ISSN: 2754-6659

Journal of Artificial Intelligence & Cloud Computing

SCIENTIFIC Research and Community

Review Article

Open Access

Optimizing LLM Deployments through Inference Backends

Ashish Bansal

USA

ABSTRACT

As model size increases, especially in the area of natural language processing, models expand past the number of parameters that can fit on a single GPU memory—requiring multiple GPUs to run inference on these models without skirting performance. But, as teams scale across more GPUs, the cost to serve inference scales, too—often outpacing what a company can afford and make these implementation profitable for organisation. Another consideration is that many LLM- based services run in real time, so low latency is a must to deliver great user experiences.

Teams today need an efficient way to serve inference from LLMs that allows infrastructure to scale across multiple GPUs—without breaking the bank. In this paper we will discuss about differ- ent llm inference backends that can be utilised to serve these llm for high performance, low cost and much robust implementation the major LLMs known today include billions of parameters. Either its ChatGPT-3 from OpenAI, Claude from Anthropic or LLama from Meta, the generative AI models are continuously increasing their parameters and this brings its own challenges to imple- ment these technologies for Real-World applications and easy to run these on devices.

As model size increases, especially in the area of natural language processing, models expand past the number of parameters that can fit on a single GPU memory—requiring multiple GPUs to run inference on these models without skirting performance. But, as teams scale across more GPUs, the cost to serve inference scales, too—often outpacing what a company can afford and make these implementation profitable for organisation. Another consideration is that many LLM- based services run in real time, so low latency is a must to deliver great user experiences

Teams today need an efficient way to serve inference from LLMs that allows infrastructure to scale across multiple GPUs—without breaking the bank. In this paper we will discuss about differ- ent llm inference backends that can be utilised to serve these llm for high performance, low cost and much robust implementation

***Corresponding author**

Ashish Bansal, USA.

Received: July 08, 2024; **Accepted:** July 15, 2024; **Published:** July 29, 2024

Keywords: Generative AI, LLM Inference, LLM Inference Backends

Introduction

As Generative AI is becoming very promise to revolutionize various companies utilizing these technologies in variety of application to boost productivity, improving client experience and lower- ing cost to serve clients. While these Large language models (LLMs) seems very promising but they come with their issue of utilizing them in real time solutions as it require nuanced performance benchmarks. But they aren't the only systems that are hard to boil down to just one number.

AI applications that produce human-like text, such as chatbots, virtual assistants, language translation, text generation, and more, are built on top of Large Language Models (LLMs).

Deploying LLMs in production-grade applications, companies faces performance challenges with running these models and consideration in optimizing your deployment with an LLM inference engine or server becomes important.

In this paper we are going to explore the different LLM inference engines and servers available to deploy and serve LLMs in production. We'll take a look at vLLM, LMDeploy, TensorRT-LLM, Triton Inference Server, RayLLM with RayServe, and HuggingFace Text Generation Inference.

Choosing the right inference backend for serving large language models (LLMs) is crucial. It not only ensures an optimal user experience with fast generation speed but also improves cost ef- ficiency through a high token generation rate and resource utilization. Today, developers have a variety of choices for inference backends created by reputable research and industry teams. How- ever, selecting the best backend for a specific use case can be challenging.This forces companies to consider multiple factors when serving inference from LLMs:

1. How to get high performance (high throughput and inference accuracy)
2. How to ensure it's cost-effective (affordable access to GPUs at scale)
3. How to create a good user experience (low latency)

These inference backends were evaluated using two key metrics:

- **Time to First Token (TTFT):** Measures the time from when a request is sent to when the first token is generated, recorded in milliseconds. TTFT is important for applications requiring immediate feedback, such as interactive chatbots. Lower latency improves perceived perfor- mance and user satisfaction.
- **Token Generation Rate:** Assesses how many tokens the model generates per second dur- ing decoding, measured in tokens per second. The token generation rate is an indicator of the model's capacity to handle high loads. A higher rate suggests that the model can ef- ficiently manage multiple requests and generate responses quickly, making it suitable for high-concurrency environments.

LLM Inference Backend Concepts

Let's deep dive into various LLM Inference Backends that are widely available and are open source make them easy to adopt and contribute making the deployment easy and customize them for variety of use cases. These depends on specific use case, the size of your model, the number of requests you need to handle, and the latency requirements of your application.

There are two main concepts while we dive into Inference Backends:

- **Inference Engines** run the models and are responsible for everything needed for the genera- tion process.

