How Speculative Can Speculative Decoding Be?

Zhuorui Liu, Chen Zhang, Dawei Song[†]
Beijing Institute of Technology
{zrliu,czhang,dwsong}@bit.edu.cn

Abstract

Large language models (LLMs) have drawn great attention from the field of natural language processing and beyond, due to their impressive capability of autoregressive modeling, yet bringing an obvious problem, i.e., the largely increased latency. An emerging idea to alleviate this problem is speculative decoding, which first uses a draft model to draft tokens autoregressively and then makes the target model verify these tokens in parallel. The draft model is typically smaller than the target model, and it essentially trades generation quality for speed. Thereby, speculative decoding can be viewed as a speculative game for the target model in term of verification failures. That is, the lengthy draft tokens proposed by the small draft models could fail in the verification stage. Naturally, a critical question arises: how speculative can speculative decoding be, or in other words, how small can an adequate draft model be and how large can an appropriate number of draft tokens be? This work aims to investigate these questions and demonstrate how the scale of the draft model and the number of draft tokens would have an impact on the overall latency of the speculative decoding. We theoretically show that neither of above two factors will be infinitely speculative. Namely, there is a certain turning point for each of them. We then empirically show that the scale of the draft model could be 10-20$\times$ smaller than the target model and the optimal number of draft tokens should lie in 3-5.

Keywords: Speculative decoding, Draft model, Draft tokens

1. Introduction

In recent years, large language models (LLMs) such as PaLM (Chowdhery et al., 2022), PaLM-v2 (Anil et al., 2023), GPT-3 (Brown et al., 2020) and ChatGPT (OpenAI, 2023b), have garnered a significantly increasing attention. Notably, those constructed using transformer architectures (Vaswani et al., 2017) and scaling up to billions of parameters have showcased exemplary performance across diverse tasks. This momentum has reverberated beyond the realm of NLP, influencing areas like computer vision (e.g., Zou et al., 2023; Kirillov et al., 2023). Exemplifying this progression, GPT-4 (OpenAI, 2023a), an LLM with trillions of parameters, has captured the imagination of many. GPT-4's striking capabilities have highlighted the potential of LLMs, prompting individuals from various disciplines to view this advancement as the dawn of a new productivity paradigm.

While LLMs offer undeniable advantages, they are facing a key challenge due to their immense sizes. Such scales can impose substantial hardware requirements and elevate inference latency (Xiao et al., 2023; Hu et al., 2021), potentially compromising the user experience, especially for large batches or time-sensitive tasks. To mitigate this challenge, various strategies have been proposed to expedite LLM inference and curtail latency (Guo et al., 2023; Yang et al., 2023). Correspondingly, a range of approaches have emerged (Zhang et al., 2024; Dao, 2023). A particularly promising approach is speculative decoding, where a leaner draft model aids the generation process (Chen et al., 2023). This draft model produces a sequence of preliminary tokens (called draft tokens), which are subsequently sent to the LLM (the target model) for verification in a single pass.

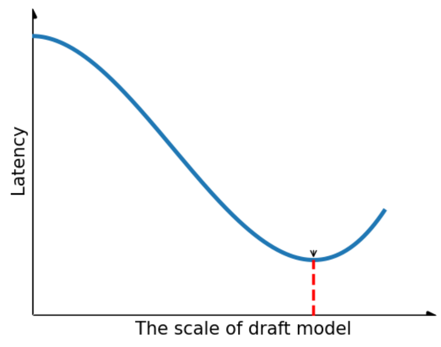
While this strategy has shown promise in addressing the inference latency problem and enhancing user experience with more efficient interactions, it has also surfaced from a new set of challenges. Specifically, the degree to which the latency is mitigated depends on the scale of the chosen draft model and the number of draft tokens selected.

This observation raises a critical question: How speculative can speculative decoding truly be? Specifically, what would be the minimum viable scale for the draft model, and what is the upper limit for the number of draft tokens? We are the first to dive into this important question. This is essential for making a good use of the typically limited resource for deployment while achieving an ideal result.

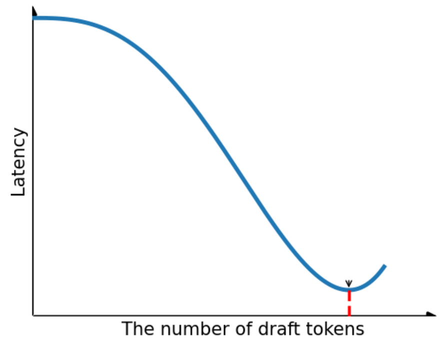
We illustrate this intricacy in Figure 1. Our hypothesis posits that initially, as the draft model assists, the inference latency should decrease. As the scale of the draft model diminishes, this latency reduction should continue up to a specific model size. Beyond this point, the limited capacity of a smaller draft model may hamper its ability to produce meaningful tokens, leading to an increase in latency. We substantiate this concept using mathematical for-

[†] Corresponding author. Also with the Open University, UK.

[‡] The code is available at https://github.com/ZhuoruiLiu12/SpecGame



(a) The impact of the scale of draft model.



(b) The impact of the number of draft tokens.

