyuan Shi, Ian En-Hsu Yen, and Beidi Chen. 2024. Magicdec: Breaking the latency-throughput tradeoff for long context generation with speculative decoding. *arXiv preprint arXiv:2408.11049*.

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. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality.

Together Computer. 2023. Redpajama: An open source recipe to reproduce llama training dataset.

Tri Dao, Daniel Haziza, Francisco Massa, and Grigory Sizov. 2023. Flashdecoding. https://pytorch.org/blog/flash-decoding/. Accessed: January 26, 2024.

Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. 2019. The design and operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, pages 1–14.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

John L Hennessy and David A Patterson. 2011. *Computer architecture: a quantitative approach*. Elsevier.

Ke Hong, Guohao Dai, Jiaming Xu, Qiuli Mao, Xiuhong Li, Jun Liu, Kangdi Chen, Hanyu Dong, and Yu Wang. 2023. Flashdecoding++: Faster large language model inference on gpus. *arXiv preprint arXiv:2311.01282*.

HuggingFace. 2023. Open llm leaderboard. https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard. Accessed: January 26, 2024.

Jungo Kasai, Nikolaos Pappas, Hao Peng, James Cross, and Noah A Smith. 2020. Deep encoder, shallow decoder: Reevaluating non-autoregressive machine translation. *arXiv preprint arXiv:2006.10369*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626.

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

Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Ion Stoica, Zhijie Deng, Alvin Cheung, and Hao Zhang. 2023a. Online speculative decoding. *arXiv preprint arXiv:2310.07177*.

Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, et al. 2023b. Deja vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning*, pages 22137–22176. PMLR.

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

Microsoft. 2023. Deepspeed. https://github.com/microsoft/deepspeed. Accessed: January 26, 2024.

Saurav Muralidharan, Sharath Turuvekere Sreenivas, Raviraj Joshi, Marcin Chochowski, Mostofa Patwary, Mohammad Shoeybi, Bryan Catanzaro, Jan Kautz, and Pavlo Molchanov. 2024. Compact language models via pruning and knowledge distillation. *arXiv preprint arXiv:2407.14679*.

Andrea Santilli, Silvio Severino, Emilian Postolache, Valentino Maiorca, Michele Mancusi, Riccardo Marin, and Emanuele Rodolà. 2023. Accelerating transformer inference for translation via parallel decoding. *arXiv preprint arXiv:2305.10427*.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Tran, Yi Tay, and Donald Metzler. 2022. Confident adaptive language modeling. *Advances in Neural Information Processing Systems*, 35:17456–17472.

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

Hanshi Sun, Zhuoming Chen, Xinyu Yang, Yuandong Tian, and Beidi Chen. 2024a. Triforce: Lossless acceleration of long sequence generation with hierarchical speculative decoding. *arXiv preprint arXiv:2404.11912*.

Ziteng Sun, Ananda Theertha Suresh, Jae Hun Ro, Ahmad Beirami, Himanshu Jain, and Felix Yu. 2024b. Spectr: Fast speculative decoding via optimal transport. *Preprint*, arXiv:2310.15141.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023a. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023b. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Xiaohui Wang, Ying Xiong, Yang Wei, Mingxuan Wang, and Lei Li. 2020. Lightseq: A high performance inference library for transformers. *arXiv preprint arXiv:2010.13887*.

Heming Xia, Tao Ge, Peiyi Wang, Si-Qing Chen, Furu Wei, and Zhifang Sui. 2023a. Speculative decoding: Exploiting speculative execution for accelerating seq2seq generation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 3909–3925.

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

Sen Yang, Shujian Huang, Xinyu Dai, and Jiajun Chen. 2024. Multi-candidate speculative decoding. *arXiv preprint arXiv:2401.06706*.

Seongjun Yang, Gibbeum Lee, Jaewoong Cho, Dimitris Papailiopoulos, and Kangwook Lee. 2023. Predictive pipelined decoding: A compute-latency trade-off for exact llm decoding. *arXiv preprint arXiv:2307.05908*.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A distributed serving system for {Transformer-Based} generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 521–538.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. 2023. Draft & verify: Lossless large language model acceleration via self-speculative decoding. *arXiv preprint arXiv:2309.08168*.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *arXiv preprint arXiv:2306.05685*.

Yongchao Zhou, Kaifeng Lyu, Ankit Singh Rawat, Aditya Krishna Menon, Afshin Rostamizadeh, Sanjiv Kumar, Jean-François Kagy, and Rishabh Agarwal. 2024. Distillspec: Improving speculative decoding via knowledge distillation. In *The Twelfth International Conference on Learning Representations*.

