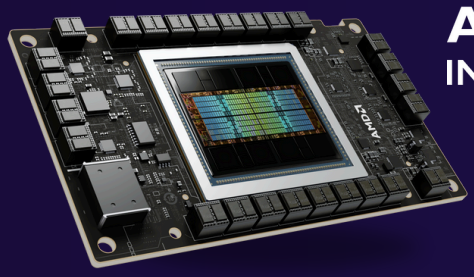
Putting AMD Instinct™ MI300X Accelerators to the Test on Dell PowerEdge™ XE9680 Rack Server With LORA Fine-tuning and vLLM Model Serving

| August 2024



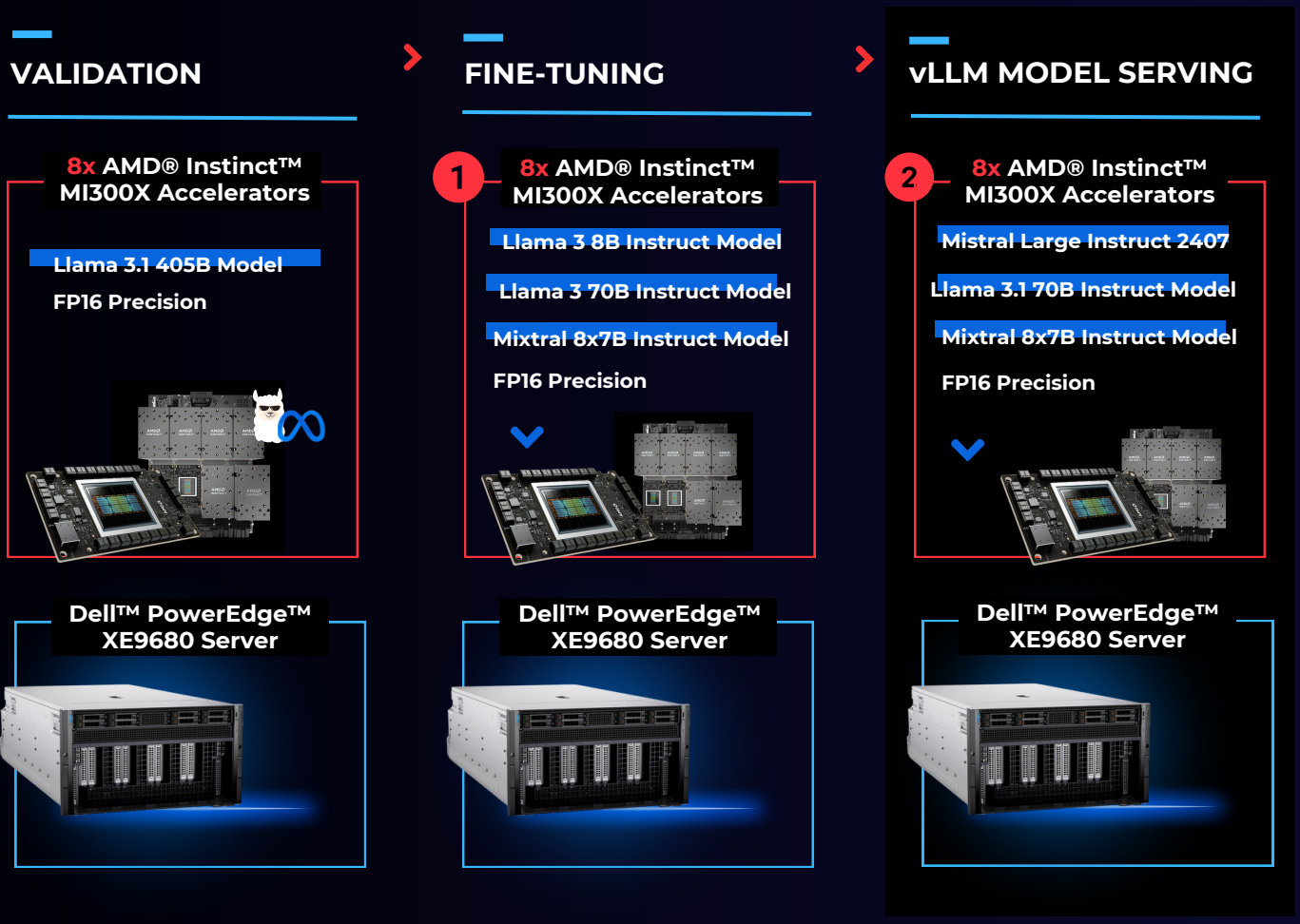
AMD INSTINCT

DELL Technologies AMD METRUM AI Hugging Face

The release of the AMD Instinct MI300X accelerator marks a significant milestone in the AI hardware landscape, providing an alternative for powering today's generative AI solutions. Dell has integrated eight AMD Instinct MI300X accelerators into their flagship PowerEdge XE9680 server designed for high performance AI applications. To evaluate this combination, Metrum AI was granted early access and conducted real-world performance tests in fine-tuning and model serving across industry leading open weight models, revealing the following total cost of ownership advantages:

- Ability to deploy Llama 3.1 405B FP16 Model on a single Dell PowerEdge XE9680 server.
- Ability to fine-tune large language models with higher batch sizes on a single Dell PowerEdge XE9680 server reducing overall training time.
- Ability to deliver leadership scalability on Mixture of Experts models for concurrent users or agents.

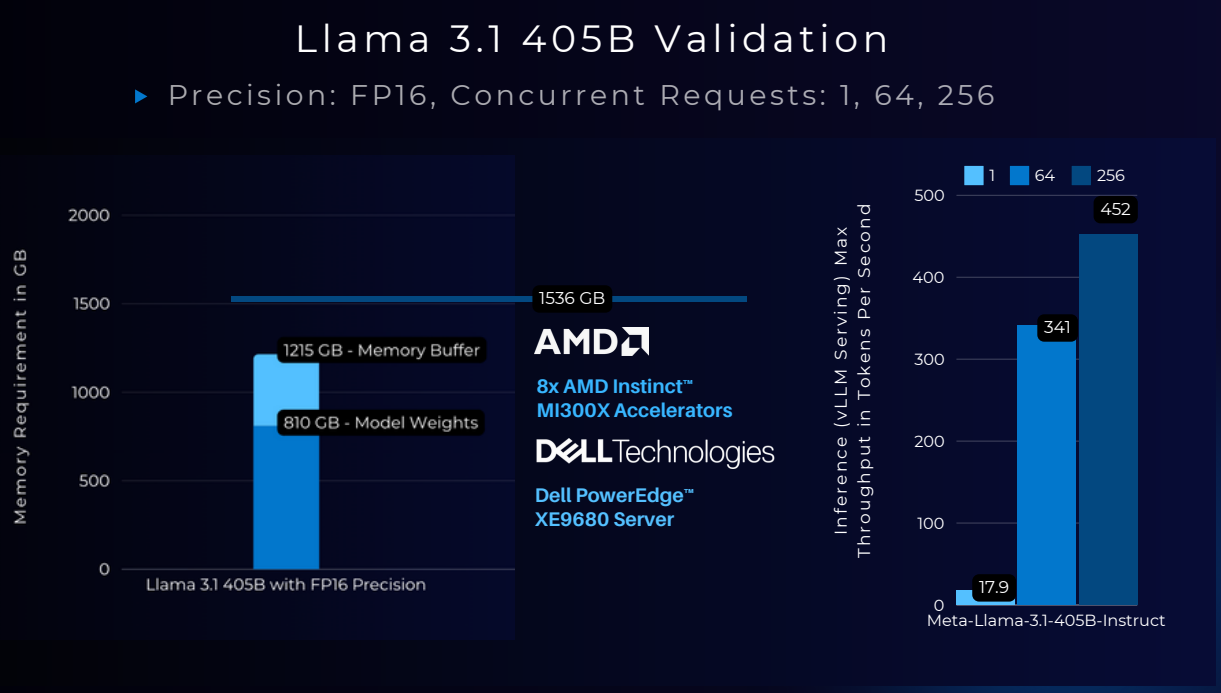
In this blog, we'll demonstrate the deployment of Llama 3.1 405B at FP16 precision on a single Dell PowerEdge XE9680 server with eight AMD Instinct MI300X accelerators. We'll also introduce our performance testing methodology and results for fine-tuning and vLLM model serving.



"Dell PowerEdge XE9680 server paired with AMD Instinct MI300X accelerators delivers industry leading TCO for single node fine-tuning and deployment of state of the art LLMs such as Llama 3.1 405B at FP16 precision.".