Figure 1: An optimal scale for the draft model and an optimal number of draft tokens exist. In (a), as the horizontal axis increases, the scale of the draft model decreases. This leads to an initial decrease in latency, followed by an increase, attributed to the reduced capacity of the draft model. In (b), given a specific draft model and target model, the number of draft tokens significantly influences latency. Both an excess and a deficiency of tokens can adversely affect the speedup.

mulations in subsequent sections. Similarly, for a given draft model, the inference latency might first decrease with an increasing number of draft tokens but then escalate. This can occur because an excessive number of tokens produced by the draft model might not be optimally accepted by the target model, resulting in increased latency.

A more rigorous validation of these hypotheses will be presented in Section 3.

We carried out a range of experiments, varying the size of the draft model and the number of tokens under standard conditions. Remarkably, our findings reveal that in extreme circumstances, the model can be scaled down far more than we initially thought, while still achieving substantial speedup. Moreover, the draft model's capacity plays a pivotal role in influencing latency. We discern that the optimal number of draft tokens lies the range between 3 to 5, and that the smaller draft models tend to outperform the larger ones, with the possibility of scaling down by up to 60 times in extreme situations.

The main contributions of this paper include:

- Pioneering research on key factors influencing the inference latency of speculative decoding.
- Comprehensive experiments exploring the boundaries of speculative decoding, targeting the aforementioned factors. Subsequent analysis yields insights into the optimal range of draft tokens and the most effective scale of the draft model.

2. Background

A wealth of research has focused on the pivotal challenge of reducing LLMs' inference latency. Often, studies juxtapose compression and acceleration technologies, given their shared objective of boosting model efficiency. This alignment is evident in the comprehensive reviews by Choudhary et al. (2020) and Xu and McAuley (2023).

Inference latency is fundamentally intertwined with memory size. As such, strategies aimed at compressing memory size inherently contribute to reducing this latency. For example, pruning techniques (e.g., LeCun et al., 1989; Lin et al., 2020; Guo et al., 2020) endeavor to discard less crucial weights. Quantization targets the reduction of weight precision, while distillation approaches (e.g., Hinton et al., 2015; Jiao et al., 2019; Sanh et al., 2019; Zhang et al., 2023b; Yang et al., 2022; Zhang et al., 2023d,c,a) seek broader performance enhancements. These techniques have yielded significant speed improvements, often with negligible or zero compromise in performance.

Conversely, some strategies don't inherently modify memory size. Early exiting (EE) methodologies, exemplified by FastBERT (Liu et al., 2020), SkipBERT (Wang et al., 2022), and BERxiT (Xin et al., 2021), employ criteria to halt inference at intermediary layers, facilitating quicker results. Another innovative method, token skipping (e.g., Ye et al., 2021; Guan et al., 2022), bypasses select tokens across layers based on their significance.

One of the most efficacious paradigms for acceleration embraces a trade-off between space and time. As previously alluded to, leveraging a draft model in tandem with the primary model for decoding emerges as an optimal choice. This methodology, termed speculative decoding, has multiple incarnations in the research ecosystem. Noteworthy implementations include Google's speculative decoding (Leviathan et al., 2023), DeepMind's speculative sampling (or SpS) (Chen et al., 2023), and

HuggingFace's assisted generation (or AsG) (Joao Gante, 2023), among others.

The speculative decoding approach employs autoregressive models that consist of both the target model and a draft model. In this context, when we refer to autoregressive models, we mean models that generate sequences by regressing on their own previous outputs. Typically, the straightforward application of the target model to generate text directly serves as our baseline method, against which we compare the speedup achieved using different speculative strategies.

Let's consider a sequence of tokens $x_1, ..., x_n$ as our input. The subsequent K tokens, denoted by $\tilde{x}_{n+1}, ..., \tilde{x}_{n+k}$ are either sampled or generated from the draft model, acting as candidate tokens. Based on this setup, we delve into various speculative decoding algorithms.

For the autoregressive decoding method, abbreviated as ArD, our attention is squarely on the input sequences. These autoregressive models generate the x_{n+1} token directly through the following relationship:

$$x_{n+1} \sim q(x|x_1, ..., x_n) \tag{1}$$

Here, $q(\cdot|\cdot)$ represents the distribution of the next token x derived from the target model. ArD continuously generates tokens in this manner, one at a time, until it reaches the specified maximum token length.

The Speculative Sampling (SpS) algorithm operates in the following manner:

Token Acceptance Criteria SpS determines the acceptance of the next candidate token $\tilde{x}_{n+1}$ based on:

$$r < \min(1, \frac{q(\tilde{x}_{n+1}|x_1, ..., x_n)}{p(\tilde{x}_{n+1}|x_1, ..., x_n)}) \tag{2}$$

Here:

- $p(\cdot|\cdot)$ denotes the probability of the candidate token $\tilde{x}_{n+1}$ as predicted by the draft model.
- r is a random number drawn from the uniform distribution $U[0, 1]$.

Token Rejection and Sampling If the condition in Equation (2) is not satisfied, the token $\tilde{x}_{n+1}$ is rejected. The next token is then sampled based on the distribution:

$$(q(\tilde{x}_{n+1}|x_1, ..., x_n) - p(\tilde{x}_{n+1}|x_1, ..., x_n))_+ \tag{3}$$

The function $(\cdot)_+$ is defined as:

$$(f(x))_+ = \frac{\max(0, f(x))}{\sum_x \max(0, f(x))} \tag{4}$$

Upon sampling, SpS exits this iterative loop.

