Accelerating LLM Inference with Staged Speculative Decoding

Anonymous Authors[1]

Abstract

Recent advances with large language models (LLM) illustrate their diverse capabilities. We propose a novel algorithm, staged speculative decoding, to accelerate LLM inference in small-batch, on-device scenarios. We address the low arithmetic intensity of small-batch inference by improving upon previous work in speculative decoding. First, we restructure the speculative batch as a tree, which reduces generation costs and increases the expected tokens per batch. Second, we add a second stage of speculative decoding. Taken together, we reduce single-batch decoding latency by 3.16x with a 762M parameter GPT-2-L model while perfectly preserving output quality.

1. Introduction

Large Language Models (LLMs) have witnessed tremendous growth over the last few years, demonstrating capabilities that range from high-quality text generation to complex reasoning, decision-making, and problem-solving tasks (Brown et al., 2020; OpenAI, 2023; Chowdhery et al., 2022). These strides, enabled by advances in deep learning architectures (Vaswani et al., 2017), training methodologies (Kingma & Ba, 2014), and vast amounts of data (Halevy et al., 2009; Gao et al., 2020; Kocetkov et al., 2022), have paved the way for applications in fields as varied as natural language processing (Brown et al., 2020), machine translation (Raffel et al., 2020), code synthesis (Chen et al., 2021), and beyond (OpenAI, 2023).

However, this exciting progress comes with its own set of system-level challenges. As LLMs have become more powerful, their computational demands have increased in tandem, often requiring substantial cloud resources for inference (Sheng et al., 2023). This requirement is prohibitive for many potential applications, especially those requiring low-latency responses (Wang et al., 2023) or those where data privacy is paramount (Carlini et al., 2021).

Our paper addresses these challenges by accelerating local (small-batch) inference for LLMs, which suffers from poor compute utilization due to its low arithmetic intensity. We view this problem as crucial for three reasons: latency, personalization, and privacy. First, optimizing local inference latency improves real-time interactivity and responsiveness. Accelerating local inference also opens the door for more personalized LLM experiences as it allows models to be locally tailored to individual users. Finally, local inference enhances data privacy, as it removes the need for data to leave the user's device.

More philosophically, we believe that methods to efficiently run LLMs locally promote AI democratization by empowering individuals with limited computational resources.

In this work, we build on the speculative decoding techniques introduced by (Leviathan et al., 2022; Chen et al., 2023), which use a fast but inaccurate draft model to anticipate the oracle model and batch queries to it, which improves sequential decoding performance while perfectly retaining the model distribution. These techniques scale well at first but their performance gains quickly saturate, because the probability of a draft model correctly guessing many sequential tokens is exponentially small. We improve speculative methods in two key ways:

1. We restructure the speculative batch as a tree of possible token sequences, so as to more quickly create larger and higher quality speculative batches.

2. We speculatively decode the draft model, too, to further improve performance.

We find these techniques significantly improve the performance of speculative decoding in both deterministic and sampling-based decoding.

2. Background

In this section, we provide a brief overview of autoregressive LLM inference, key principles of GPU performance optimization, and prior work in optimizing LLM inference.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

2.1. Autoregressive LLM Inference

Autoregressive generation from decoder-only LLMs is generally split into two phases. First, the prompt is run through the model to generate the KV cache and the first output logits. This is usually fast, as the entire prompt can be handled in parallel.

The second phase is decoding. A token is selected from the outputted logits and fed back into the model, which produces logits for the following token. This is repeated until the desired number of tokens is produced. Because decoding must be done sequentially, with the entire model's weights streamed through the compute units each time in order to generate a single token, the arithmetic intensity (that is, FLOP of compute / byte of memory bandwidth) of this second phase is extremely low when run in small batches. As such, decoding is usually the most expensive part of autoregressive generation. (Leviathan et al., 2022)

2.2. GPU optimization

Modern LLM inference is most often conducted on GPUs due to the highly parallel nature of the workload, which consists principally of large matrix multiplications.