- Chetan Gadgil, CTO, Scalers AI

Part 1.0: Deploying Llama 3.1 405B



Metrum AI deployed Llama 3.1 405B with FP16 precision on vLLM using a single Dell PowerEdge XE9680 Server equipped with eight AMD Instinct MI300X accelerators. The high memory capacity and bandwidth of the server made it possible to handle the model's 405 billion parameters, totaling 810 GB of memory. As illustrated in the chart above, total token throughput scales effectively with the number of concurrent requests - an essential factor in real world scenarios for enterprises implementing API & agent-based workflows or batch applications.

Now that we have demonstrated the ability to deploy a massive open-weight LLM, we'll showcase how you can fine-tune pretrained leading open-weight LLMs using a domain specific dataset.

Part 2.0: Fine Tuning

Why Fine-Tuning is Important for Enterprise AI

Fine-tuning is essential for enterprise AI because it customizes pre-trained models to meet specific industry needs, improving their performance on specialized tasks. By training models on domain-specific data, fine-tuning improves accuracy and relevance towards specific enterprise functions, resulting in more coherent and context-aware outputs. This approach is also cost-effective, as it requires less data and computational resources compared to training models from scratch.

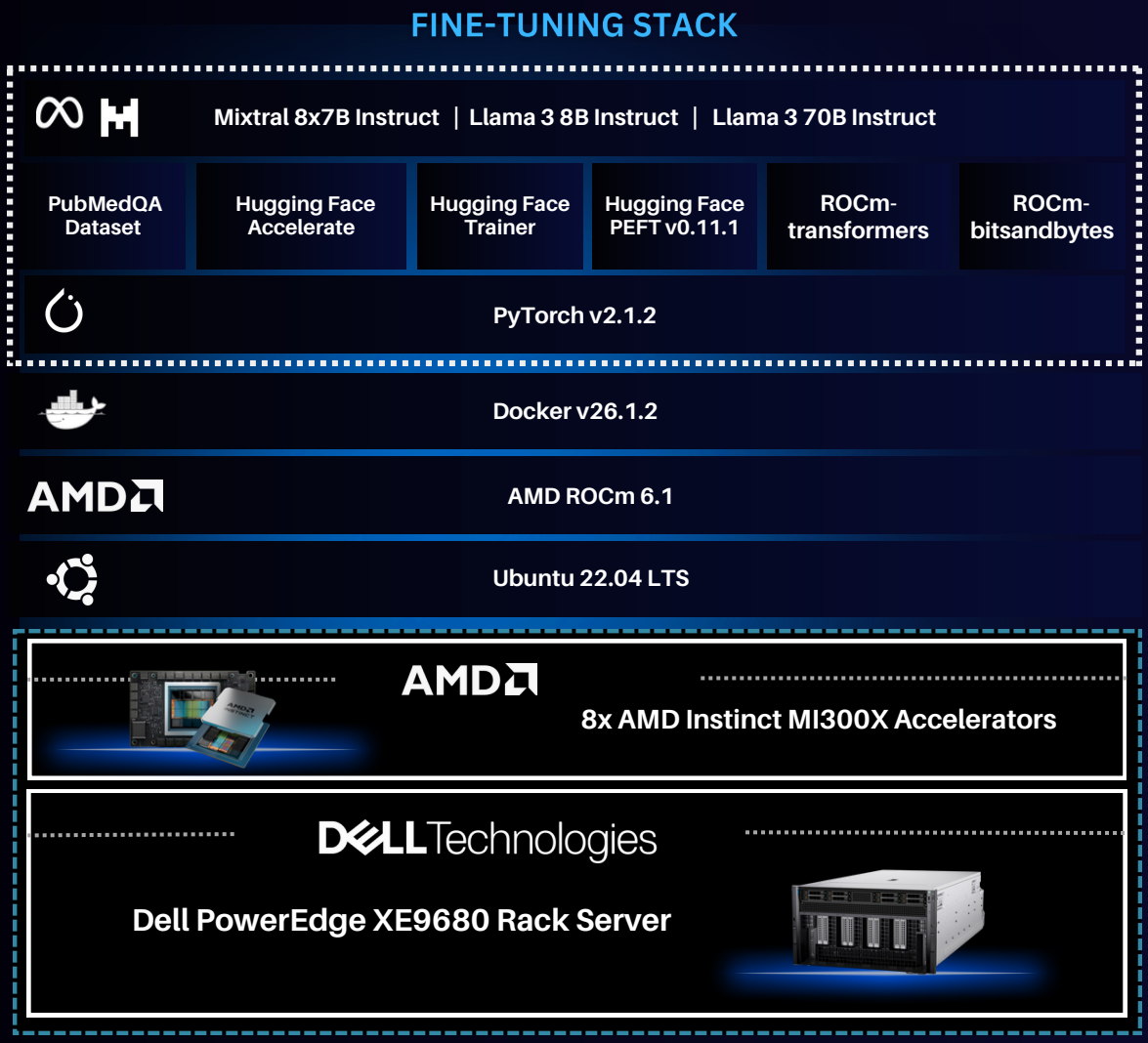
Methodology Overview