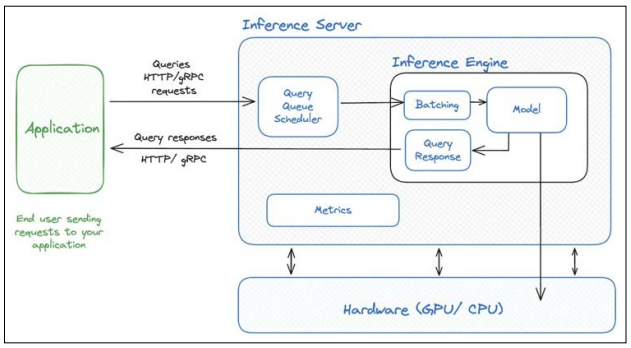

Figure 1: Inference Server and Inference Engine

- **Inference Servers** handle the incoming and outgoing HTTP and gRPC requests from end users for your application, collect metrics to measure your LLM's deployment performance, and more.

LLM Inference Engines

vLLM: an open-source library for fast LLM inference and serving. vLLM utilizes PagedAtten- tion, our new attention algorithm that effectively manages attention keys and values. vLLM equipped with PagedAttention redefines the new state of the art in LLM serving: it delivers up to 24x higher throughput than HuggingFace Transformers, without requiring any model architecture changes.

It is the core technology that makes LLM serving affordable even for a small research team like LMSYS with limited compute resources.

PagedAttention, an attention algorithm inspired by the classic idea of paging in operating systems. Unlike the traditional attention algorithms, PagedAttention allows storing continu- ous keys and values in non-contiguous memory space. Specifically, PagedAttention partitions the KV cache of each sequence into KV blocks. Each block contains the key and value vec- tors for a fixed number of tokens, which we denote as KV block size(B). During the attention computation, the PagedAttention kernel identifies and fetches different KV blocks separately. PagedAttention algorithm allows the KV blocks to be stored in non-contiguous physical mem- ory,which enables more flexible paged memory management in vLLM. vLLM offers LLM in- ferencing and serving with SOTA throughput, Paged Attention, Continuous batching, Effcient KV cache, Quantization (GPTQ, AWQ, FP8), and optimized CUDA kernels.

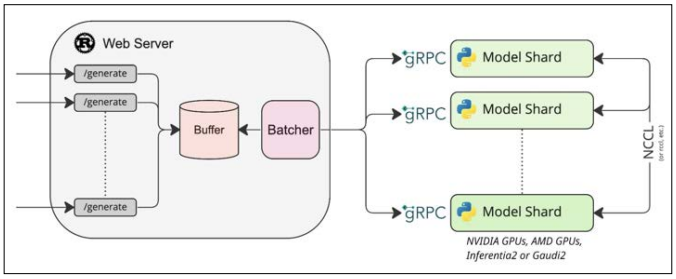

Figure 2: HuggingFace TGI Server

TensorRT-LLM: TensorRT-LLM is another inference engine that accelerates and optimizes in- ference performance for the latest LLMs on NVIDIA GPUs. LLMs are compiled into TensorRT Engine and then deployed with a triton server to leverage inference optimizations such as In-Flight Batching (reduces wait time and allows higher GPU utilization), paged KV caching, MultiGPU-MultiNode Inference, and FP8 Support. TensorRT-LLM is Created by NVIDIA,Several models are supported out of the box, including Falcon, Gemma, GPT, Llama, and more. An important limitation of TensorRT LLM is that it was built for Nvidia hardware. Also, whichever GPU you use to compile the model, you must use the same for inference. It also leverages PagedAttention, which frees up memory allocation more dynamically than traditional atten- tion approaches and In-flight batching.

Hugging Face Text Generation Inference: Text Generation Inference (TGI) is a toolkit for de- ploying and serving Large Language Models (LLMs). TGI enables high-performance text gen- eration for the most popular open-source LLMs, including Llama, Falcon, StarCoder, BLOOM, GPT-NeoX, and T5. TGI supports various hardware ranging from GPU, TPU, Gaudi to In- frentia. Hugging Face develops and distributes it under the HFOILv1.0 license, permitting commercial use provided it serves as an auxiliary tool within the product or service offered rather than the main focus. The main challenges it addresses are:

- High-performance text generation. TGI uses techniques like Tensor Parallelism (a tech- nique used to fit a large model in multiple GPUs) and dynamic batching (batching prompts together on the fly inside the server) to optimizse the performance of popular open- source LLMs, including models like StarCoder, BLOOM, GPT-NeoX, Llama, and T5.
- Effcient resource usage. Features like continuous batching, optimized code, and Ten- sor Parallelism allow TGI to handle multiple requests simultaneously while minimizing resource usage.
- Flexibility. TGI supports a variety of safety and security features like watermarking, logit warping (modifies the logits of specific tokens by infusing a bias value into them) for bias control, and stop sequences to ensure responsible and controlled LLM usage.

