EMS-SD: Efficient Multi-sample Speculative Decoding for Accelerating Large Language Models

Anonymous ACL submission

Abstract

Speculative decoding emerges as a pivotal technique for enhancing the inference speed of Large Language Models (LLMs). Despite recent research aiming to improve prediction efficiency, multi-sample speculative decoding has been overlooked due to varying numbers of accepted tokens within a batch in the verification phase. Vanilla method adds padding tokens in order to ensure that the number of new tokens remains consistent across samples. However, this increases the computational and memory access overhead, thereby reducing the speedup ratio. We propose a novel method that can resolve the issue of inconsistent tokens accepted by different samples without necessitating an increase in memory or computing overhead. Furthermore, our proposed method can handle the situation where the prediction tokens of different samples are inconsistent without the need to add padding tokens. Sufficient experiments demonstrate the efficacy of our method. Our code will be released later.

1 Introduction

Large Language Models (LLMs) (Radford et al., 2019; Achiam et al., 2023; Touvron et al., 2023; Wang et al., 2023) have demonstrated considerable capabilities, particularly in the realm of natural language processing. Autoregressive Large Language Models generate a token in a single pass, whereas speculative decoding allows large models to generate multiple tokens in a single pass, thereby greatly improving inference speed. It is crucial to highlight that the inference time of LLMs on a single token and multiple tokens is approximate. Consequently, reducing the number of inference steps can significantly reduce the inference time.

A plethora of efficient speculative decoding methods have been proposed recently. However, none of these methods provide a comprehensive study of speculative sampling in multi-sample scenarios. To the best of our knowledge, only EAGLE (Li et al., 2024) presents results for batch sizes ≤ 4 but doesn't discuss larger batch sizes.

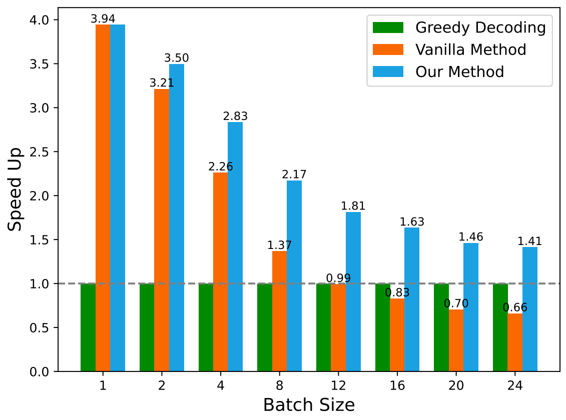


Figure 1: Speedup ratio of Opt-6.7b on the CNN/Daily Mail Dataset for greedy settings when batch size ≥ 1, utilizing LLMA (Yang et al., 2023b) as the basic speculative decoding method. Our method demonstrates superior performance to the vanilla method under varying batch sizes. The larger the batch size, the more pronounced the advantage of our method.

The primary challenge in multi-sample speculative decoding is the inconsistency in the number of accepted tokens across samples following a single inference. Vanilla solution is to add padding tokens in order to achieve uniformity. This approach is also employed by EAGLE. Nevertheless, these padding tokens increase the computational and memory access overhead, which becomes significant as batch size increases, thereby reducing speedup ratio.

Can we perform multi-sample speculative decoding without increasing computational and memory access overhead?

We proposed a novel and efficient method to resolve this issue. Specifically, we propose unpad Key-Value (KV) cache in the verification phase, which specifies the start locations of the KV cache for different samples, thus eliminating the need for padding tokens. Furthermore, in anticipation of the potential discrepancy in the number of predicted tokens across different samples, we propose the unpad input tokens method as a solution in the prediction phase. This method concatenates all input tokens prior to inference and expands these tokens during the calculation of attention.

The main contributions are as follows:

1. We proposed an Efficient Multi-sample Speculative Decoding method (EMS-SD), which takes full account of the inhomogeneity between different samples. Even if the new generated token numbers of different samples vary, the KV cache is continuous without the addition of padding tokens. Similarly, when the prediction token numbers of different samples vary, all input tokens are spliced without the addition of padding tokens.
2. Sufficient experiments have proven that our proposed method achieves a much higher speedup than vanilla methods in multi-sample speculative decoding.
3. We are the first to study speculative decoding in the context of multi-sample situations, and we have proposed an effective method for addressing this issue. Our method can be easily integrated into almost all basic speculative sampling methods.

2 Related Works

Large Language Models. Since the advent of the GPT (Radford et al., 2019) series of models, particularly after the emergence of ChatGPT (Achiam et al., 2023), there has been a proliferation of large language models, including Llama (Touvron et al., 2023), Vicuna (Chiang et al., 2023), ChatGLM (Zeng et al., 2022), QWen (Bai et al., 2023), Baichuan (Yang et al., 2023a), Gemini (Team et al., 2023), Pangu-π (Wang et al., 2023), Mistral (Jiang et al., 2023, 2024), etc.

Speculative decoding. Speculative decoding can be divided into two stages in general: prediction and verification. Some works have been published proposing efficient prediction methods. These prediction methods can be broadly classified into two categories: those that require training and those that do not. For example, methods that do not require training include LLMA (Yang et al., 2023b), REST (He et al., 2023), Lookahead (Fu et al., 2023), PLD (Saxena, 2023), etc. In contrast, methods that do require training include draft model prediction (Leviathan et al., 2023), Medusa (Cai et al., 2024), Hydra (Ankner et al., 2024), kangaroo (Liu et al., 2024), EAGLE (Li et al., 2024), and so forth.

