

# Large Language Model (LLM) Inference

This document does not contain any confidential information. All information in this document is from public sources.

## Applications, Optimization Techniques, and Libraries

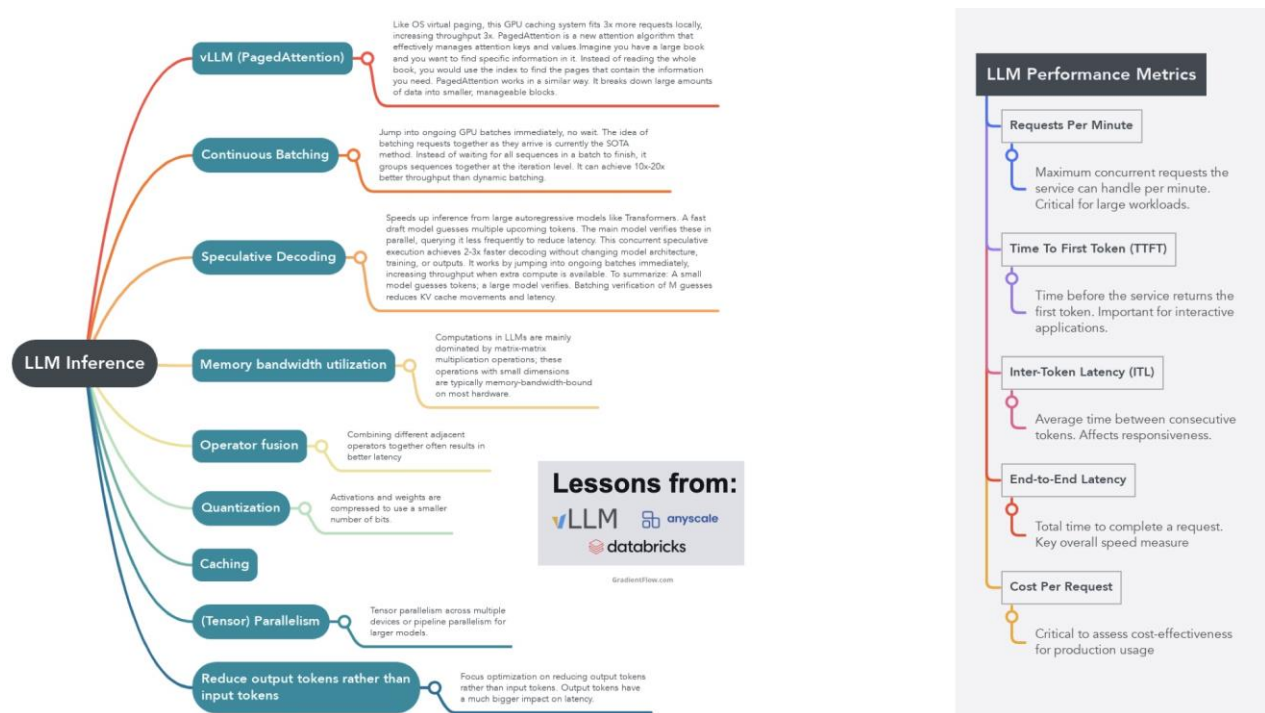
### Key Takeaways

1. Inference Optimization will become a critical necessity for enterprises and ISVs deploying language models as they grapple with the price-performance conundrum.
2. Key inference optimization techniques such as Pruning, Quantization, and Distillation to reduce the size and computational load will need to be complemented by emerging techniques such as [Speculative Decoding](#).
3. Hardware Acceleration utilizing GPUs and TPUs significantly accelerates model inference, but is a costly proposition for most.
4. [Commodity CPU Based Inference](#) is emerging as an alternative for some use cases using Sparsity as effective approach.
5. Key Performance metrics for inference: Requests per minute, Time to first token, Inter-token latency, End-to-end latency and Cost per request.

### Context

Large Language Model (LLM) inference refers to the process of using a pre-trained language model to generate predictions or insights on new input data. These models, often based on transformer architectures, have gained significant attention due to their remarkable performance in natural language understanding and generation tasks. LLMs, such as Llama and OpenAI's GPT series, have been applied across a spectrum of applications, and optimizing their inference process is crucial for achieving real-time and resource-efficient performance.

LLMs promise to fundamentally change how we use AI across all industries. However, serving these models is challenging and can be surprisingly slow even on expensive hardware.