RayLLM with RayServe: RayLLM is an LLM serving solution for deploying AI workloads and open source LLMs with native support for continuous batching, quantization, and streaming. It provides a REST API similar to the one from OpenAI, making it easy to cross test.

RayServe is a scalable model for building online inference APIs. Ray Serve is built on top of Ray, allowing it to easily scale to many machines. Ray Serve has native support for autoscaling, multi-node deployments, and scale to zero in response to demand for your application. RayServe uses Continuous batching, quantization, and streaming with vLLM, Out-of-the-box support for vLLM and Tensor-RTLLM.

Ray minimizes the complexity of running your distributed individual and end-to-end machine learning workflows with these components:

- Scalable libraries for common machine learning tasks such as data preprocessing, dis- tributed training, hyperparameter tuning, reinforcement learning, and model serving.
- Pythonic distributed computing primitives for parallelizing and scaling Python applica- tions.
- Integrations and utilities for integrating and deploying a Ray cluster with existing tools and infrastructure such as Kubernetes, AWS, GCP, and Azure.

LMDeploy: is a toolkit for compressing, deploying, and serving LLM, developed by the MM- Razor and MMDeploy teams. It has the following core features:

- **Effcient Inference:** LMDeploy delivers up to 1.8x higher request throughput than vLLM, by introducing key features like persistent batch(a.k.a. continuous batching), blocked KV cache, dynamic splitfuse, tensor parallelism, high-performance CUDA kernels and so on.
- **Effective Quantization:** LMDeploy supports weight-only and k/v quantization, and the 4-bit inference performance is 2.4x higher than FP16. The quantization quality has been confirmed via OpenCompass evaluation.
- **Effortless Distribution Server:** Leveraging the request distribution service, LMDeploy facilitates an easy and effcient deployment of multi-model services across multiple machines and cards.
- **Interactive Inference Mode:** By caching the k/v of attention during multi-round dialogue processes, the engine remembers dialogue history, thus avoiding repetitive processing of historical sessions.
- **Excellent Compatibility:** LMDeploy supports KV Cache Quant, AWQ and Automatic Pre- fix Caching to be used simultaneously.

Triton Inference Server: enables teams to deploy any AI model from multiple deep learning and machine learning frameworks, including TensorRT, TensorFlow, PyTorch, ONNX, Open- VINO, Python, RAPIDS FIL, and more. Triton supports inference across cloud, data center, edge and embedded devices on NVIDIA GPUs, x86 and ARM CPU, or AWS Inferentia. Triton Inference Server delivers optimized performance for many query types, including real time, batched, ensembles and audio/video streaming. Triton inference Server is part of NVIDIA AI Enterprise, a software platform that accelerates the data science pipeline and streamlines the development and deployment of production AI. The following figure 3 shows the Triton In- ference Server high-level architecture. The model repository is a file-system based repository of the models that Triton will make available for inferencing. Inference requests arrive at the server via either HTTP/REST or GRPC or by the C API and are then routed to the appropriate per-model scheduler. Triton implements multiple scheduling and batching algorithms that can be configured on a model-by-model basis. Each model's scheduler optionally performs batching of inference requests and then passes the requests to the backend corresponding to the model type. The backend performs inferencing using the inputs provided in the batched requests to produce the requested outputs. The outputs are then returned.

Triton supports a backend C API that allows Triton to be extended with new functionality such as custom pre- and post-processing operations or even a new deep-learning framework.

The models being served by Triton can be queried and controlled by a dedicated model man- agement API that is available by HTTP/ REST or GRPC protocol, or by the C API.

Readiness and liveness health endpoints and utilization, throughput and latency metrics ease the integration of Triton into deployment framework such as Kubernetes.

In the next section, we will see how these backends benchmark with various scenarios.