GPUs consist of thousands of extremely small efficient cores supported by a multi-level memory hierarchy. The key challenge of optimizing small-batch LLM inference for GPUs is to deal with the extremely low arithmetic intensity. Operating in 16-bit precision with a batch size of 1, decoding has an arithmetic intensity of 1. For example, for a reference PyTorch (Paszke et al., 2019) implementation of GPT-2 Large (762M parameters), inference requires approximately 1.4 GFLOP, and yet a quiesced NVIDIA RTX 4090 achieves only 150 tokens/second, for a compute utilization of a mere 0.13% (NVIDIA, 2022). This abysmal performance is substantially due to the GPU roofline (Ofenbeck et al., 2014), which is governed by memory bandwidth at low arithmetic intensities (visualized in Figure 1).

2.3. Speculative Decoding

There are many techniques under investigation today to accelerate inference, such as quantization (Dettmers et al., 2022; Frantar et al., 2022), flash attention (Dao et al., 2022), and speculative decoding (Leviathan et al., 2022; Chen et al., 2023). In this section, we'll briefly examine speculative decoding as described in (Leviathan et al., 2022; Chen et al., 2023), as it is the primary subject of this work.

The basic idea of speculative decoding is to use a smaller, faster draft model to decode several tokens in advance, and then feeds them into the oracle model as a single batch. If the draft model was right about its predictions – the larger model agrees – one can decode several tokens with a single

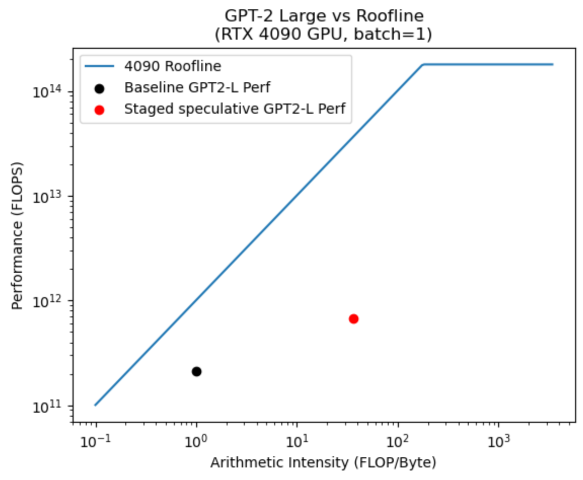


Figure 1: A roofline plot for single-query GPT-2-L inference on an RTX 4090. At small batch sizes, inference is completely memory bandwidth bound. Thus this plot shows that the only way to significantly increase performance is to increase the arithmetic intensity of inference.

batch, which saves considerable memory bandwidth, and thus time, per token. However, if the larger model rejects the tokens predicted by the draft model, then the rest of the batch is discarded and the algorithm naturally reverts to standard token-by-token decoding. Speculative decoding may also be accompanied by a rejection sampling scheme to sample from the original distribution. Note this is only useful in small-batch settings where bandwidth is the bottleneck. Speculative decoding trades compute for bandwidth.

There are two key reasons why speculative decoding is an attractive performance engineering target. First, it does not degrade model quality at all. Second, the gains it provides are generally orthogonal to other methods, because its performance comes from converting sequential execution to parallel execution. (Leviathan et al., 2022)

3. Methods

We make two improvements to speculative decoding: tree-structured batches, and additional stages. We term the combination of these methods "staged speculative decoding".

3.1. Tree-structured batches

Current speculative methods predict a single sequence for the batch. However, this doesn't scale well to large batch sizes or low draft model alignments. Intuitively, the probability that two models agree for long consecutive sequences of tokens is exponentially low, which means that speculative decoding has rapidly diminishing returns as one scales its arithmetic intensity.

Sampling method	Baseline rel. bandwidth	Speculative rel. bandwidth	Staged spec. rel. bandwidth
Deterministic	1.00	0.31	**0.23**
Topk	1.00	0.48	**0.35**

Table 1: Memory bandwidth consumption (relative to baseline) of speculative and staged speculative decoding methods.

Our approach is to dynamically build a tree of the possible sequences, which provides three benefits: more expected true tokens per batch, an increased number of leaf nodes, and better parallelism for the small draft model.

First, by reallocating computation from the end of very long sequences to the beginning, and considering the second or third most likely tokens to be produced by the model, one increases the expected number of tokens per batch compared to the naive approach.