Dynamic Tree decoding. SpecInfer (Miao et al., 2023) introduces tree decoding mechanism, which predicts multiple tokens at the same position to improve the acceptance rate. The tree structure is manually designed, and so is Medusa, EAGLE, etc. Some recent studies have focused on the problem of dynamic tree decoding. Sequoia (Chen et al., 2024) introduces a hardware-aware tree optimizer. And RSD (Jeon et al., 2024) dynamically modifies the tree structure within fixed computational budgets.

3 Approach

3.1 Rethinking Vanilla Multi-sample Speculative Decoding

Restrictions on memory access. It should be noted that mainstream AI frameworks such as PyTorch (Paszke et al., 2019) only support **aligned key-value cache access**. Consequently, two key requirements must be met for LLMs inference: (1) the number of tokens across different samples with in a batc must be equal prior to inference, and (2) the input token count must remain consistent for all samples during inference. To ensure uniformity, padding tokens are added to samples with varying token lengths. Additionally, attention masks are used to prevent the computation of padding tokens.

Add padding tokens to align the output lengths of different samples. The primary issue is that the number of accept tokens during the verification phase varies considerably between samples within a batch. To illustrate, if k tokens are predicted in the prediction stage, then the number of accept tokens can be varied from 1 to $k + 1$. Vanilla method adds padding tokens to ensure that the number of new tokens is the same for each sample within in a batch. Nevertheless, this approach leads to a considerable increase in the computational and memory access overhead, which in turn results in a significant reduction in the speedup. In Appendix B, we present a theoretical analysis of the impact of padding tokens on speedup.

Add padding tokens to align the input lengths of different samples. A further issue is that the number of predicted tokens for different samples in the prediction stage may vary. In this case, padding token also needs to be added to align the input lengths. This issue does not arise in all circumstances, and is most commonly observed in retrieval-based prediction scenarios, including

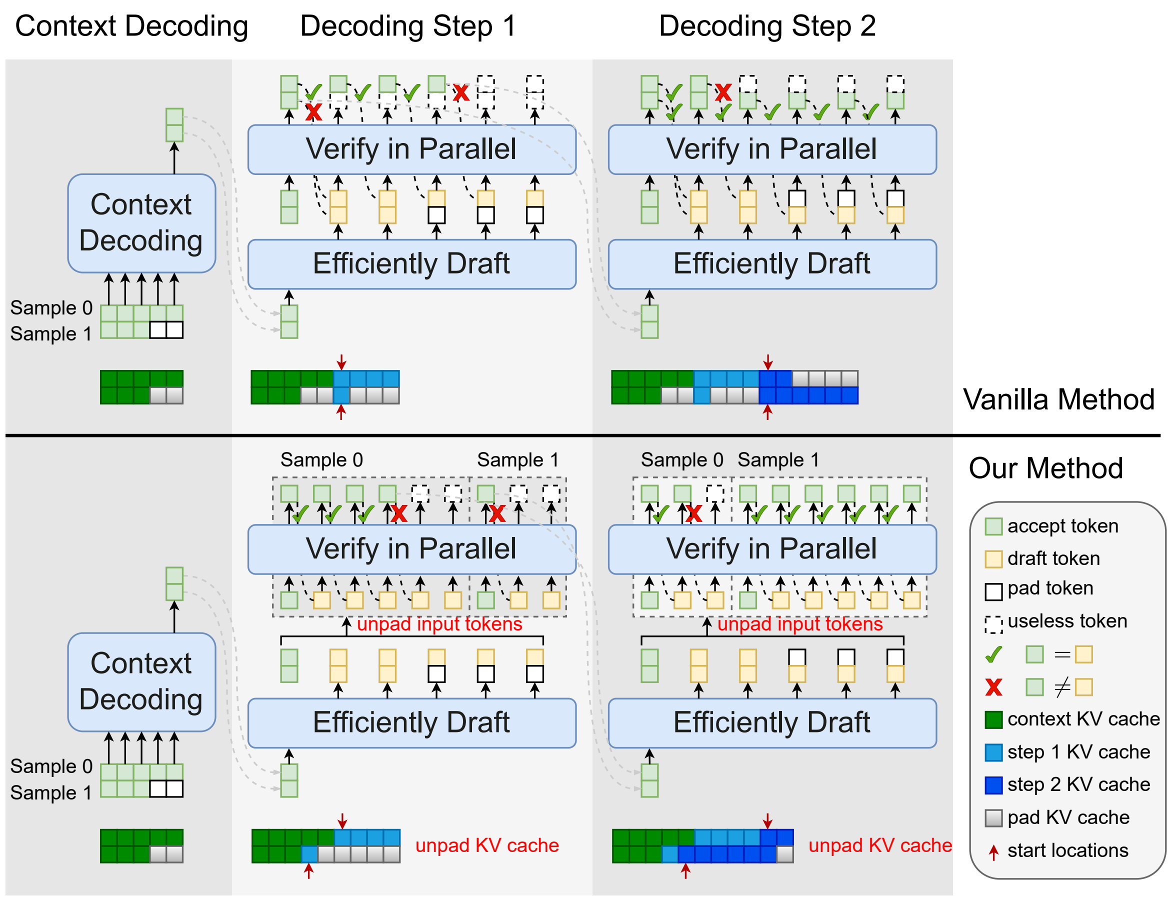


Figure 2: Our Method v.s. Vanilla Method. We specify the location of the KV cache for each sample individually, thus eliminating the necessity for the addition of padding to the KV cache. And we concatenate all input tokens of each sample into a single sequence without padding tokens when the number of prediction tokens differs between samples. Our method demonstrates superior performance than the vanilla method, without the need for additional computational and memory access overhead.