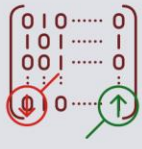


Source: [LLM Inference Hardware: Emerging from Nvidia's Shadow \(substack.com\)](https://substack.com/p/llm-inference-hardware-emerging-from-nvidia-s-shadow)

## Applications of LLM Inference:

- **Natural Language Processing (NLP):** LLMs excel in various NLP applications, including text summarization, sentiment analysis, and language translation.
- **Content Generation:** LLMs are utilized for generating human-like text content, such as articles, poetry, and dialogue.
- **Chatbots and Virtual Assistants:** LLMs serve as the backbone for conversational AI, enabling chatbots and virtual assistants to process user queries.
- **Code Generation:** LLMs are employed for generating code snippets and aiding in programming-related tasks.

## Top Inference Optimization Techniques:

- **Pruning:** Removing unnecessary connections or neurons from the model to reduce computational complexity and enhance inference speed.
- **Model Compression:** Techniques like knowledge distillation and model distillation aim to transfer the knowledge of a larger model to a smaller, more efficient one.
- **Hardware Acceleration:** Leveraging specialized hardware, such as GPUs and TPUs, can significantly boost inference speed.
- **Sparsity:** Reduce computational complexity by removing less important connections.
- **Quantization:** Reduce the precision of model weights and activations to save memory and increase speed.

Optimizing Neural Networks		
<b>Sparsity</b>		<p>Sparsity in neural networks involves having a large number of zero or near-zero elements in the network's weights, reducing computational complexity and memory usage. Sparsity is achieved through pruning, a process that removes less important connections, making the network more efficient without compromising accuracy. Sparsity is particularly useful in Large Language Models (LLMs) for efficient execution on CPUs, allowing rapid processing on widely-available hardware without relying on specialized GPU resources. Introducing sparsity is a key optimization for deploying powerful AI models more broadly and cost-effectively across devices. Overall, neural network sparsity enables leaner, faster, and more accessible AI through selective connectivity and efficient use of resources.</p>
<b>Pruning</b>		<p>Pruning in neural networks involves selectively removing less important neuron connections to reduce the network's size and complexity without significantly impacting its performance. This process, typically done post-training, involves removing connections with minimal weight contributions, thus reducing the model's memory and computational demands. Ideal for deploying on devices with limited resources, pruning results in a more compact and efficient model. The method enhances neural network sparsity, striking a balance between maintaining accuracy and reducing model density.</p>
<b>Quantization</b>		<p>Quantization in neural networks reduces the precision of numbers representing model weights and activations, usually by converting from high- to lower-bit formats. This shrinks model size and quickens inference speed, beneficial for hardware with limited resources or requiring real-time performance. In LLMs for CPU inference, quantization is crucial as it compresses models and accelerates inference by minimizing data movement and computational demands. This makes LLMs more efficient for deployment on CPUs, especially in resource-constrained environments.</p>

Source: [LLM Inference Hardware: Emerging from Nvidia's Shadow \(substack.com\)](https://substack.com/p/llm-inference-hardware-emerging-from-nvidia-s-shadow)

## Libraries for LLM Inference Optimization

### Top Libraries

- VLLM
- TensorRT
- DeepSpeed
- ColossalAI
- FairScale
- TensorMesh

## Text Generation Inference

Text Generation Inference (TGI) is a toolkit for deploying and serving Large Language Models (LLMs). TGI enables high-performance text generation for the most popular open-source LLMs, including Llama, Falcon, StarCoder, BLOOM, GPT-NeoX, and T5.

## DeepSpeed

DeepSpeed is a deep learning optimization library developed by Microsoft, specifically designed to facilitate the training of large-scale machine learning models. What sets DeepSpeed apart is its innovative approach to model parallelism, particularly through its ZeRO (Zero Redundancy Optimizer) technology.

## Colossal-AI

While the primary focus of Colossal-AI is on the training phase, its capability to handle diverse forms of parallelism also extends to inference optimization. By efficiently utilizing hardware resources and distributing computational loads, Colossal-AI can significantly speed up the inference process, especially for large-scale models.

## FairScale

FairScale is a PyTorch-based library that provides a suite of advanced tools for optimizing the training of deep learning models across distributed systems. It offers features like model parallelism and sharded data parallelism, which are key in reducing the memory footprint and accelerating the training process of large neural networks.