To best reflect the experience enterprises would have in fine-tuning, we selected an industry specific data set, leveraged several leading open-weight models, and conducted tests using industry leading fine-tuning tools.

We specifically use the pqa-labeled subset of the pubmed_qa dataset, containing 1000 data samples with ~516k total tokens (estimated using the Llama 3 tokenizer), for fine-tuning the following models: Mixtral 8x7B Instruct v0.1, Llama-3-8B-Instruct, Llama-3-70B-Instruct. PubMedQA was selected as a domain specific dataset that reflects enterprise fine-tuning workloads. The models Mixtral 8x7B Instruct, Llama 3 8B Instruct, and Llama 3 70B Instruct were selected because they are leading open-weight models, well positioned for enterprise fine-tuning due to their performance, quality, and commercial friendly licensing. We conducted testing for batch sizes 1, 16, 64 and 128 using FP16 model precision. Each scenario included 200 steps with a maximum length of tokens of 512, preceded by a warm-up phase of 10 steps. Warm-up steps are excluded from metrics collection.

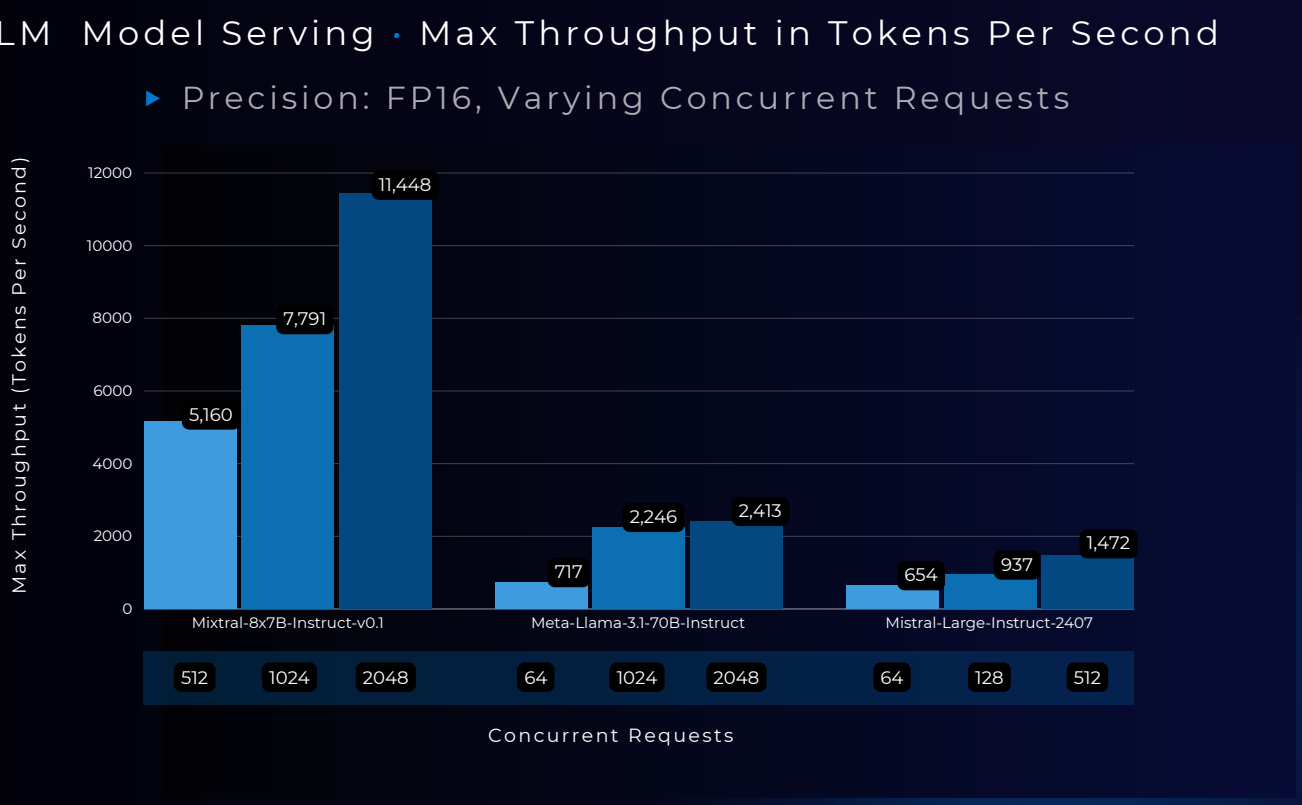
The following fine-tuning stack was used to conduct the tests, leveraging the Hugging Face Transformers library in combination with Hugging Face Accelerate. We selected Hugging Face Accelerate for its ease of use and widespread developer adoption. This stack uses AMD ROCm™ 6.1, AMD's open-source software stack designed for GPU computation. AMD ROCm 6.1 enables seamless support of PyTorch v2.1.2, Hugging Face Accelerate, and leading state of the art open-weight models.