A More Experiment Results

Table 11: This table shows the latency of each autoregressive generation step of the draft model.

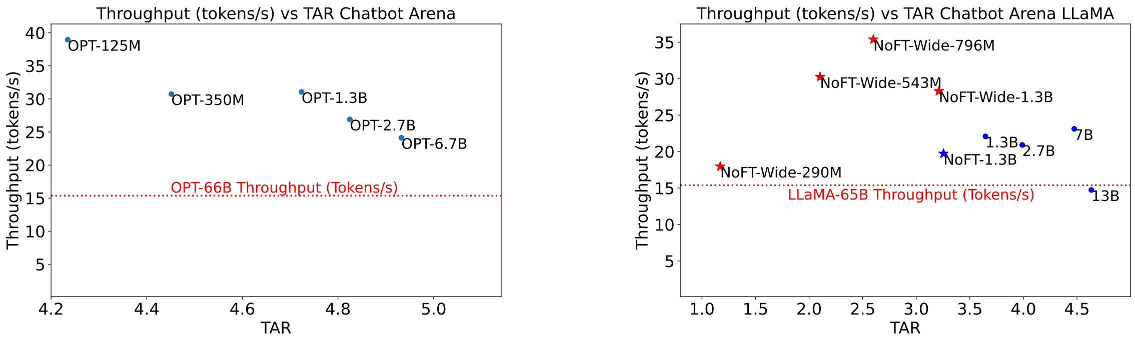
Model	Latency (ms)
OPT-125M	6.23
OPT-350M	11.74
OPT-1.3B	12.64
OPT-2.7B	16.35
OPT-6.7B	18.56

A.1 OPT analysis

In this section, we show more experimental analysis of speculative decoding. In Figure 7, we plot the throughput of OPT and LLaMA models against its TAR on Chatbot Arena. This figure shows that as model size increases, throughput generally decreases due to significantly higher inference latency despite consistent increases in TAR.

In Figure 9, we plot the throughput of OPT and LLaMA models against its TAR on Chatbot Arena. This figure shows that draft latency occupies a large chunk of time in a speculative decoding iteration, opening up new avenues for designing draft models optimal for speculative decoding.

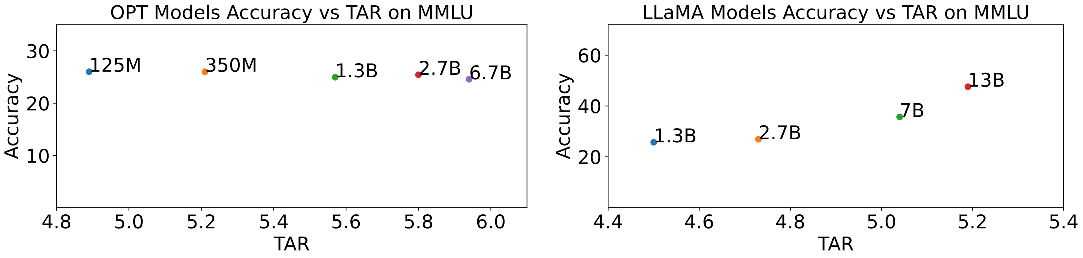
In Figure 8, we plot the task accuracy versus TAR for OPT and LLaMA models on MMLU.



(a) Throughput scaling with increasing TAR in OPT series on Chatbot Arena

(b) Throughput scaling with increasing TAR in LLAMA series on Chatbot Arena

Figure 7: This figure shows the throughput scaling against TAR for Chatbot Arena.



(a) Model accuracy vs TAR for OPT models

(b) Model accuracy vs TAR for LLAMA models

Figure 8: This figure shows the task accuracy versus TAR for OPT and LLAMA models on MMLU. The accuracy numbers are obtained from OpenLLM Leaderboard (HuggingFace, 2023).

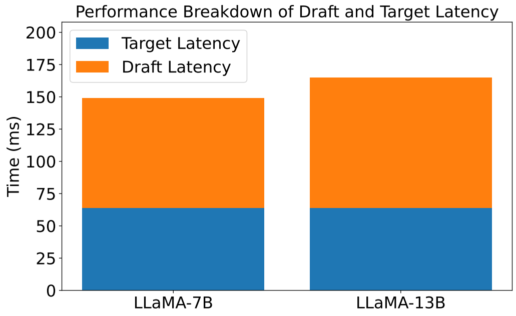


Figure 9: This figure shows the performance breakdown of LLAMA speculative decoding, look ahead length is set to be optimal look ahead length found empirically.