When comparing DeepSpeed, Colossal-AI, FairScale, and TensorFlow Mesh, a few key distinctions become apparent. DeepSpeed and FairScale are both heavily focused on optimizing memory usage and computational efficiency, making them ideal for training very large models on limited hardware resources. TGI is best for large text-generation models. DeepSpeed, with its ZeRO technology, pushes the boundaries in terms of the size of models that can be trained, while FairScale offers a range of advanced training techniques to improve training efficiency.

Colossal-AI, on the other hand, emphasizes flexibility in parallel training techniques, supporting various forms of parallelism. This makes it a versatile choice for training large models across different types of hardware setups. TensorFlow Mesh integrates closely with TensorFlow's ecosystem and focuses on simplifying the distributed training process, especially across Google's TPUs, which can be a significant advantage for users already embedded in the TensorFlow environment. Source: [Inference Optimization Strategies for Large Language Models: Current Trends and Future Outlook \(ankursnewsletter.com\)](https://ankursnewsletter.com/inference-optimization-strategies-for-large-language-models-current-trends-and-future-outlook)

## VLLM

vLLM is an open source, fast and easy-to-use library for LLM inference and serving. Based on a paper written by researchers at UC Berkeley, Stanford and UC San Diego ([Efficient Memory Management for Large Language Model Serving with PagedAttention](https://arxiv.org/abs/2309.03409)), vLLM equipped with PagedAttention delivers up to 24x higher throughput than HuggingFace Transformers, without requiring any model architecture changes.

If you use the PyTorch API to build and run a model on GPUs, 60-60% of the GPU memory is wasted by the [Attention layer](#). The KV Cache consumes a lot of the GPU memory. VLLM's secret sauce is the PagedAttention algorithm, inspired by Virtual Memory and Paging techniques in Operating

Systems. Using PagedAttention, vLLM creates a 24x performance improvement compared to regular Hugging Face API and 3.5x over the Hugging Face TGI library.

VLLM consists of

1. The secret sauce – the PagedAttention algorithm; Without PagedAttention, 60-60% of GPU is fragmented and wasted because Values of the KV Cache are of variable length. Long values demand continuous memory and fragmentation occurs because of over allocation for values of variable length. With PagedAttention, 90-95% of GPU memory is utilized.
2. FastAPI server – a python library for backend apps.

## TensorRT

TensorRT is an SDK (Software Development Kit) for high performance deep learning inference. Powered by Nvidia's [CUDA \(Compute Unified Device Architecture\)](#) framework for high-performance GPU accelerated applications, TensorRT can be used on the edge or in the data center. It supports all frameworks and is an integral part of the TensorFlow ecosystem that focuses on optimizing the runtime performance of TensorFlow models, including LLMs.

The open-source library — which was not ready in time for August submission to MLPerf — enables customers to more than double the inference performance of their already purchased H100 GPUs at no added cost.

## Tensor RT8 Optimization techniques

1. Quantization Aware Tuning (QAT)
2. Sparsity Support for Ampere GPUs (e.g. A100)

### *Quantization Aware Tuning (QAT)*

Quantization is the process of reducing the precision of model weights and activations to reduce memory requirements and increase the speed of inference. Quantization converts a floating-point matrix to an integer matrix. Most models are trained with 32 or 16 bits of precision, where each parameter and activation element takes up 32 or 16 bits of memory—a single-precision floating point. However, most deep learning models can be effectively represented with eight or even fewer bits per value. TensorRT's QAT technique makes it easier for a model to adapt to quantization without compromising accuracy. It does so via post training quantization i.e. quantization after training is complete. The trained model is retrained with quantization to achieve FP32 (Floating Point – 32 bit) precision using INT8 (Integer – 8 bit) precision.

### *Sparsity Support for Ampere GPUs (e.g. A100)*

Nvidia's Ampere GPUs have a 'sparse kernel' i.e. a significant portion of the model does not need to be computed because the sparse kernel generates sparse matrices – matrices that are comprised mostly of zero values. It generates a sparse matrix by zeroing out low values while capturing the locations of

values that were zeroed out. This significantly boosts the GPU's performance because the GPU can ignore many calculations, just as we did as students in school whenever we saw numbers that needed to be multiplied by zero.

TensorRT also supports:

- Reduced mixed precision – to optimize throughput
- Layer and tensor fusion – to optimize GPU memory
- Kernel autotuning – to select the best data layer and algorithms
- Time fusion – to optimize [Recurrent Neural Networks](#) (RNN)
- Multi-stream execution – to manage input streaming
- Dynamic tensor memory – to optimize memory consumption.