Additional Token Sampling If all tokens in the sequence are accepted, SpS samples an extra token x_{n+K+1} from the distribution:

$$x_{n+K+1} \sim q(x|x_1, ..., x_n, x_{n+1}, ..., x_{n+K}) \tag{5}$$

Notably, in this scenario, SpS can provide a maximum of $K + 1$ tokens per iteration. This is more efficient than traditional methods, making it advantageous in certain applications.

AsG method is grounded in a greedy decoding approach. Unlike sampling methods, it deterministically chooses the highest probability token at each step. This deterministic approach is often faster and can be crucial for certain applications. Instead of sampling, AsG selects the token with the highest conditional probability, as described:

$$\tilde{x}_{n+1} = \text{argmax}\ p(\tilde{x}_{n+1}|x_1, ..., x_n) \tag{6}$$

Once the candidate tokens $\tilde{x}_{n+1}, ..., \tilde{x}_{n+k}$ are generated, AsG performs critical steps:

Token Acceptance Criteria AsG forwards the candidate tokens to the target model, which subsequently produces a validation token sequence. AsG then decides whether to accept the token $\tilde{x}_{n+1}$ from the candidate token sequence based on a strict match with the corresponding token in the validation sequence. It is evident that if the token $\tilde{x}_{n+1}$ aligns with the token x_{n+1}, AsG will accept the token $\tilde{x}_{n+1}$. This matching criterion is represented by:

$$Acceptance = C(\tilde{x}_{n+i}, x_{n+i}) \tag{7}$$

In this context, the function $C(\cdot, \cdot)$ is delineated as:

$$C(f(x), g(x)) = \begin{cases} 0, & \text{if } f(x) \neq g(x) \\ 1, & \text{if } f(x) = g(x) \end{cases} \tag{8}$$

Provided that Equation (7) is met, AsG continues this evaluation for the subsequent token $\tilde{x}_{n+i+1}$ until either Equation (7) is no longer fulfilled or the token $\tilde{x}_{n+i}$ is the concluding one in the sequence.

Token Rejection and Sampling If the condition in Equation (7) is unmet, the token $\tilde{x}_{n+i}$ is discarded. At this juncture, AsG terminates the discrimination loop, retaining all accepted tokens while discarding the remaining candidate tokens from this position onward. Additionally, AsG accepts the token situated immediately after the last accepted position in the validation sequence, as generated by the target model.

Additional Token Sampling Should all candidate tokens be accepted, AsG, akin to SpS, will recognize the token positioned at the end of the validation sequence, as produced by the target model.

This additional acceptance serves to bolster the efficiency of decoding. In summary, given a consecutive matching token count of N, AsG invariably acknowledges $N+1$ tokens within the validation sequence, thereby offsetting the stringent matching criterion.

For ease of understanding, Figure 2 shows the process of ArD algorithm. And both of SpS and AsG share the same idea which is illustrated in Figure 3. They follow a similar process: individuals utilize LLMs to complete their tasks. Upon text generation by the LLMs, users review the output to ensure it meets their criteria. If the generated text is satisfactory, it is accepted; otherwise, users discard it and opt to write on their own. Specifically, as for SpS and AsG, they could maintain the same quality of output as output of LLMs because of the internal principle, for end-to-end tasks, these methods can achieve the unanimous metrics as LLMs.

3. Research Questions

In this section, we delve into a detailed examination of the questions raised in previous sections.

To initiate our discussion, let's consider a specific scenario where the number of draft tokens, denoted as K, is 1. Our focus is the latency associated with generating each token. Let's hypothesize that the probability of the target model accepting a token is p. Furthermore, let a and b represent the latencies of the draft and target models in generating and verifying draft tokens, respectively. We can define the expected latency per token, E_{pt}, for SpS as follows:

$$\begin{aligned} E_{pt} &= \frac{p(a+b)}{2} + (1-p)(a+b) \\ &= (1-\frac{p}{2})(a+b) \end{aligned} \tag{9}$$

In Equation (9), p represents the probability of each token being accepted. This probability, p, is influenced by the scale of the draft model, with K being a contributing factor. Moreover, different draft models can also affect p. Specifically, we can express it as:

$$p(x|\text{scale}, K) \tag{10}$$

where x is the event of token acceptance. Within the same family of models, it's evident that the size of a model correlates positively with its computational capacity empirically. Likewise, an increase in model capacity leads to a proportional increase in p. This relationship can be represented as:

$$p \propto \text{capacity} \propto \text{scale} \tag{11}$$

Equation (9) depicts the maximum latency of SpS in a specific scenario where $K=1$ and a given draft model is used. By altering the draft model from the same family, we can adjust p to achieve an ideal latency. The pressing question then becomes: how can one minimize this latency? There may exist an optimal scale for the draft model that minimizes E_{pt}.

From the preceding analysis, let's extend our consideration to a general scenario where $K=k$. The expected latency per token, E_{pt}, which is divided into two situations, can be expressed as:

$$E_{pt} = (1-p)(a+b) + L(1) \tag{12}$$

Here, L presents a recursive decomposition for the overall latency, which is defined by:

$$L(i) = \begin{cases} p[(1-p)\frac{a+b}{i+1} + L(i+1)] & ,i \neq k \\ p\frac{a+b}{i+1} & ,i = k \end{cases} \tag{13}$$

Upon scrutinizing Equation (12) and Equation (13), it becomes evident that there might be an optimal scale for the draft model that minimizes E_{pt}, which can be referred to as the smallest size of the draft model.