Second, the cost of running the draft model to produce the batch is non-negligible in standard speculative decoding. However, in a tree of predictions which constitute the batch to the oracle model, the draft model is only run at internal nodes of the tree. So, a wider tree increases the number of leaf nodes, which means that one gets more of the batch for free.

A third benefit of the wider tree is that one can parallelize execution for the small model across the tree, which also decreases its cost. In the limit, one only needs to run the draft on a number of batches equal to the depth of the tree. This is important because draft models are usually smaller transformer-based models and are thus memory-bound in small-batch inference, too.

Implementing a tree-structured batch requires some care. The simplest approach is to partition self-attention while decoding into cross-attention with the KV cache and self-attention within the batch. The tree-structured batch can then be constructed by controlling both the positional embeddings and causally masking the batch self-attention matrix according to the tree. Finally, the new KV cache for the whole batch must be stored separately, and then the appropriate slices appended to the main KV cache after tokens are sampled.

3.2. Staged Speculation

Current speculative methods use a single smaller model as the draft, usually a smaller LLM (Chen et al., 2023). In this setting, the size of the draft model is an important hyperparameter: a larger draft model will have better alignment with the oracle, but will cost more, whereas a smaller model will produce lower quality speculative batches, but at a lower cost. In practice, draft models that are about 15-20x smaller than the oracle seem optimal.

Sampling method	Baseline tokens/sec	Speculative tokens/sec	Staged spec. tokens/sec
Deterministic	150	350	**475**
Topk	150	219	**298**

Table 2: Relative performance (in tokens/second decoded) with baseline (non-speculative), standard speculative, and staged speculative decoding methods.

However, under naive speculative decoding, assembling large batches inverts the cost structure, with more time spent on the draft model than the oracle. So, one should accelerate the draft model in generating sequences of tokens, and speculative decoding is a natural solution for this, too. We correspondingly add speculative decoding to the draft model in our approach. Thus the overall method of "staged speculative decoding", consists of oracle, draft, and draft^2 models with tree-structured batches.

4. Results

For our experiments, we use three models: a GPT-2-Large (762M) parameter oracle model (Radford et al., 2019) fine-tuned on the Python subsection of the Stack (Kocetkov et al., 2022), a small (40M) parameter GPT-2 draft model trained on the same, and a Katz backoff trigram model (Katz, 1987) as the draft^2 model. The Katz backoff model was generated by running the draft model for two hours at a sampling temperature of 1.5 to generate 120M tokens. All evaluations were conducted on a quiesced RTX 4090 GPU (NVIDIA, 2022), which is top-end consumer hardware.

We evaluate against two alternative inference methods. First, our standard baseline is simple token-by-token decoding with the oracle. Second, we also evaluate against speculative decoding as proposed by (Leviathan et al., 2022), so as to isolate the effects of our improvements.

To evaluate, we ran the 164 prompts from HumanEval (Chen et al., 2021), using non-speculative, speculative, and our staged speculative methods, and with both deterministic and topk sampling (Radford et al., 2019). Details of batch sizes and internal heuristics can be found in our code.

We first measured the memory bandwidth requirements of each method, to validate that our approach saves appreciable bandwidth. We detail the results in table 1, which illustrate that staged speculative decoding uses substantially less memory bandwidth than either alternative method.

Second, we measure sequential decoding throughput for each approach. The results are summarized in table 2 and detailed in Figure 2.

With deterministic sampling, our implementation provides an average performance boost of 3.16x over our reference