The accuracy numbers are obtained from OpenLLM Leaderboard (HuggingFace, 2023). This figure shows that task accuracy is irrelevant to TAR.

B Discussion

In this section, we discuss how our insights can change if the models or the underlying hardware change.

B.1 Future Draft Model Design

To study how compute and performance changes can lead to different choices of draft models, we use a performance model. The original speculative decoding (Leviathan et al., 2023) model $\frac{1-\alpha^{\gamma+1}}{(1-\alpha)(\gamma c+1)}$ can be simplified to the following to remove the unnecessary assumption of mutual independence between generated tokens in a sequence:

$$\text{Throughput} = \begin{cases} \dfrac{TAR}{(t^d_{target} + t^d_{draft})} & \text{if } TAR > 1, \\ \dfrac{1}{(t^d_{target} + t^d_{draft})} & \text{if } TAR \leq 1. \end{cases}$$

We show that this simplified formula almost perfectly captures the real speculative decoding throughput. Here, t^d represents the latency to generate d tokens autoregressively. In this section, with the aid of the performance model, we provide quantitative answers to several questions: First, we study the improvement in TAR a larger draft model needs to be provided to compensate for the additional inference cost. Next, we study how much improvement in latency is required to change the choice of the draft model.

Table 12: This table shows the latency reduction needed for larger draft models to achieve parity throughput with OPT-125M on MMLU.

Model	Latency (ms)	Parity Latency	Reduction (%)
125M	43.7	43.7	0
350M	79.8	50.6	36.6
1.3B	87.1	58.7	32.6
2.7B	114.3	49.8	56.4
6.7B	139.5	68.2	51.1

Required TAR to match throughput: We use our analytical model to predict the TAR necessary for different models to achieve a target throughput. This can be useful in scenarios where developers deploy speculative decoding-based LLMs and must meet a throughput goal. In Figure 10, we plot the TAR needed by existing models to achieve a specific throughput.

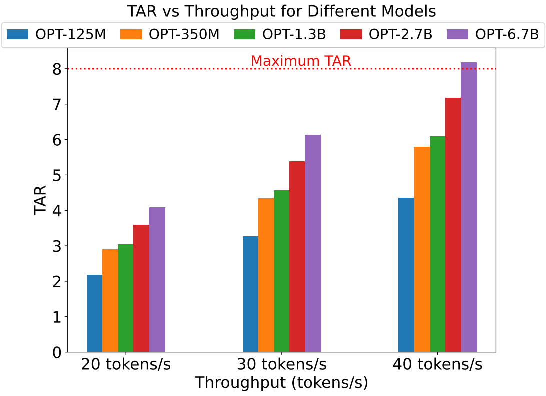


Figure 10: This figure shows the required TAR to achieve a given throughput.

The figure shows that the TAR gap between draft models at each given throughput is much larger than we observed in Figure 2. When the throughput requirement is high, a large draft model, such as OPT-6.7B, can't achieve the desired throughput. This will allow model designers to quickly judge which draft and target model pair allows them to meet throughput requirements.

Improvement in TAR needed to switch to a larger draft model: In Figure 2, we observed that with existing datasets and models, we are better off with the smallest model as the draft model, *e.g.*, OPT-125M, than choosing a larger model. However, there is a possibility that the TAR difference will become greater for new datasets. In Figure 11, we plot the improvement in TAR (extra TAR), which larger models in the OPT model family should provide to match the throughput of the smallest model (OPT-125M) for MMLU. We find that if a 1.3B model can achieve a TAR advantage greater than 2 over OPT-125M for a new workload, we would choose the 1.3B model instead. Furthermore, given that the maximum TAR is capped at 8 in our scenario due to the length of draft token generation, it becomes unfeasible for OPT-2.7B and OPT-6.7B to surpass OPT-125M in performance. This is because the improvement needed in TAR for OPT-6.7B to match the throughput of OPT-125M would exceed this maximum limit.

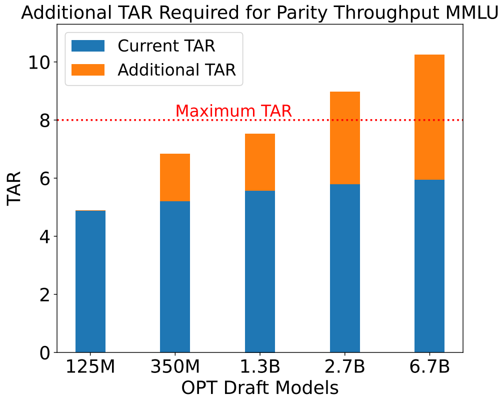


Figure 11: This figure shows the extra TAR needed for each model to achieve parity throughout with OPT-125M on MMLU.