In parallel, there could be a single value or a range for K that, in conjunction with a specific family of draft models, minimizes E_{pt}. Building upon this analysis, our aim is to identify the optimal K and the ideal scale of the draft model.

4. Experiments

4.1. Experiments Setup

We employ the test dataset from LAMBADA (Paperno et al., 2016) as prompts. The LAMBADA dataset comprises the full text of 2,662 novels for training, 4,869 passages for validation, and 5,153 passages for testing. The draft model generates subsequent text based on these prompts, which is then verified by the LLM to determine the acceptance of the tokens. Throughout this process, we measure two primary durations: the average generation time based on the number of prompts, which evaluates the acceleration effect, and the average generation time per token.

To conduct the experiments, several hyperparameters must be established, including sample size, the number of tokens generated in each iteration, and the number of tokens generated with the assistance of the small model. In our experiments, we execute the entire process 500 times to compute the average time. We set the maximum length of generated tokens at 128. We also vary the number of tokens generated with assistance, incrementing from 1 to 7, to discern how large the number of draft tokens could be. The batch size is set to 1 for these experiments.

For our model series, the selection criteria for the model family were twofold: firstly, the family should

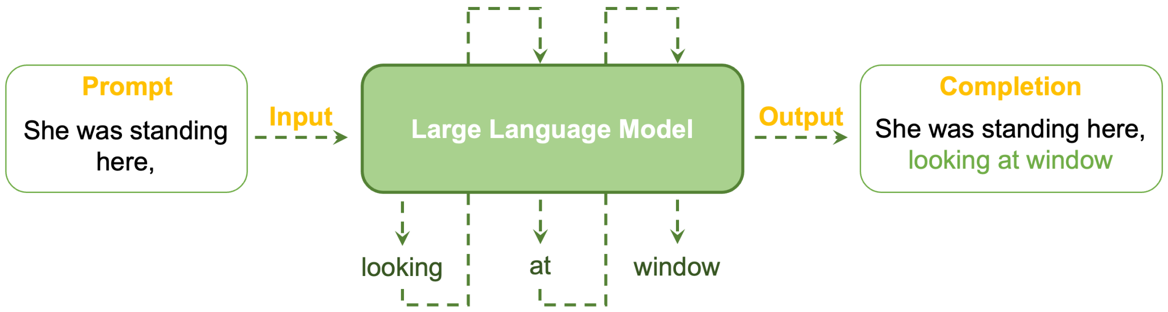


Figure 2: Autoregressive LLM drives the text generation process. Based on the given prompt, ArD initially produces the token “looking”. Subsequently, using the prompt and this first token, it autoregressively generates the next token, "at". This process is repeated until the end.

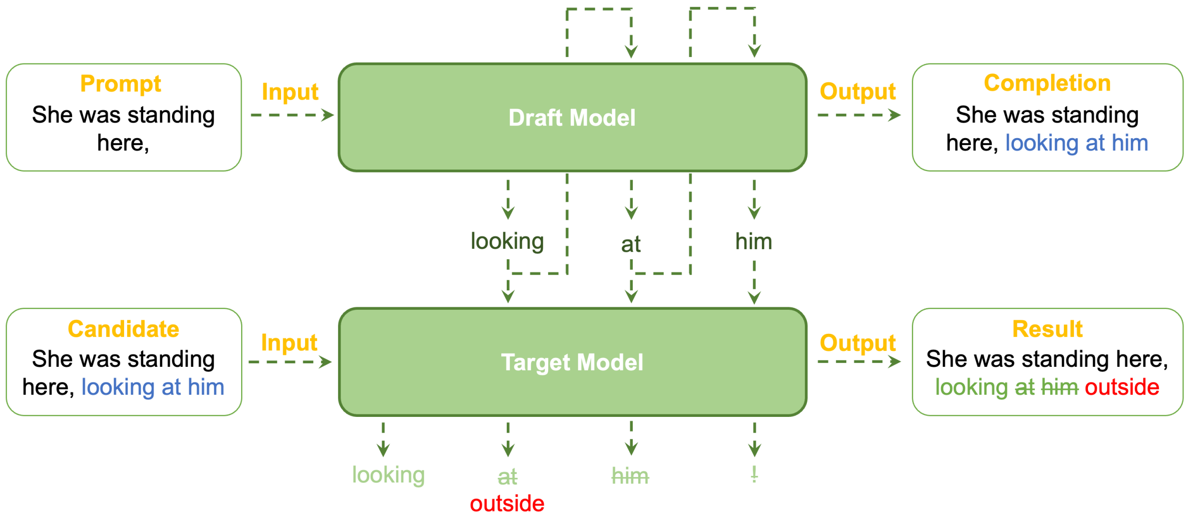
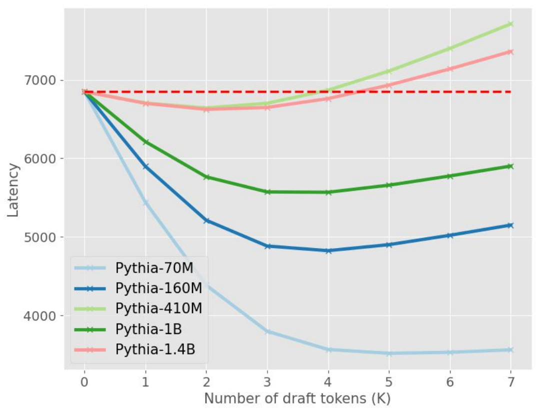
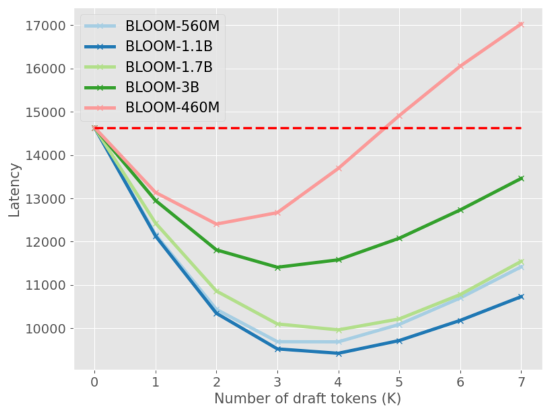