Architecture Overview



Performance Results

To measure performance we collected train tokens per second, which is defined as the number of tokens a model processes per second during fine-tuning. Train tokens per second is particularly valuable for estimating training time and resource requirements in large-scale fine-tuning projects.



The chart above illustrates significant improvements in fine-tuning token throughput with increased batch sizes on the Mixtral 8x7B Instruct model. For instance, using a batch size of 16 instead of 1, significantly boosts token throughput, as shown in the chart above, increasing tokens per second by over 6x. Even greater gains in larger tokens per second values are observed for batch sizes 64 and 128.

Similarly, we observed significant improvements in token throughput with increased batch size on the Llama 3 8B and 70B Instruct models. In these scenarios, using a batch size of 16 instead of 1 boosted token throughput by over 2.5x.

Dell PowerEdge XE9680 server, with eight AMD Instinct MI300X accelerators, excels at fine-tuning large language models (LLMs) with larger batch sizes due to its sizable memory footprint, of 192 GB per accelerator. This larger memory capacity allows for greater batch sizes, which not only boost training throughput by processing more data per iteration, but also enable faster convergence. As a result, the number of epochs needed for optimal performance is reduced, making fine-tuning faster and resource efficient - an essential factor for optimizing total cost of ownership (TCO) in enterprise environments.

Now that we have demonstrated the ability to deploy fine-tune a pretrained model for a specific domain, we'll showcase how you can deploy or serve this model in validation or product environments.