LLMA (Yang et al., 2023b) and REST (He et al., 2023). This is due to the fact that the retrieval-based prediction method employs a text matching process, whereby different samples may not be able to match the predicted text simultaneously. In more general methods, such as draft model prediction (Leviathan et al., 2023), generate same number of prediction tokens for different samples. Some recent studies have focused on the problem of dynamic tree decoding (Chen et al., 2024; Jeon et al., 2024). It is possible that in the future, there may be different optimal prediction trees or optimal numbers of tokens for different samples.

Case analysis. As illustrated in Figure 2 and Table 1, we construct two samples within a batch as an example. In the decoding step 1, sample 1 have to add 3 padding tokens in order to ensure that the input lengths are identical to those of sample 0. Subsequently, following the verification phase, sample 1 must add 3 padding tokens in the KV cache in order to ensure that the output lengths are identical to those of sample 0. In the decoding step 2, sample 0 have to add 3 padding tokens during the predication phase and 4 padding tokens in the KV cache subsequent to the verification phase.

3.2 Efficient Multi-sample Speculative Decoding

Vanilla Method tends to result in elevated computational and memory access overheads. In contrast, our approach does not entail such drawbacks, thereby conferring a higher speedup ratio. In this section, we first point out that aligned KV cache access is not immutable, and then present two key components of our approach: unpad KV cache and unpad input tokens.

Aligned KV cache access is not mandatory. In autoregressive models, each token is conditioned only on preceding tokens during the attention computation. Theoretically, given the location of the input token and access to the KV cache, we can calculate the attention output. These operations

Table 1: The two constructed samples demonstrate in Figure 2. During the two decoding steps, the number of tokens predicted and accepted by the two samples differs. If the vanilla method is employed, it's necessary to incorporate padding tokens in both predication and verification phases of speculative decoding.

Sample	**Decoding Step 1**				**Decoding Step 2**			
	Predication Phase		**Verification Phase**		**Predication Phase**		**Verification Phase**	
	Predict	Padding	Accept	Padding	Predict	Padding	Accept	Padding
0	5	0	4	0	2	3	2	4
1	2	3	1	3	5	0	6	0

can be encapsulated within CUDA kernels, as evidenced by implementations in frameworks such as FasterTransformer (NVIDIA, 2021), FlashAttention (Dao et al., 2022), and PyTorch (Paszke et al., 2019) [1]. When invoking these kernels, we can compute attention output for different tokens, even if these different tokens are in different samples and rely on different numbers of preceding tokens.

Unpad KV cache. Firstly, we introduce the first major component: unpad KV cache. This eliminates the need to add padding tokens when different samples accept different lengths in the verification phase. In particular, we specify the start location of the KV cache for each sample individually, rather than aligning writes in a manner similar to Pytorch. It should be noted that the varying start locations of samples lead to slight discrepancies in the computational workload for the attention CUDA kernels. Nevertheless, since all tokens across varying positions and samples compute their attention outputs in parallel, the overall speed is dictated by the token necessitating the greatest computational load, typically the one with the highest number of preceding tokens. As illustrated in the lower part of Figure 2, the start locations of KV cache of the two samples is distinct. For each input token, we initially compute its KV cache and subsequently write it to memory based on the specified position for each sample. Thereafter, the attention outputs for all tokens, across various samples and positions, are calculated in parallel.

By employing unique KV cache start positions for each sample, we can independently determine the subsequent start location during verification, regardless of varying acceptance lengths across samples. Consequently, this approach negates the need for extra padding tokens, thereby preventing memory waste and computational overhead. As shown in Figure 2, sample 0 accepted 4 tokens, advancing the KV cache start location by 4. While sample 1 accepted 1 token, advancing it by 1.

Unpad input tokens. Secondly, in order to address the issue of differing numbers of input tokens across different samples, we proposed the "unpad input tokens" method as a solution. In general, prior to inputting into the Transformer network, all input tokens are concatenated together, and the number of input tokens for each sample is recorded. Additionally, during the attention result calculations, the CUDA kernel reconstructs the original batch indices and sequence positions for each token. This reconstruction enables us to identify the specific KV cache that each token needs to rely on. Figure 3 shows the general processing flow. Refer to Appendix C for specific processing procedures.

[1]These frameworks provide the basic CUDA kernel for computing attention output. We need to modify these kernels to implement our method for supporting speculative decoding in multi-sample situations.

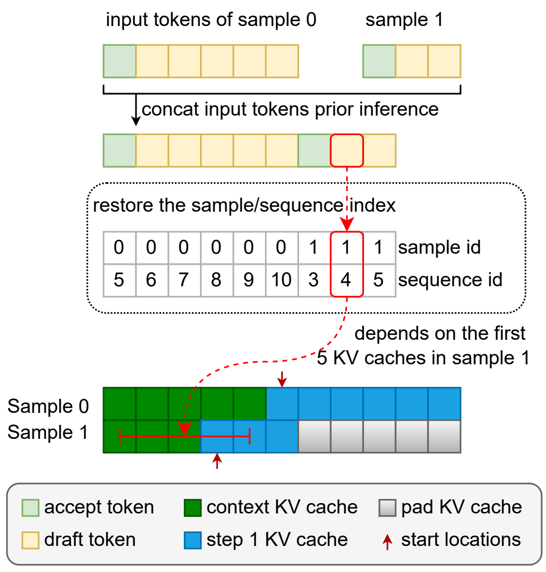


Figure 3: The detailed processing of unpad input tokens of decoding step 1 in Figure 2. Sample 0 predicted 5 tokens, while sample 1 predicted 2 tokens. All tokens are concatenated before inference, and the sample/sequence index is restored when attention is computed within the CUDA kernels. Consequently, each token is aware of the specific KV caches to which it can utilize for parallel computation.

