Milestone 1
Distributed Serving System for Transformer-Based Generative Models

Jeremy Money, Paratee Komen, and Vu Nguyen

Introduction:

Large language models (LLMs) represent a subset of transformer-based natural language processing systems which have recently been given large amounts of attention for a variety of uses. With increasing scale beyond 500 billion parameters, there exists a need for these models to make efficient use of their increasing memory requirements while rendering services at the end user level. Furthermore, for dynamic use cases such as interactive applications like chatbots, the optimization problem for resource utilization requires efficient use of resource query/partitioning strategies ad hoc. Of particular note is the managing of Key Value (KV) cache memory, which is responsible for the attention tensors of each layer. KV cache memory management can limit throughput and batch sizes of LLMs, and as such many organic serving systems have fallen short in optimizing throughput [1].

Novel partitioning strategies involving inference serving engines have arisen to meet this demand to efficiently handle workloads of LLMs. One such engine, known as vLLM, is based on PagedAttention, which uses the concept of "paging" associated with operating systems. This algorithm essentially stores KV caches in blocks of non-contiguous physical memory similar to the way virtual memory works in traditional operating systems [1,2]. A similar strategy using FlashAttention, combines attention partition into blocks with a technique known as recomputation, which selectively "checkpoints" the gradients of the attention matrices to allow for efficient memory usage in the backwards pass stage [4].

The focus of this work is to evaluate two primary partitioning strategies from frameworks for use in large language models, and to gain a better understanding on how they influence the overall latency and throughput of commonly used LLMs. These two partitioning strategies involve Inference Serving using vLLM and Hugging Face's Text Generation Inference. While work from previous groups have shown the value added by the former, this work seeks to replicate these results and gain an understanding of how token sequence length and variation in token length effects these performance metrics.

Problem Statement:

In order to understand modern inference serving strategies for LLMs, it is necessary to compare and benchmark the performance of the existing open-source LLM inference systems. As such, evaluating the throughput and latency of several commonly serviced LLMs both with and without the aforementioned algorithms can show the importance of coupling optimized inferencing on memory utilization. Additionally, by developing this fundamental understanding, additional strategies for efficient memory/resource allocation can eventually be generated.

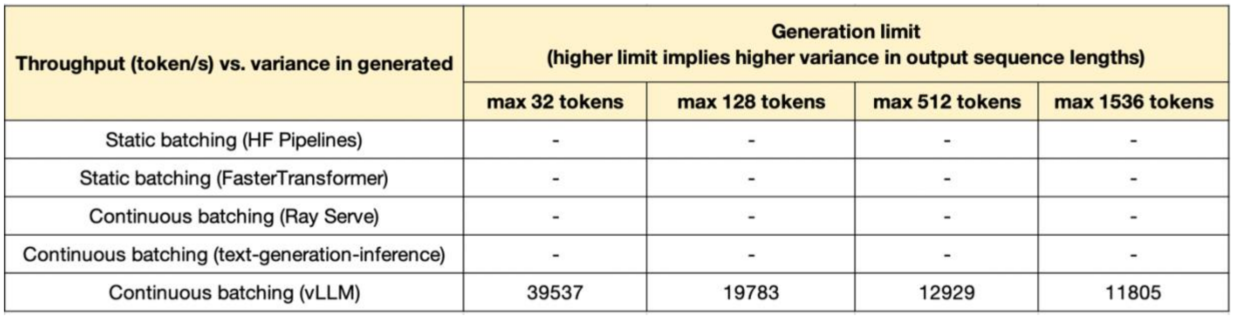
Technical Approach:

For this initial milestone, vLLM was targeted as the inference serving engine. The performance of vLLM was thus evaluated and compared against the literature [1]. All experiments were run on NVIDIA RTX 6000 GPU clusters sequestered from the University of Chicago utilizing

Chameleon cloud-based services. Instances were created utilizing images equipped with Ubuntu with CUDA version 22.04. All experiments were generated utilizing Meta's OPT-125m parameter model, compared to the 13 billion parameter model in [1]. For throughput analysis, 1000 random prompts were generated with a mean prompt length of 512 and mean variable response length of 128. In accordance with [1], exponentially sampled input lengths of 32, 128, 512, and 1536 maximum tokens were utilized to evaluate the effect of length variance on throughput. Latency was measured using 1 and 4 Queries Per Second (QPS), and all prompts were randomly generated with output lengths between 1 and 512 tokens. Total output lengths of all requests were capped at 1536 and followed a normal distribution centered at a mean of 128 tokens. All the results were produced using the source code form GitHub repository: https://github.com/anyscale/llm-continuous-batching-benchmarks which has been modified some parameters to match our experimental setup. All data and post processing scripts can be found at the following GitHub repository: https://github.com/csce585-mlsystems/Buffering