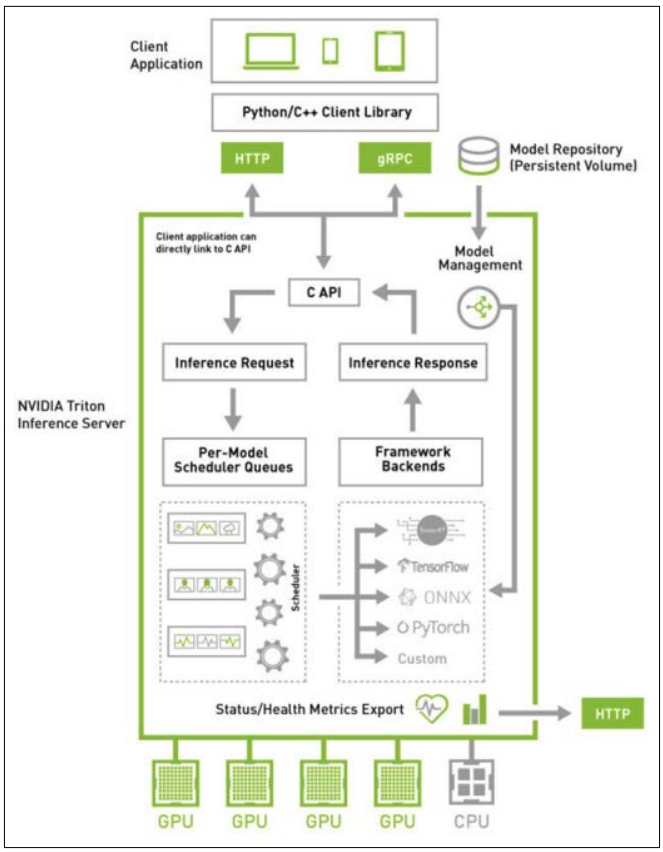

Figure 3: Triton Inference Server high-level architecture

Backend	Device	Compatibility	PEFT Adapters*	Quantization	Batching	Distributed Inference	Streaming
Transformers	GPU	High	Yes	bitsandbytes(int8/int4),AutoGPTQ(gptq), AutoAWQ(awq)	Yes	Accelerate	Yes
vLLM	GPU	High	No	awq/squeezellm	Yes	Yes	Yes
TensorRT	GPU	Medium	No		Yes	Yes	Yes
TGI	GPU	Medium	Yes	awq/eetq/gptq/bitsandbytes	Yes	Yes	Yes
LMDeploy	GPU	Medium	No	AWQ	Yes	Yes	Yes

Figure 4: Inference Backends

Benchmark
Benchmark setup:
Hardware:
GPU: 1x NVIDIA A10G 24GB
Memory: 64GB
Software:
Guest OS: Ubuntu 22.04 NVIDIA Driver Version: 536.67 CUDA Version: 12.2
PyTorch: 2.1.1
Model: meta-llama/Llama-2-7b
Prompt Length: 512 (with some random characters to avoid cache).
Max Tokens: 200

- bs: Batch Size. bs=4 indicates the batch size is 4.
- TPS: Tokens Per Second.
- QPS: Queries Per Second.
- FTL: First Token Latency, measured in milliseconds. Applicable only in stream mode.

Results are in the figure5

Backend	TPS@4	QPS@4	TPS@1	QPS@1	FTL@1
TGI	192.58	0.90	59.68	0.28	82.72
vLLM	222.63	1.08	62.69	0.30	95.43
Transformer	40.39	0.15	41.47	0.21	344.61
LMDeploy	236.03	1.15	67.86	0.33	76.81

Figure 5: Inference Backends bechmarks

Conclusion
The field of LLM inference optimization is rapidly evolving and heavily researched. The best inference backend available today might quickly be surpassed by newcomers. Based on our benchmarks and usability studies conducted at the time of writing, we will recommend to test your de- sired model for various scenarios [1-8].

Overall vLLM manages to maintain a low TTFT even as user loads increase, and its ease of use can be a significant advantage for many users.

References

1. https://lmdeploy.readthedocs.io/en/latest/installation.html.
2. https://medium.com/@zaiinn440/best-llm-inference-engine-tensorrt-vs-vllm-vs-lmdeploy-vs-mlc-llm-e8ff033d7615.
3. https://www.koyeb.com/blog/best-llm-inference-engines-and-servers-to-deploy-llms-in-production.
4. https://www.baseten.co/blog/understanding-performance-benchmarks-for-llm-inference/.
5. https://docs.nvidia.com/deeplearning/triton-inference-server/user-guide/docs/userguide/architecture.ht.
6. Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, et al. (2023) Effcient Memory Management for Large Language Model Serving with PagedAttention arXiv https://doi.org/10.48550/arXiv.2309.06180 arXiv:2309.06180.
7. https://huggingface.co/docs/text-generation-inference/en/index.
8. Ruiyang Qin, Dancheng Liu, Zheyu Yan, Zhaoxuan Tan, Zixuan Pan, et al. (2024) Empirical Guidelines for Deploying LLMs onto Resource- constrained Edge Devices arXiv https://doi.org/10.48550/arXiv.2406.03777.