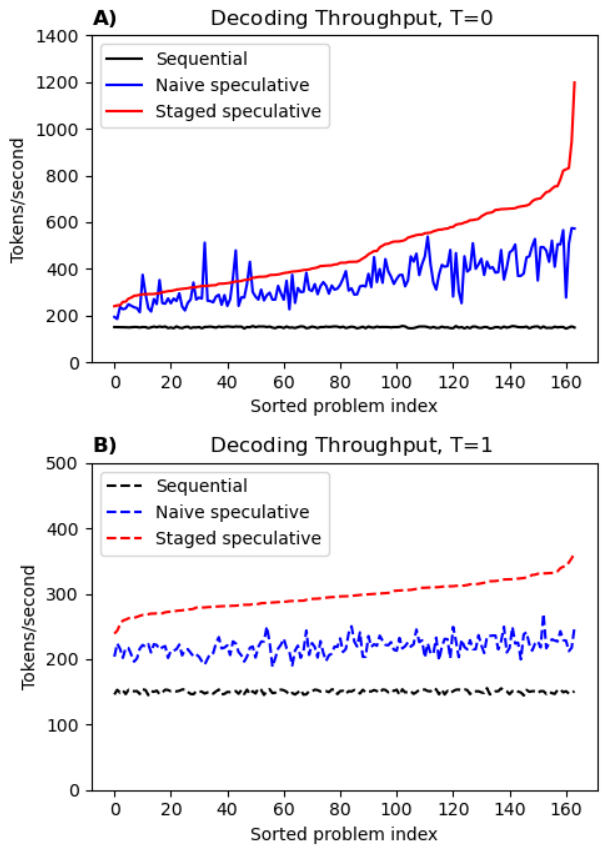
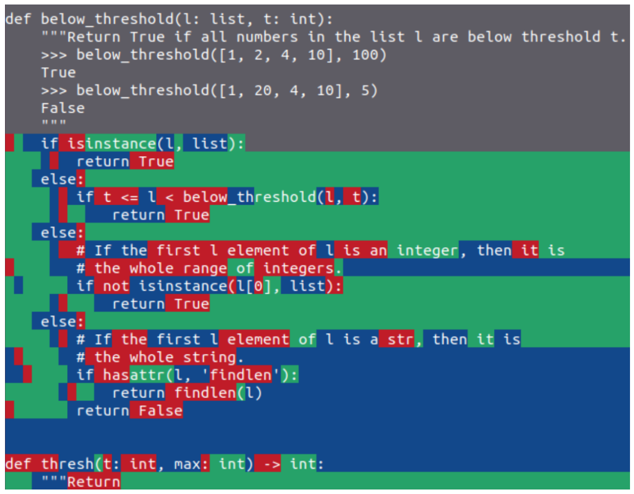


Figure 2: Relative performance distribution over different problems in the HumanEval dataset. (A) shows greedy decoding, whereas (B) shows Topk decoding. Problem indices are sorted by staged speculative performance for clarity.

implementation, and 1.36x over standard speculative sampling. Furthermore, we evaluate on relatively small models, whereas prior work uses much larger models on which one would expect greater benefits. Profiling data shows our implementation has 35% overhead from the Python infrastructure, which could be reduced by a more efficient implementation or amortized over larger models.

With topk ($k = 50, T = 1$) sampling, although both speculative methods are significantly degraded due to stochastic rejection of tokens provided in the batch, staged speculation nonetheless retains its lead, providing an average performance boost of 1.98x over baseline and 1.36x again over standard speculative sampling.

In Figure 3, we show the origin of different tokens in the completed model. (The performance gain on the shown prompt is approximately 2.5x over baseline.) The model is usually able to decode the easiest, most obvious tokens, like whitespace, in batch through both transformer models, as they originate with the N-gram models. Somewhat more difficult tokens are generated by the small model, while



```
def below_threshold(l: list, t: int):
    """Return True if all numbers in the list l are below threshold t.
    >>> below_threshold([1, 2, 4, 10], 100)
    True
    >>> below_threshold([1, 20, 4, 10], 5)
    False
    """
    if isinstance(l, list):
        return True
    else:
        if t <= l < below_threshold(l, t):
            return True
    else:
        # If the first l element of l is an integer, then it is
        # the whole range of integers.
        if not isinstance(l[0], list):
            return True
    else:
        # If the first l element of l is a str, then it is
        # the whole string.
        if hasattr(l, 'findlen'):
            return findlen(l)
        return False


def thresh(t: int, max: int) -> int:
    """Return
```

Figure 3: A visualization of the origin of tokens in an example T=1 HumanEval completion. Green background originates with the N-gram draft[2] model, blue the draft model, and red the oracle model. (Of course, all tokens are eventually checked by the oracle model.) Obvious tokens – like whitespace – are preferentially accelerated relative to difficult ones.

the most critical tokens (like the token following the "if" token) come from the oracle model. Note that due to the finite batch size, the above is only a trend and should not be expected to apply universally to every token. Some tokens which could have been accurately predicted by a smaller model will still end up originating from larger models.