Figure 3: The SpS optimizes the text generation process. Initially, as depicted in Figure 2, the draft model produces a sequence of tokens. These tokens are subsequently concatenated and introduced to the target model for validation. Based on this validation, the target model either accepts or discards the tokens. For instance, in the figure, the draft model generates the phrase “looking at him”. This concatenated phrase is then passed to the target model. The target model accepts the initial token but discards the succeeding two, choosing instead to generate the word “outside” as its next token and subsequently terminating the validation.

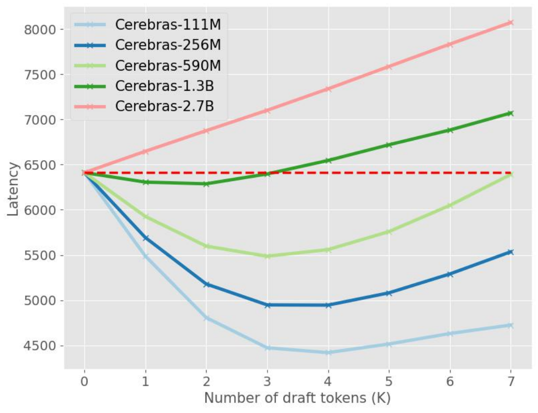
contain as many draft models as possible; secondly, there should be a noticeable size difference between the draft models and the target model. Different model families can not be mixed, because the vocabulary must be the same, and different training data may also result in different distribution between draft model and target model, decreasing the performance. We utilized models such as Pythia (Biderman et al., 2023), Cerebras-GPT (Dey et al., 2023), LLaMA (Touvron et al., 2023), BLOOM (Scao et al., 2022), and GPT2 (Radford et al., 2019) to investigate the speculative capacity of the draft model and to analyze the outcomes. Additionally, to experiment with various scales of draft models, we incorporated several smaller-scale models from different model families, including TinyLLaMA (Zhang et al., 2023e), DistilGPT2 (Sanh et al., 2019), and OpenLLaMA (Geng and Liu, 2023). Furthermore, in pursuit of the same objective, we also tested pruned models as draft models in some experiments, utilizing the LLM-pruner (Ma et al., 2023) as our pruning algorithm and attaching some pruned draft model experiments in SpS. It's worth noting that due to pruning algorithm reason, we don't use KV-cache mechanism in some SpS experiments, such as BLOOM and LLaMA, leading to corresponding outputs with AsG used KV-cache often surpass those achieved with SpS in terms of speed and get lower latency, and also getting relatively higher speed up.



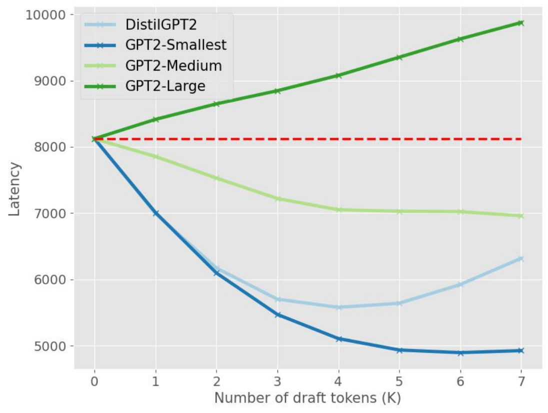
(a) Pythia-2.8B



(b) BLOOM-7.1B



(c) Cerebras-GPT-6.7B



(d) GPT2-XL

Figure 4: The average decoding latency across different model families utilizing the SpS decoding strategy is presented. The horizontal axis indicates the number of tokens decoded by the smaller model in a single instance. Notably, when $k = 0$, only the original base model is used to sample tokens without any assistance. The latency in the Figures of all experimental results in this paper are measured in milliseconds. We used smoothed data to plot this and next figures.

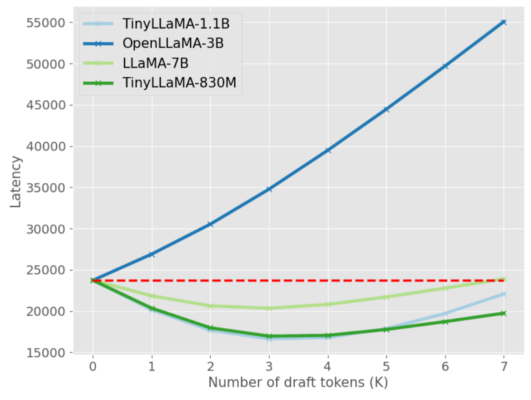
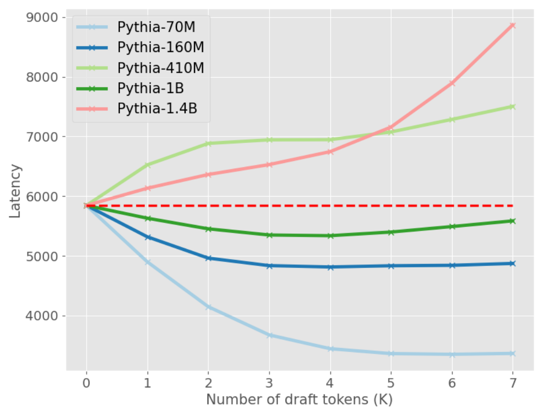


Figure 5: LLaMA-13B SpS.