Table 2: Ablation study using LLMA were conducted on two key methods: unpad KV cache and unpad input tokens. Under different batch sizes, our method demonstrated a significantly higher acceleration ratio than vanilla method. The method "unpad KV cache" played a more prominent role.TPS stands for tokens per second.

Batch Size	Inference Method	Unpad KV Cache	Unpad Input tokens	TPS	Speed up
1	Greedy Decoding	-	-	79.44	
	Vanilla Method	-	-	313.04	3.94
2	Greedy Decoding	-	-	137.49	
	Vanilla Method	×	×	441.64	3.21
	Our Method	✓	×	439.40	3.20
		×	✓	477.72	3.47
		✓	✓	480.59	**3.50**
4	Greedy Decoding	-	-	257.37	
	Vanilla Method	×	×	581.54	2.26
	Our Method	✓	×	610.41	2.37
		×	✓	728.86	2.83
		✓	✓	729.17	**2.83**
8	Greedy Decoding	-	-	468.89	
	Vanilla Method	×	×	640.58	1.37
	Our Method	✓	×	687.11	1.47
		×	✓	948.71	2.02
		✓	✓	1017.75	**2.17**
16	Greedy Decoding	-	-	774.59	
	Vanilla Method	×	×	640.94	0.83
	Our Method	✓	×	734.86	0.95
		×	✓	1134.25	1.46
		✓	✓	1264.07	**1.63**
24	Greedy Decoding	-	-	936.45	
	Vanilla Method	×	×	616.19	0.66
	Our Method	✓	×	708.53	0.76
		×	✓	1150.16	1.23
		✓	✓	1321.45	**1.41**

4 Experiments

4.1 Implementation details

Base Speculative Decoding Methods. The efficacy of our approach is evaluated through two fundamental speculative decoding methods. These include LLMA (Yang et al., 2023b), a retrieval-based method, and draft model prediction (Leviathan et al., 2023), which employs draft models to predict. In the LLMA method, the match length is set to 2 and the copy length to 7. In the draft model prediction method, the draft model is employed to predict 4 tokens.

Models and Datasets. We adopt the Opt (Zhang et al., 2022) Series models, including Opt-1.3b, Opt-2.7b, and Opt-6.7b. [1] For the draft model prediction method, we utilized Opt-125m as the draft model. The test data set comprised a total of 480 pieces of data selected from the CNN/Daily Mail Test subset (See et al., 2017). All experiments are conducted on a single NVIDIA A100 GPU. All experimental results were subjected to three independent tests and the mean values were calculated.

Metrics. In order to ascertain the speed of a given method, we employ the tokens per second as an indicator. Furthermore, the speed up ratio represents the multiple between the use of the speculative sampling method and its absence. Given that the generation length of the CNN/Daily Mail Dataset is relatively brief (less than 128), we limit our consideration to the incremental decoding pro-

[1] As the FasterTransformer framework itself does not support the Llama model, we did not utilize the more popular model like Llama for testing purposes.

Table 3: The efficacy of our method evaluated on two smaller models using LLMA. Our method demonstrates superior performance on different batch sizes and models of varying sizes. When the batch size increases, the speedup ratio of the original method declines rapidly, whereas our method exhibits a more gradual decline.

Model	Batch Size	Greedy Decoding	Vanilla Method		Our Method	
		TPS	TPS	Speed up	TPS	Speed up
Opt-1.3b	1	211.97	651.73	3.07		
	2	326.38	815.62	2.50	928.62	**2.85**
	4	570.51	948.01	1.66	1228.60	**2.15**
	8	1028.84	924.63	0.90	1552.83	**1.51**
	12	1437.61	876.22	0.61	1615.94	**1.12**
	16	1765.01	851.62	0.48	1557.77	0.88
Opt-2.7b	1	128.31	389.61	3.04		
	2	205.95	484.31	2.35	550.62	**2.67**
	4	362.88	545.67	1.50	707.25	**1.95**
	8	643.38	527.27	0.82	885.83	**1.38**
	12	881.70	521.42	0.59	973.53	**1.10**
	16	1087.33	521.16	0.48	1072.35	0.99

cess. In speculative decoding, the average acceptance length is a significant metric, with a larger average acceptance length often indicative of a higher speedup ratio. Since some padding tokens need to be added in the vanilla multi-sample speculative decoding method, the average padding ratio is also a significant metric. The larger the average padding ratio, the lower the speedup ratio.

Specific code implementation. Our proposed method in question necessitates the alteration of the CUDA kernel. And we implemented our method on the FasterTransformer (NVIDIA, 2021) framework, which is a widely used C++ acceleration library that facilitates the implementation of our method. The proposed methods were implemented by modifying the Python calling interface and the CUDA kernels. Further details can be found in the open-source.

4.2 Experiments using LLMA

In this section, LLMA is adopted as the basic method of speculative decoding.