Improvement in latency for switching to higher TAR model: As hardware evolves, latency scaling patterns may change with more computing power and memory bandwidth. Therefore, conclusions drawn on specific hardware (*e.g.*, A100) may not hold for newer or older hardware (*e.g.*, H100 or V100). To account for changing hardware, we study how much draft model latency improvement is needed to achieve throughput parity. To demonstrate this, we first compute the latency reduction needed for different members in OPT family to reach the same throughput as the smallest draft model in Table 12. We find that up to 56% of latency reduction is needed to achieve the same throughput. For instance, for OPT-1.3B to achieve parity throughput with OPT-125M, its latency needs to be reduced by 32.9%. This reinforces our finding that latency reduction provided by the smaller models has significantly more benefit than the extra TAR provided by a larger draft model.

C OPT-350M Configurations

Table 13 shows the detailed model configurations of the OPT-350M variants we created. The goal is to keep the total parameter count close to that of OPT-350M while adjusting model width and depth.

Table 13: This table shows the model configuration of various OPT-350M models we created. The goal is to explore the tradeoff between model depth and width while keeping the total parameter count constant.

Num Layers	Attn. Heads	Hidden size	FFN Dim
24	16	1024	4096
20	20	1280	3448
16	22	1408	4096
12	28	1792	3448
8	36	2304	3448
4	56	3584	3448

D Simplifying Analytical model

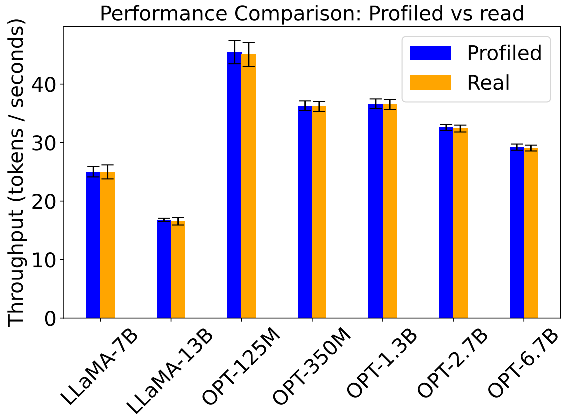


Figure 12: This figure shows that our performance model correctly captures the real performance of speculative decoding. We use LLaMA-65B and OPT-66B as the target model for each model family, respectively.

The original speculative decoding paper (Leviathan et al., 2023) proposed an analytical model $\frac{1-\alpha^{\gamma+1}}{(1-\alpha)(\gamma c+1)}$ to describe the speedup achieved by speculative decoding, where α denotes the expected token acceptance rate (in percentage) and γ denotes the look ahead length. However, this model is inaccurate since it assumes that the tokens generated in a sentence are mutually independent. We simplify this cost model and use our updated analytical model in our experiments.

Assuming a setup similar to prior work (Leviathan et al., 2023; Chen et al., 2023; Miao et al., 2023) where speculative execution of the draft model and target model verification phases happen sequentially, the performance of speculative decoding can be decomposed into the following factors,

$$\text{Throughput} = \begin{cases} \dfrac{TAR}{(t^d_{target} + t^d_{draft})} & \text{if } TAR > 1, \\ \dfrac{1}{(t^d_{target} + t^d_{draft})} & \text{if } TAR \leq 1. \end{cases}$$

Considering a case where, in each iteration, d tokens are generated by the draft model, t^d_{draft} depicts the time draft models take to generate d draft tokens, while t^d_{target} is the time taken by the target model for verifying those d draft tokens. TAR is used to denote the average number of tokens that were matched across a query or a dataset.

Verifying analytical model: In Figure 12, we compare the throughput predicted by our model with throughput measured on real hardware for two model families: LLaMA (7B and 13B) and OPT (125M, 350M, 1.3B, 2.7B, and 6.7B) to serve LLaMA-65B and OPT-66B on MMLU.

We run these experiments on 4 Nvidia 80GB A100 GPUs for 100 iterations on the real server, and the error bars in Figure 12 represent the standard deviation of the measurement. For the performance model, we collect t^d_{draft} and t^d_{target} on a real cluster with a single iteration. For TAR, we collect the average token acceptance rate from the MMLU dataset. The maximum deviation we observed between our proposed analytical model and the results obtained is 3.5%. The close correspondence between our performance model and real measurements shows that our performance model accurately predicts the throughput of speculative decoding.