## TensorRT's Optimization sequence

1. Step 1: Export model to the [ONNX](#) runtime
2. Step 2: TFRT's ONNX parser parses it and feeds it to the Builder
3. Step 3: The Builder builds an engine file, a static graph optimized for inference computation.
4. Step 4: The engine file is used for inference and needs an execution context which stores intermediate execution values.
5. Step 5: A CUDA stream is opened to copy CPU memory to GPU memory
6. Step 6: And finally, the inference output is copied to CPU memory for post processing.

## Torch-TensorRT and TensorFlow-TensorRT

Tensor RT is available in two flavors

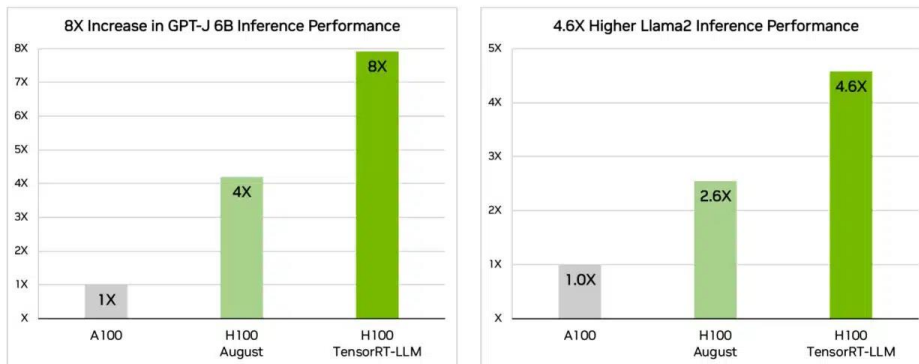
1. Torch TensorRT
2. TensorFlow Tensor RT

Torch-TensorRT brings together PyTorch and the inference optimizations of TensorRT on Nvidia GPUs. A single line of code generates an accelerated model with 6x performance gains over PyTorch. TorchScript is an intermediate representation of a PyTorch model that can then be run in a high-performance environment like C++. It's a high-performance subset of Python that is meant to be consumed by the PyTorch JIT Compiler, which performs run-time optimization on a model's computation. Torch-TensorRT automatically generates TorchScript and applies TensorRT optimizations on the intermediate representation and produces an optimized TorchScript model that can be used in the PyTorch ecosystem.

TensorFlow-TensorRT provides optimizations for the TensorFlow ecosystem. The TensorFlow-TensorRT API is part of the TensorFlow library so one can simply use a TensorFlow container, startup a Jupyter notebook and generate an optimized model with 6X-10X performance gains.

## TensorRT-LLM Supercharges Hopper Performance

Software optimizations double leading performance



Text summarization, variable input/output length, CNN / DailyMail dataset  
A100 FP16 PyTorch eager mode | H100 FP8 | H100 FP8, TensorRT-LLM, in-flight batching

Source: [NVIDIA Grace Hopper Superchip Sweeps MLPerf Inference Benchmarks | NVIDIA Blogs](#)

When comparing DeepSpeed, Colossal-AI, FairScale, and TensorFlow Mesh, a few key distinctions become apparent. DeepSpeed and FairScale are both heavily focused on optimizing memory usage and computational efficiency, making them ideal for training very large models on limited hardware resources. TGI is best for large text-generation models. DeepSpeed, with its ZeRO technology, pushes the boundaries in terms of the size of models that can be trained, while FairScale offers a range of advanced training techniques to improve training efficiency. Colossal-AI, on the other hand, emphasizes flexibility in parallel training techniques, supporting various forms of parallelism. This makes it a versatile choice for training large models across different types of hardware setups. TensorFlow Mesh integrates closely with TensorFlow's ecosystem and focuses on simplifying the distributed training process, especially across Google's TPUs, which can be a significant advantage for users already embedded in the TensorFlow environment.

## Nvidia GPU Inference Performance Benchmarks

	Nvidia H200 (Inferences/Second)	Nvidia H100 (Inferences/Second)
GPT-J (Large Language Model)	13.34	13.29
DLRMv2 (Recommender)	49,002	42,856

<b>BERT (Natural Language Processing)**</b>	8,646	7,878
<b>ResNet-50 v1.5 (Image Classification)</b>	93,198	88,526
<b>RetinaNet (Object Detection)</b>	1,849	1,761
<b>RNN-T (Speech Recognition)</b>	25,975	23,307
<b>3D U-Net (Medical Imaging)</b>	6.8	6.5

Source: [MLPerf AI Benchmarks | NVIDIA](#)