Ablation study on two Key components: unpad KV cache and unpad input tokens. As illustrated in Table 2, the opt-6.7b model was employed to conduct ablation experiments on two key methods. Firstly, it can be observed that under varying batch sizes, our method exhibits a superior speedup compared to the vanilla method. When the batch size was set to 8, our method achieved a speedup of 2.17 times, whereas the vanilla method only achieved a speedup of 1.37 times. Secondly, both sub-methods are of significance, with "unpad KV cache" playing a particularly pivotal role.

Experiments on different model sizes. In addition, we tested two smaller-sized models, namely opt-1.3b and opt-2.7b. As illustrated in Table 3, the two smaller models exhibit higher speedup ratios when utilising our method in comparison to the vanilla method, regardless of the varying batch sizes. When the batch size was set to 8, the opt-1.3b model demonstrated an acceleration of 1.51 times, whereas the original method exhibited no acceleration effect and was slower than the greedy decoding method.

4.3 Experiments using Draft Model Prediction

In this section, the draft model prediction method is adopted as the basic method of speculative decoding. It is important to note that when utilizing the draft model prediction approach, the number of predictions for each sample is identical. Consequently, only "unpad KV cache" are employed in this section.

As illustrated in Table 4, we utilize the opt-125m model as the draft model, and test three models of varying sizes. Our method exhibits a superior speedup compared to the vanilla method across diverse models and varying batch sizes.

4.4 Analysis of Speedup Decrease with Multi-sample

As illustrated in Table 2, it is evident that the speedup ratio exhibits a decline in the context of multiple samples. When the batch size is set to

Table 4: Evaluating the effectiveness of our method on three models of different sizes using draft model prediction, with opt-125m model as the draft model. In models of varying sizes, our method exhibits a greater speedup than the vanilla method.

Model	**Batch Size**	**Greedy Decoding**	**Vanilla Method**		**Our Method**	
		TPS	TPS	Speed up	TPS	Speed up
Opt-1.3b	1	211.97	280.03	1.32		
	2	326.38	398.22	1.22	410.82	**1.26**
	4	570.51	604.83	1.06	628.04	**1.10**
	8	1028.84	864.43	0.84	954.51	0.93
Opt-2.7b	1	128.31	213.08	1.66		
	2	205.95	306.42	1.49	316.56	**1.54**
	4	362.88	452.65	1.25	464.56	**1.28**
	8	643.38	611.80	0.95	691.54	**1.07**
	12	881.70	685.64	0.78	843.70	0.96
Opt-6.7b	1	79.44	177.69	2.24		
	2	137.49	273.53	1.99	280.52	**2.04**
	4	257.37	432.38	1.68	438.87	**1.71**
	8	468.89	582.28	1.24	686.75	**1.46**
	12	644.47	635.49	0.99	824.23	**1.28**
	16	774.59	675.88	0.87	939.32	**1.21**
	20	863.32	709.52	0.82	1018.68	**1.18**
	24	936.45	728.04	0.78	1100.82	**1.18**

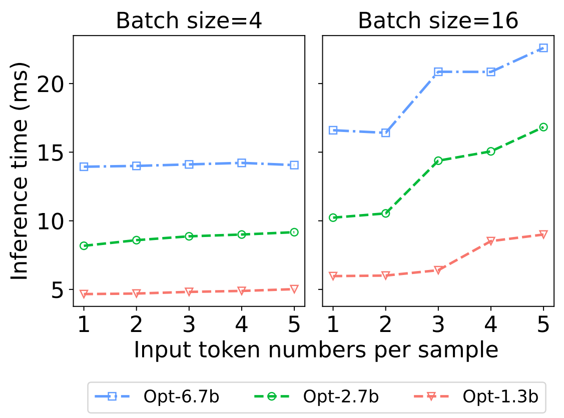


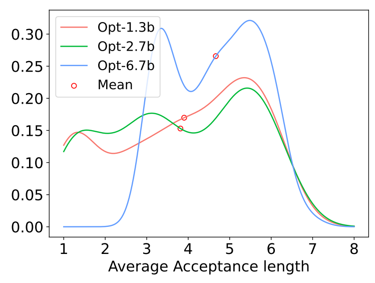
Figure 4: The inference time of different numbers of input tokens per sample under different batch sizes. We set the number of existing tokens in each sample to 512. When the number of input tokens per sample is varied with a batch size of 4, the inference time remains essentially unchanged. However, when the batch size is increased to 16, the inference time changes significantly.

4, the speedup ratio is 2.83, while when the batch size is set to 16, the speedup ratio is 1.63. Similar conclusions can be drawn from Table 4.

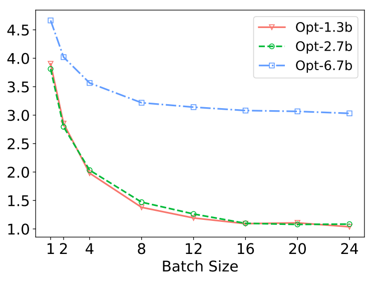
We have identified two factors contributing to the reduction in the speedup ratio. Firstly, when the batch size is sufficiently large and multiple tokens are processed simultaneously, the individual inference time for LLMs escalates considerably. As illustrated in Figure 4, the inference time for the opt-6.7b model with a batch size of 16 is 22.6 milliseconds for the processing of five tokens per sample, whereas for a single token, it is 16.6 milliseconds, which is 1.36 times slower.

Secondly, the principal reason for this decline in performance is the considerable disparity in the speedup across different samples. The average acceptance length is positively correlated with the speedup ratio. As illustrated in Figure 5(a), the average acceptance length difference of opt-1.3b/2.7b on different samples is greater than that of opt-6.7b when the LLMA method is employed. Figure 5(d) shows that the discrepancies in the average acceptance length across models were relatively modest when utilising a draft model to predict.