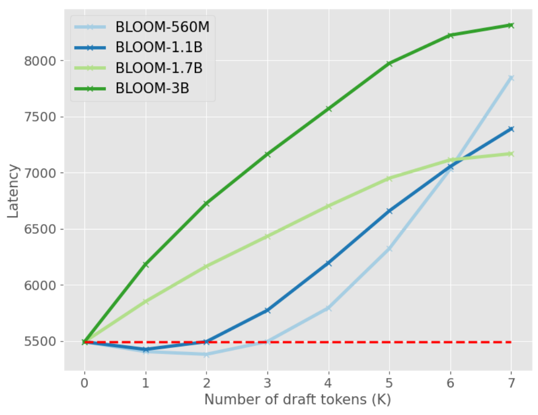
4.2. Main Results

As previously discussed, we have chosen several model families for our study.

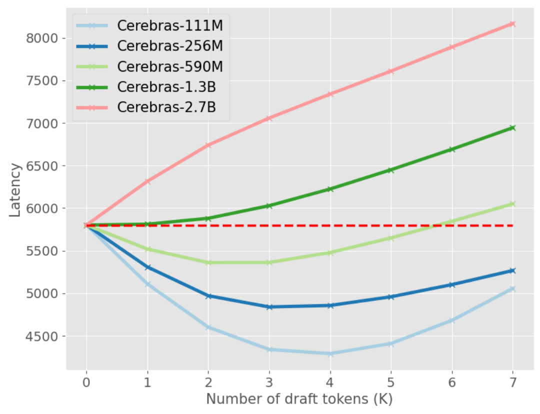
Representative Results. The results from the SpS experiments are depicted in Figure 4 and Figure 5. Each subfigure within this figure represents a distinct target model. In contrast, the AsG experiments results are shown in Figure 6 and Figure 7, serving as a control group to mitigate the effects of the algorithm. As for LLaMA series of experiments, its target model has the largest number of parameters in the whole experiments which is beyond the size in others(1B to 7B), and the draft models used are fewer compared to others, many of them are not released by Meta, resulting in dif-



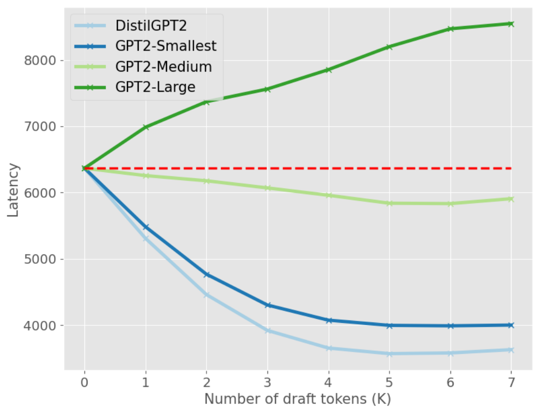
(a) Pythia-2.8B



(b) BLOOM-7.1B



(c) Cerebras-GPT-6.7B



(d) GPT2-XL

Figure 6: AsG.

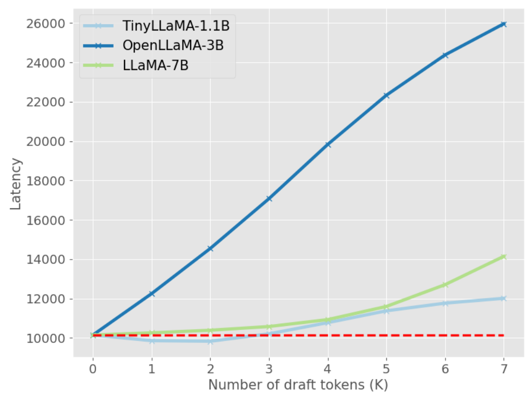


Figure 7: LLaMA-13B AsG.

ferent trend, so we list this set of results separately for both highlighting the importance of draft model and further analysis in later of this Section. The optimal speedups and associated metrics for each target model experiment are provided in Table 1.

Back to Initial Question. From Figure 4 and Figure 5, it's evident that nearly every combination of the draft model and K results in a speedup. However, the magnitude of acceleration varies. This leads us back to our initial queries: What is the maximum number of draft tokens (K) that can be used, and how minimal can the draft model be to still be effective for speculative decoding?