We also wish to acknowledge the extreme range of the performance benefits as a downside of the work. While performance benefits run as high as 10x on realistic prompts, they can also be limited to only 2x. To a large degree, this depends on the denseness or sparseness of difficult content. For example, highly indented Python code will make better use of the N-gram models than unindented code, and thus reap greater performance benefits.

We speculate that these models represent an approximately fixed cost per entropy of the data. Extremely low entropy generation, like pure whitespace, will be generated very quickly by staged speculative decoding, with performance approaching that of large-batch inference, whereas dense generations with high entropy will need to rely on small-batch decoding at all stages. So, a corollary implication of this work is that most of the text generated by LLMs has entropy lower than the capabilities of their authoring models, and that the increased accuracy of big models is isolated to a relatively small number of key tokens.

We see several paths for future work:

1. We suspect it may be possible to speculatively sample with $T > 0$ even faster by generating the multinomial CDFs first, and then using this sequence to help choose

the tokens to assemble into the full batch. For example, if the multinomial CDF sampled is 0.99, it may be best to only include in the batch the draft model's fifth through tenth most likely tokens.

2. Running with larger models would likely yield even greater performance boosts while still fitting on-device. With 8-bit quantization, it should be possible to fit 20B models on consumer GPUs in small-batch, allowing for an entire additional stage of speculation. ($20B \rightarrow 1B \rightarrow 50M \rightarrow$ N-gram).

3. Investigating better lowest-level draft models could also improve performance – models which perform better than N-gram models but still run in $< 10\mu s$.

5. Conclusions

In this work, we described and implemented several improvements over previous work in speculative decoding. First, we restructured the batch provided to the oracle model as a tree, in order to decrease the cost of generation and increase the expected number of tokens per batch. Second, we added a second stage of speculation to accelerate the decoding of the draft model. Altogether, we achieved an average speedup of 3.16x over standard single-batch inference.

Acknowledgements

[Left blank for blind review.]

References

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Carlini, N., Tramer, F., Wallace, E., Jagielski, M., Herbert-Voss, A., Lee, K., Roberts, A., Brown, T. B., Song, D., Erlingsson, U., et al. Extracting training data from large language models. In *USENIX Security Symposium*, volume 6, 2021.

Chen, C., Borgeaud, S., Irving, G., Lespiau, J.-B., Sifre, L., and Jumper, J. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

Dao, T., Fu, D., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. Llm. int8 (): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*, 2022.

Frantar, E., Ashkboos, S., Hoefler, T., and Alistarh, D. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Gao, L., Biderman, S., Black, S., Golding, L., Hoppe, T., Foster, C., Phang, J., He, H., Thite, A., Nabeshima, N., et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

Halevy, A., Norvig, P., and Pereira, F. The unreasonable effectiveness of data. *IEEE intelligent systems*, 24(2): 8–12, 2009.

Katz, S. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE transactions on acoustics, speech, and signal processing*, 35(3):400–401, 1987.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kocetkov, D., Li, R., Allal, L. B., Li, J., Mou, C., Ferrandis, C. M., Jernite, Y., Mitchell, M., Hughes, S., Wolf, T., et al. The stack: 3 tb of permissively licensed source code. *arXiv preprint arXiv:2211.15533*, 2022.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Leviathan, Y., Kalman, M., and Matias, Y. Fast inference from transformers via speculative decoding. *arXiv preprint arXiv:2211.17192*, 2022.

NVIDIA. Nvidia RTX 4090 GPU architecture, 2022.

Ofenbeck, G., Steinmann, R., Caparros, V., Spampinato, D. G., and Püschel, M. Applying the roofline model. In *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 76–85. IEEE, 2014.

OpenAI. Gpt-4 technical report, 2023.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library, 2019.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

Sheng, Y., Zheng, L., Yuan, B., Li, Z., Ryabinin, M., Fu, D. Y., Xie, Z., Chen, B., Barrett, C., Gonzalez, J. E., et al. High-throughput generative inference of large language models with a single gpu. *arXiv preprint arXiv:2303.06865*, 2023.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Wang, Y., Chen, K., Tan, H., and Guo, K. Tabi: An efficient multi-level inference system for large language models. In *Proceedings of the Eighteenth European Conference on Computer Systems*, pp. 233–248, 2023.