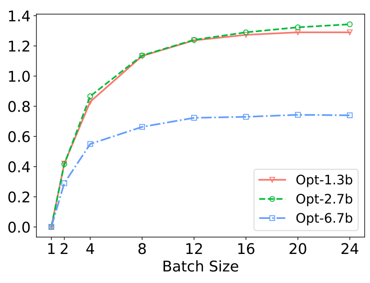
As batch size increases, the average acceptance length difference within the batch also rises, and the minimum average acceptance length within the batch decreases. As illustrated in Figure 5(b), a comparison of the opt-6.7b and opt-2.7b models reveals that the speedup ratio of the latter is more uneven on the test samples. As the batch size increases, the minimum average acceptance length within the batch decreases at a faster rate, although their speedup ratios are similar when the batch size is equal to one. Figure 5(c)(f) illustrates the pro-



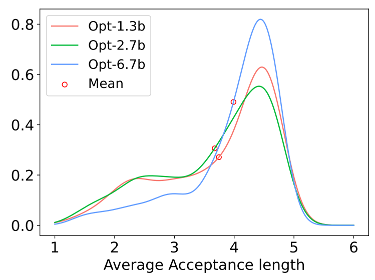
(a) Probability density



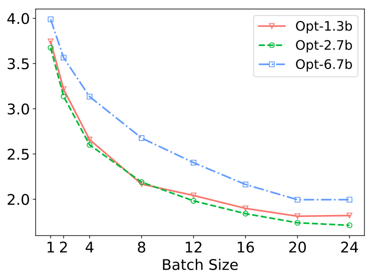
(b) Minimum average acceptance length within batch



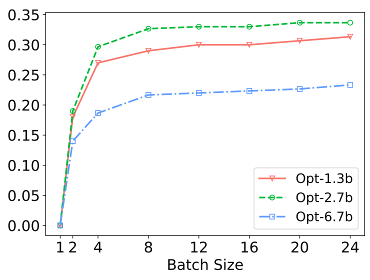
(c) Average Padding Ratio



(d) Probability density



(e) Minimum average acceptance length within batch



(f) Average Padding Ratio

Figure 5: A comprehensive analysis of the factors that contribute to the reduction of the speedup ratio in multi-batch scenarios. Figure (a)(b)(c) employ LLMA as the basic speculative decoding method, while Figure (d)(e)(f) utilize the draft model prediction method, utilising opt-125m as the draft model. Figure (a)(d) illustrates the probability density function of the average acceptance length of distinct samples, with batch size set to 1. It is evident that the average acceptance length of different samples exhibits considerable variability. Figure (b)(e) illustrates the reduction in the minimum average acceptance length within batch as the batch size increases. Given that the speedup ratios of different samples within the batch are disparate, the inference time of this batch is constrained by the slowest sample. Figure (c)(f) illustrates the ratio of the number of padding tokens to the total number of newly generated tokens, as a function of varying batch sizes, when employing the vanilla method.

portion of padding tokens in multi-sample cases utilising vanilla method. The opt-6.7b model, with batch size set to 8, exhibits a significant increase in the number of padding tokens, exceeding 60% using LLMA and exceeding 20% using draft model prediction. This explains why vanilla method have serious speedup ratio degradation in multi-sample cases.

In order to maintain the speedup ratio in the case of multiple samples, the most straightforward method is to ensure that the speedup ratios of different samples are similar under the basic speculative decoding method. However, the optimal solution to this issue is dynamic batching (Yu et al., 2022), which entails replacing a finished sample in the batch with a new sample once it has been completed, rather than waiting for all samples in the batch to be completed before proceeding to the next inference. The implementation of dynamic batching is expected to enhance the efficiency of multi-sample processing, with the potential for achieving a comparable speedup to that in single-sample cases.

5 Conclusions

In this paper, we present the first study of multi-sample speculative sampling. we introduce an effective method, called EMS-SD. EMS-SD is an effective solution to the inconsistency problem of different samples in the prediction and verification stages, without the need of padding tokens. The proposed method is flexibly integrated with almost any basic speculative decoding method. Extensive comparisons show that EMS-SD exhibits superior performance compared to the vanilla method in multi-sample speculative decoding.

Limitations

This work has four limitations: 1) Theoretical evidence indicates that dynamic batching may serve to mitigate the performance degradation that occurs in multi-sample speculative decoding. However, this has not been empirically validated. Subsequent experiments will assess the efficacy of multi-sample speculative decoding in conjunction with dynamic batching. 2) The potential negative impact of non-contiguous memory accesses on performance was not considered. In batched greedy decoding, the memory access between different samples is continuous. However, in the proposed method, due to the varying lengths of different samples, the memory access is not continuous. This may have a negative effect on acceleration. 3) Although our method is independent of the inference framework, we have not yet implemented our method on frameworks such as PyTorch (Paszke et al., 2019) or vLLM (Kwon et al., 2023). This undoubtedly limits the ease of use of our method. In subsequent work, we will consider implementing our method in these frameworks. 4) Tree decoding will further accelerate speculative decoding, which has been widely verified in the single-sample speculative decoding (Miao et al., 2023; Cai et al., 2024; Liu et al., 2024; Li et al., 2024). Nevertheless, the efficacy of integrating tree decoding with multi-sample speculative reasoning has yet to be validated. Future experiments will evaluate the effectiveness of the multi-sample speculative decoding when integrated with tree decoding.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Zachary Ankner, Rishab Parthasarathy, Aniruddha Nrusimha, Christopher Rinard, Jonathan Ragan-Kelley, and William Brandon. 2024. Hydra: Sequentially-dependent draft heads for medusa decoding.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

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*.

Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. 2024. Sequoia: Scalable, robust, and hardware-aware speculative decoding. *arXiv preprint arXiv:2402.12374*.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)*, 2(3):6.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. 2023. Breaking the sequential dependency of llm inference using lookahead decoding.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason D Lee, and Di He. 2023. Rest: Retrieval-based speculative decoding.

Wonseok Jeon, Mukul Gagrani, Raghavv Goel, Junyoung Park, Mingu Lee, and Christopher Lott. 2024. Recursive speculative decoding: Accelerating llm inference via sampling without replacement. *arXiv preprint arXiv:2402.14160*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088*.

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

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.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024. Eagle: Speculative sampling requires rethinking feature uncertainty. In *International Conference on Machine Learning*.

Fangcheng Liu, Yehui Tang, Zhenhua Liu, Yunsheng Ni, Kai Han, and Yunhe Wang. 2024. Kangaroo: Lossless self-speculative decoding via double early exiting. *arXiv preprint arXiv:2404.18911*.

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*.

NVIDIA. 2021. Fastertransformer. `https://github.com/NVIDIA/FasterTransformer`.

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

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.

Apoorv Saxena. 2023. Prompt lookup decoding.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada. Association for Computational Linguistics.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

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

Yunhe Wang, Hanting Chen, Yehui Tang, Tianyu Guo, Kai Han, Ying Nie, Xutao Wang, Hailin Hu, Zheyuan Bai, Yun Wang, et al. 2023. Pangu-π: Enhancing language model architectures via nonlinearity compensation. *arXiv preprint arXiv:2312.17276*.

Aiyuan Yang, Bin Xiao, Bingning Wang, Borong Zhang, Ce Bian, Chao Yin, Chenxu Lv, Da Pan, Dian Wang, Dong Yan, et al. 2023a. Baichuan 2: Open large-scale language models. *arXiv preprint arXiv:2309.10305*.

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

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.

Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, et al. 2022. Glm-130b: An open bilingual pre-trained model. *arXiv preprint arXiv:2210.02414*.

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*.

A Preliminaries

Autoregressive Decoding. Autoregressive large language models (LLMs) P generates a token at each step. Let x be the sequence of tokens, with x_j denoting the token at position j. The probability distribution of the the token at position i over vocabulary V, y_i, is contingent upon the preceding tokens. Consequently, y_i can be expressed as Equation 1.

$$y_i \sim P(y|x_{[0,i)}) \tag{1}$$

For greedy decoding, the subsequent token is selected according to the maximum value of the probability distribution.

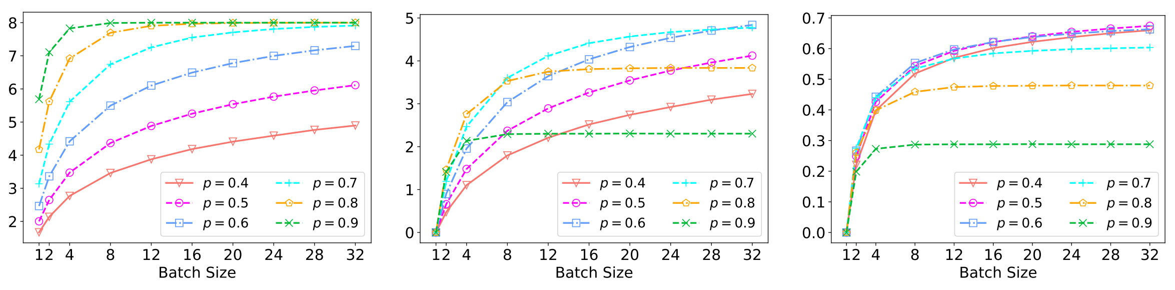
$$x_i = \arg\max_{y \in V} y_i \tag{2}$$

Single-sample Speculative Decoding. With regard to Speculative Decoding, the process can be divided into two stages: prediction and verification. In the prediction stage, a prediction method, f, is employed to predict the subsequent k tokens $d_i, .., d_{i+k-1}$ at each step.

$$d_i, .., d_{i+k-1} = f(x_{[0,i)}) \tag{3}$$

In the verification stage, these k predicted tokens are simultaneously input to the LLMs, together with existing tokens. This enables the LLMs to generate $k+1$ tokens in a single decoding.

$$\begin{gathered} x_j = \arg\max_{y \in V} P(y|x_{[0,i)}, d_{[i,j)}), \\ i \leq j < i+k+1 \end{gathered} \tag{4}$$



(a) Maximum Acceptance Length (b) Average Padding Length (c) Average Padding Ratio

Figure 6: Numerical simulation of the expected value of three variables: the maximum acceptance length τ_{max}, the average padding length $\overline{\delta}$ and the average padding ratio $\overline{r}$.

The output token x_i is identical to the result generated by the Autoregressive method. Nevertheless, it is essential to ascertain the remaining k tokens ($x_j, i+1 \leq j < i+k+1$) to ascertain their acceptability. The acceptance length τ can be calculated using the Equation 5. Since d_{i+k} is undefined, it follows that x_{i+k} and d_{i+k} are always unequal.

$$\begin{aligned} \tau = \arg\max_j \{ & x_{i+j-1} \neq d_{i+j-1} | \\ & x_{i+m-1} = d_{i+m-1}, 1 \leq m < j\}, \\ & 1 \leq j \leq k+1 \end{aligned} \tag{5}$$

Consequently, the LLMs is capable of accepting τ tokens simultaneously, rather than just one, within a similar timeframe. It is important to note that the average acceptance length $\overline{\tau}$ and the speedup ratio are closely related. As the average acceptance length increases, the speedup ratio also rises.