The Optimal Range of Draft Token. Observing the trends in Figure 4 and Figure 5, we note that as K increases, the inference latency first decreases and then rises. In most scenarios, an ideal speedup is achieved when K ranges between 3 to 5, and it is independent from the draft model family we select, it's a general conclusion. A token count for the draft that is either too large or too small tends to diminish the acceleration level, sometimes even resulting in

Target Model	Sampling Method	Draft Model	K	Speed Up	Scaling
Pythia-2.8B	Autoregressive	None	0	1×	1×
	SpS	Pythia-70M	5	1.984×	40×
	AsG	Pythia-70M	5	1.766×	40×
BLOOM-7.1B	Autoregressive	None	0	1×	1×
	SpS	BLOOM-1.1B	4	1.583×	6.45×
	AsG	BLOOM-560M	2	1.053×	12.68×
Cerebras-GPT-6.7B	Autoregressive	None	0	1×	1×
	SpS	Cerebras-GPT-111M	4	1.507×	60.36×
	AsG	Cerebras-GPT-111M	4	1.387×	60.36×
GPT2-XL	Autoregressive	None	0	1×	1×
	SpS	GPT2-Smallest	5	1.695×	12.097×
	AsG	DistilGPT2	5	1.827×	18.29×
LLaMA-13B	Autoregressive	None	0	1×	1×
	SpS	TinyLLaMA-1.1B	3	1.506×	11.82×
	AsG	TinyLLaMA-1.1B	2	1.096×	11.82×

Table 1: We showcase the optimal acceleration levels attained. The results for each target model are presented in three distinct rows. Each row details the draft model used, the value of K, the speedup ratio, and the model scaling ratio. These parameters are crucial in achieving the best speedup effect across different sampling methods. Within the table, "Autoregressive" refers to the baseline method that target model is responsible for generation. Consequently, in this row, there's no accompanying draft model, and the value of K is designated as 0. This table we show the exact data rather than smoothed.

a latency slower than when solely using the target model. Furthermore, Table 1 confirms this optimal K range across various sampling methods. While there are instances where the optimal K lies outside this range, the difference in speedup between the K values within this range and the optimal K from Figure 7 is negligible.

How Small The Draft Model Can Be? To determine the minimum effective size of the draft model, we conducted several experiments. As observed in Table 1, the draft model can be scaled down to between 6 and 18 times smaller than the target model while still maintaining a high acceleration level within the optimal K range. In more extreme cases, as depicted in Figure 4a and Figure 4c, the draft model can be reduced to 40 and 60 times smaller than the target model, respectively, and still achieve comparable or even superior speedups. Intuitively, one might hypothesize that a larger-scale model, while more powerful, would also be slower in inference. Furthermore, as the size gap between the draft model and the target model widens, latency should initially decrease and then increase. However, the data from Figure 4 suggests the opposite: the smaller the draft model, the better the speedup achieved. This leads us to believe that the draft model can be significantly reduced in size and still deliver superior performance. The scales of the draft models we've examined so far might not be small enough to observe the point at which the latency curve begins to rise. In simpler terms, we haven't yet identified the optimal scale for draft model acceleration.

Compression Techniques Can't Be Applied Directly. Relying solely on methods like pruning to generate a series of smaller draft models is not sufficient to determine the optimal scale. As illustrated in Figure 4b and Figure 5, pruned models exhibit varied performance, often deviating from the general trends observed in our experiments. Specifically, the pruned BLOOM model underperforms in terms of speedup, whereas the pruned TinyLLaMA model demonstrates superior performance. In Figure 5, the performance of the OpenLLaMA draft model (Geng and Liu, 2023) appears to be an outlier, leading us to hypothesize that its capacity might be inadequate for generating consistent experimental results. Meanwhile, the distribution of draft model should align with target model for normal result (Zhou et al., 2023). Simply pruning a model doesn't guarantee that the resulting draft model will perform effectively, and as such, may not yield valid experimental findings. The outcomes from these pruned models may not provide a decisive conclusion regarding the optimal scale for the draft model due to their post-pruning capacities. To thoroughly investigate the ideal size of the draft model, alternative strategies, such as training models from scratch or employing knowledge distillation (Hinton et al., 2015), should be explored.

5. Conclusions

In this paper, we delve into the extent of speculation in speculative decoding. Specifically, we explore the maximum appropriate number of draft tokens (K) and the minimum effective size of the draft model, and find that this K is applicable to different draft models. To neutralize the specific effects of SpS, we employ two sampling algorithms and conduct experiments using five distinct model families. Our experiments yield an ideal range for K and lead to a counterintuitive finding: smaller draft models often offer superior speedup levels. Remarkably, a draft model with a size gap of 60 times smaller than the target model can achieve the optimal performance in some special cases. We believe that after some processing (e.g. constructing a special draft model like Zhou et al., 2023), this finding, namely around 60 times smaller draft model could get an ideal performance, is applicable to more general scenarios. For larger scale models, although further experimentation is yet to be done, the blog of Hugging Face (Joao Gante, 2023) has also suggested an increase of size gap. This is indeed consistent with our findings. We hope that our findings will inform a series of follow-on explorations, e.g., pursuing methods to generate smaller draft models, pinpoint the optimal draft model scale and uncover underlying patterns.

Limitations

This work is focused on addressing the limitations of speculative decoding. Due to the constraints of computation resources, we solely investigated the model scope of 3-13B. For extrapolation to larger model scale, we need to perform more practical experiments. Furthermore, we used some pruned models as well as some existing draft models to detect the acceleration effect. However, some of these models are of a poor generation capability, leading to a poor speculative decoding performance which may not be related to the model scale. Therefore, how to construct a useful and smaller draft model to align with the target model for speculative decoding is still an open research question worth further exploration. Solving the problem would lay a solid basis for us to more accurately detect the scale bounds.