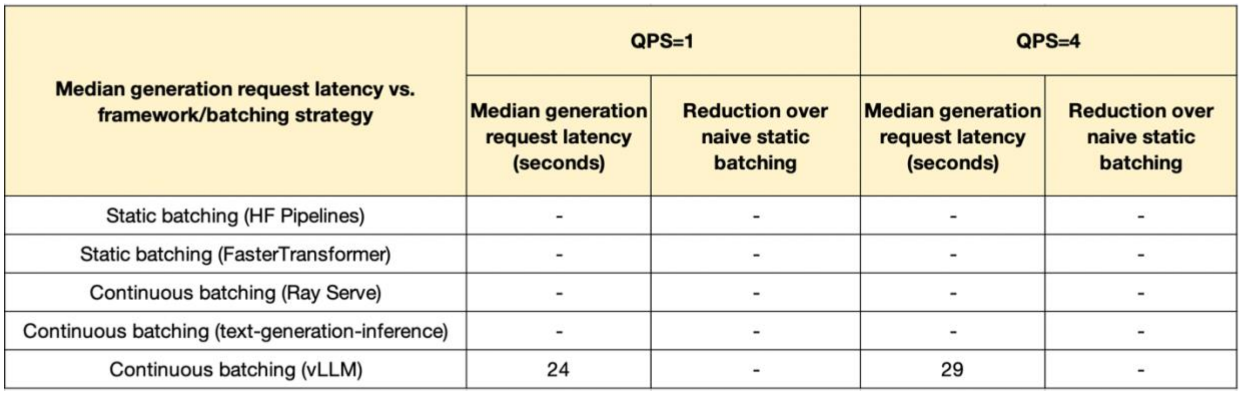
Preliminary results and Future Direction:

Table 1. Throughput in tokens per second

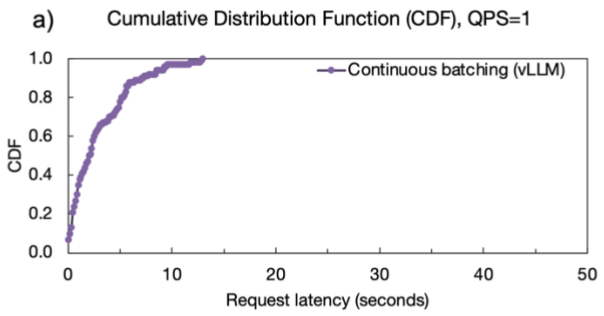


Throughput (token/s) vs. variance in generated	Generation limit (higher limit implies higher variance in output sequence lengths)			
	max 32 tokens	max 128 tokens	max 512 tokens	max 1536 tokens
Static batching (HF Pipelines)	-	-	-	-
Static batching (FasterTransformer)	-	-	-	-
Continuous batching (Ray Serve)	-	-	-	-
Continuous batching (text-generation-inference)	-	-	-	-
Continuous batching (vLLM)	39537	19783	12929	11805

Table 2. Request latency in seconds



Median generation request latency vs. framework/batching strategy	QPS=1		QPS=4	
	Median generation request latency (seconds)	Reduction over naive static batching	Median generation request latency (seconds)	Reduction over naive static batching
Static batching (HF Pipelines)	-	-	-	-
Static batching (FasterTransformer)	-	-	-	-
Continuous batching (Ray Serve)	-	-	-	-
Continuous batching (text-generation-inference)	-	-	-	-
Continuous batching (vLLM)	24	-	29	-



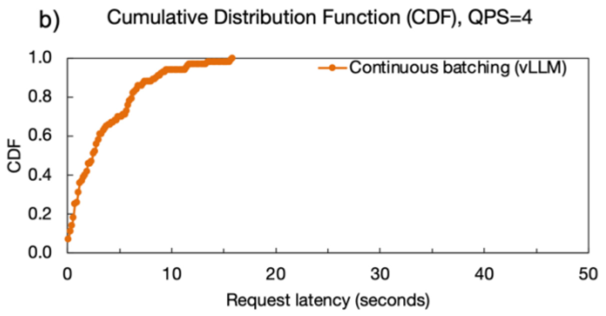


Figure. Cumulative Distribution Function (CDF) vs Request latency in seconds with a) QPS = 1 and b) QPS = 4.

Technical hurdles were present on choosing the correct platform and architecture to run both the selected model and inference serving engines. Much of the troubleshooting by the group was dedicated towards accessing the servicing clusters, creating instances, and effectively managing any errors with running benchmarking scripts. Successful trials were conducted in benchmarking the performance of vLLM. Qualitatively, the performance data trends similar to the results shown in [1]. A deeper understanding of the quantitative trends is currently in progress at this point of authorship. However, utilizing a different model (OPT-125m vs OPT-13B) could be one of the main factors contributing towards the quantitative difference in overall performance. Establishing the same for HuggingFace's Text Generation Inference is the goal of the next milestone. We will focus on setting up and deploying both the model and inference serving engine to generate experimental results to compare and contrast between those already collected.

References

1) Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J., Zhang, H., & Stoica, I. (2023). Efficient Memory Management for Large Language Model Serving with PagedAttention. *Proceedings of the 29th Symposium on Operating Systems Principles*, 611–626. https://doi.org/10.1145/3600006.3613165

2) Kilburn, T., Edwards, D. B. G., Lanigan, M. J. and Sumner, F. H. (1962). *One-Level Storage System*, 405-517.

3) Pope, R., Douglas, S., Chowdhery, A., Devlin, J., Bradbury, J., Levskaya, A., Heek, J., Xiao, K., Agrawal, S., & Dean, J. (2022). *Efficiently Scaling Transformer Inference*. http://arxiv.org/abs/2211.05102

4) Dao, T., Fu, D. Y., Ermon, S., Rudra, A., & Ré, C. (2022). *FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness*. http://arxiv.org/abs/2205.14135