B Theoretical Analysis of the Impact of Padding Tokens in Vanilla Multi-sample Speculative Decoding

As mentioned in Section 3.1, the introduction of additional padding tokens in the vanilla method will result in a reduction in speedup ratio in multi-sample cases. We assume that k tokens in the prediction stage, and the prediction accuracy of the next token is p, thus the accepted length τ conforms to the geometric distribution. The probability mass function of τ can be formulated as Equation 6. And the expected value $E(\tau)$ is formulated as Equation 7.

$$P(\tau = k) = p^{k-1}(1-p), k = 1, 2, 3, 4, ... \tag{6}$$

$$E(\tau) = \frac{1}{1-p} \tag{7}$$

If b samples are inferred simultaneously (batch size is set to b), the maximum acceptance length τ_{max} and the average padding length $\overline{\delta}$ can be expressed as Equation 8 and 9. Furthermore, we define the average padding ratio $\overline{r}$, which is the ratio of $\overline{\delta}$ and τ_{max}, as shown in Formula 10. It can be demonstrated that as the value of $\overline{r}$ increases, the proportion of padding also increases, resulting in a greater waste of computational and memory access overhead. It can be observed that as the average padding ratio increases, the negative impact on acceleration also increases.

$$\tau_{max} = max(\tau_0, \tau_1, ..., \tau_{b-1}) \tag{8}$$

$$\overline{\delta} = \tau_{max} - \frac{1}{b}(\tau_0 + \tau_1 + ... + \tau_{b-1}) \tag{9}$$

$$\overline{r} = \overline{\delta}/\tau_{max} \tag{10}$$

The probability mass function of τ_{max} can be expressed as Equation 11.

$$P(\tau_{max} = k) = \begin{cases} (1-p^k)^b - (1-p^{k-1})^b, & \text{if } k > 1 \\ (1-p)^b, & \text{if } k = 1 \end{cases} \tag{11}$$

The expected value of τ_{max} and $\overline{\delta}$ are challenging to express in a concise manner. This is why we employed numerical simulation method. As shown in Figure 6, We show the expected value of τ_{max}, $\overline{\delta}$ and $\overline{r}$ as they vary with the prediction accuracy of next token p and batch size b. In practical applications, the maximum acceptance length is constrained by the limitations of the predicted length. In this figure, we limited the maximum acceptance length to 8.

Two conclusions can be drawn from Figure 6. Firstly, the maximum acceptance length and the average padding length both increase as the prediction accuracy of next token and the batch size increase. Secondly, when the maximum acceptance length is limited, the higher the prediction accuracy of next token, the lower the average padding ratio. In particular, when the prediction accuracy of next token is below 0.8 and the batch size is greater than 8, the padding ratio increases rapidly to exceed 50%. However, even if the prediction accuracy reaches 90%, 30% of the computational and memory access overhead are still wasted.

C The Specific Process of Unpad Input Tokens

Before being fed into the Transformer model, all input tokens are merged into a single sequence, and the quantity of input tokens for each sample is documented.This process is detailed in Algorithm 1.

Furthermore, when computing the attention output, the original batch index and sequence position of each token are restored in the CUDA kernel.

This process is detailed in Algorithm 2. In addition, modifications will be required to the grid responsible for invoking the CUDA kernel during the attention calculation process. Equation 12 illustrateds the specific alterations. In the context of the CUDA kernel, the value of $blockIdx.y$ represents the index of the current token among all inputs.

$$\begin{gathered} grid(num\_heads, batch\_size) \rightarrow \\ grid(num\_heads, total\_input\_token\_nums) \end{gathered} \tag{12}$$

Algorithm 1: Concatenate the input tokens of different samples

```
Data: $list\_of\_input\_tokens$
Result: $concatenated\_input\_tokens$, $token\_nums\_per\_sample$, $total\_input\_token\_nums$
$batch\_size$ = len( $list\_of\_input\_tokens$ )
$concatenated\_input\_tokens$ = []
$token\_nums\_per\_sample$ = [0 for _ in range($batch\_size$)]
$total\_input\_token\_nums$ = 0
for $i = 0$ to $batch\_size - 1$ do
    $total\_input\_token\_nums$ += len($list\_of\_input\_tokens[i]$)
    $token\_nums\_per\_sample[i]$ = len($list\_of\_input\_tokens[i]$)
    $concatenated\_input\_tokens$.extend($list\_of\_input\_tokens[i]$)
end
```

Algorithm 2: Restore the original batch index and sequence position in CUDA kernels

```
Data: $token\_nums\_per\_sample$, $blockIdx$
Result: $original\_batch\_index$, $original\_sequence\_position$
$batch\_size$ = len( $token\_nums\_per\_sample$ )
$original\_sequence\_position$ = $blockIdx.y$
$original\_batch\_index$ = 0
for $i = 0$ to $batch\_size - 1$ do
    if $original\_sequence\_position \geq token\_nums\_per\_sample[i]$ then
        $original\_batch\_index$ += 1
        $original\_sequence\_position$ -= $token\_nums\_per\_sample[i]$
    else
        break
    end
end
```