Acknowledgements

This work is funded in part by the Natural Science Foundation of China (grant no: 62376027) and Beijing Municipal Natural Science Foundation (grant no: 4222036 and IS23061).

Bibliographical References

Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. 2023. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. 2023. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pages 2397–2430. PMLR.

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.

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

Tejalal Choudhary, Vipul Mishra, Anurag Goswami, and Jagannathan Sarangapani. 2020. A comprehensive survey on model compression and acceleration. *Artificial Intelligence Review*, 53:5113–5155.

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

Tri Dao. 2023. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*.

Nolan Dey, Gurpreet Gosal, Hemant Khachane, William Marshall, Ribhu Pathria, Marvin Tom, Joel Hestness, et al. 2023. Cerebras-gpt: Open compute-optimal language models trained on the cerebras wafer-scale cluster. *arXiv preprint arXiv:2304.03208*.

Xinyang Geng and Hao Liu. 2023. Openllama: An open reproduction of llama.

Yue Guan, Zhengyi Li, Jingwen Leng, Zhouhan Lin, and Minyi Guo. 2022. Transkimmer: Transformer learns to layer-wise skim. *arXiv preprint arXiv:2205.07324*.

Cong Guo, Jiaming Tang, Weiming Hu, Jingwen Leng, Chen Zhang, Fan Yang, Yunxin Liu, Minyi Guo, and Yuhao Zhu. 2023. Olive: Accelerating large language models via hardware-friendly outlier-victim pair quantization. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, pages 1–15.

Demi Guo, Alexander M Rush, and Yoon Kim. 2020. Parameter-efficient transfer learning with diff pruning. *arXiv preprint arXiv:2012.07463*.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2019. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*.

Joao Gante. 2023. Assisted generation: a new direction toward low-latency text generation.

Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. 2023. Segment anything. *arXiv preprint arXiv:2304.02643*.

Yann LeCun, John Denker, and Sara Solla. 1989. Optimal brain damage. *Advances in neural information processing systems*, 2.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 19274–19286. PMLR.

Zi Lin, Jeremiah Zhe Liu, Zi Yang, Nan Hua, and Dan Roth. 2020. Pruning redundant mappings in transformer models via spectral-normalized identity prior. *arXiv preprint arXiv:2010.01791*.

Weijie Liu, Peng Zhou, Zhe Zhao, Zhiruo Wang, Haotang Deng, and Qi Ju. 2020. Fastbert: a self-distilling bert with adaptive inference time. *arXiv preprint arXiv:2004.02178*.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. Llm-pruner: On the structural pruning of large language models. *arXiv preprint arXiv:2305.11627*.

OpenAI. 2023a. Gpt-4 technical report.

OpenAI. 2023b. Introducing chatgpt. `https://openai.com/blog/chatgpt`.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. 2016. The lambada dataset: Word prediction requiring a broad discourse context. *arXiv preprint arXiv:1606.06031*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. 2022. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Jue Wang, Ke Chen, Gang Chen, Lidan Shou, and Julian McAuley. 2022. Skipbert: Efficient inference with shallow layer skipping. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7287–7301.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. 2020. DeeBERT: Dynamic early exiting for accelerating BERT inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages

2246–2251, Online. Association for Computational Linguistics.

Ji Xin, Raphael Tang, Yaoliang Yu, and Jimmy Lin. 2021. Berxit: Early exiting for bert with better fine-tuning and extension to regression. In *Proceedings of the 16th conference of the European chapter of the association for computational linguistics: Main Volume*, pages 91–104.

Canwen Xu and Julian McAuley. 2023. A survey on model compression and acceleration for pretrained language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 10566–10575.

Nan Yang, Tao Ge, Liang Wang, Binxing Jiao, Daxin Jiang, Linjun Yang, Rangan Majumder, and Furu Wei. 2023. Inference with reference: Lossless acceleration of large language models. *arXiv preprint arXiv:2304.04487*.

Yi Yang, Chen Zhang, and Dawei Song. 2022. Sparse teachers can be dense with knowledge. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 3904–3915, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Deming Ye, Yankai Lin, Yufei Huang, and Maosong Sun. 2021. Tr-bert: Dynamic token reduction for accelerating bert inference. *arXiv preprint arXiv:2105.11618*.

Chen Zhang, Dawei Song, Zheyu Ye, and Yan Gao. 2023a. Towards the law of capacity gap in distilling language models.

Chen Zhang, Yang Yang, Jiahao Liu, Jingang Wang, Yunsen Xian, Benyou Wang, and Dawei Song. 2023b. Lifting the curse of capacity gap in distilling language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4535–4553, Toronto, Canada. Association for Computational Linguistics.

Chen Zhang, Yang Yang, Jingang Wang, and Dawei Song. 2023c. Task-agnostic distillation of encoder-decoder language models.

Chen Zhang, Yang Yang, Qifan Wang, Jiahao Liu, Jingang Wang, Yunsen Xian, Wei Wu, and Dawei Song. 2023d. Minidisc: Minimal distillation schedule for language model compression.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. 2023e. Tinyllama.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. 2024. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36.

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

Xueyan Zou, Jianwei Yang, Hao Zhang, Feng Li, Linjie Li, Jianfeng Wang, Lijuan Wang, Jianfeng Gao, and Yong Jae Lee. 2023. Segment everything everywhere all at once.