Part 3.0: vLLM Model Serving

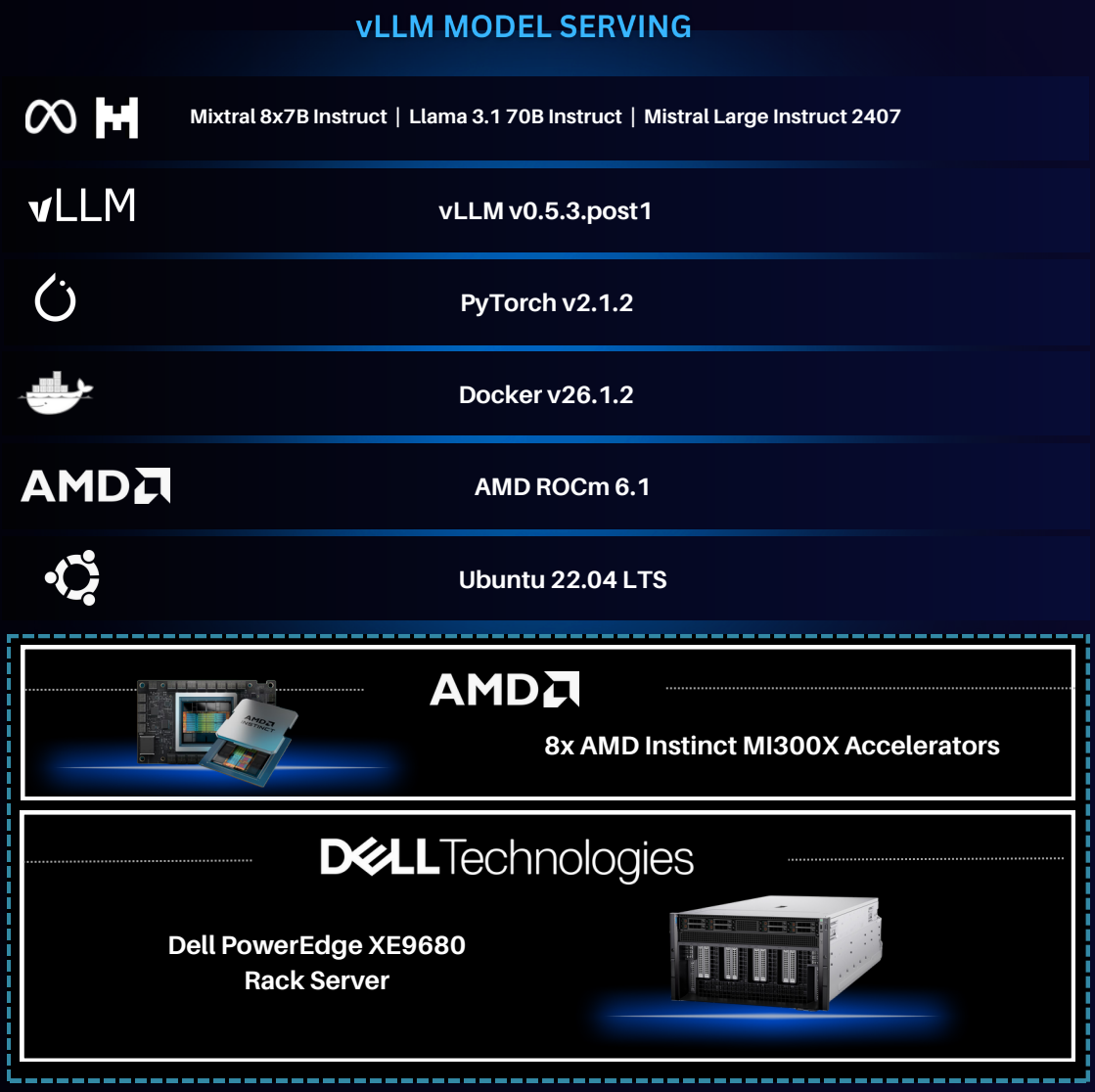
Methodology Overview

To deploy the model, we use vLLM, an industry-leading model serving solution, to reflect the common approach enterprises take to deploy AI. vLLM (Virtual Large Language Model) is an open-source library designed to optimize the deployment (inference and serving) of large language models (LLMs), which addresses the challenges of high computational demands and inefficient memory management typically associated with deploying LLMs in real-world, client-server applications. To achieve this, vLLM implements a number of optimizations including dynamic batching.

We deployed the following models using vLLM with FP16 precision: Mistral-Large-Instruct-2407, Llama-3.1-70B-Instruct, Mixtral 8x7B Instruct v0.1 and conducted performance testing by varying the number of concurrent requests using Apache Bench with the following values: 64, 128, 512, 1024 & 2048. We also employed a prompt randomizer component which substitutes a random prompt from a pool of "k" prompts, configurable to millions of randomized prompts from a given pool, to simulate real-world concurrent user activity. Here, we use an average input prompt length of 32 tokens and a maximum length of output tokens of 256.

The methodology used leverages a high-performance load balancer to simulate a single virtualized service, an architecture that can elastically respond to high demand. This approach simulates supporting multiple concurrent users or agents simultaneously within typical client server applications.

The image below illustrates in detail the architecture used for common LLM deployment scenarios. In this setup, we utilize the vLLM v0.5.3.post1 inference server. The AMD ROCm 6.1.1 driver is installed on the host vLLM v0.5.3.post1 inference server, and the vLLM ROCm docker image supported on AMD ROCm 6.1.1 is built using vLLM.



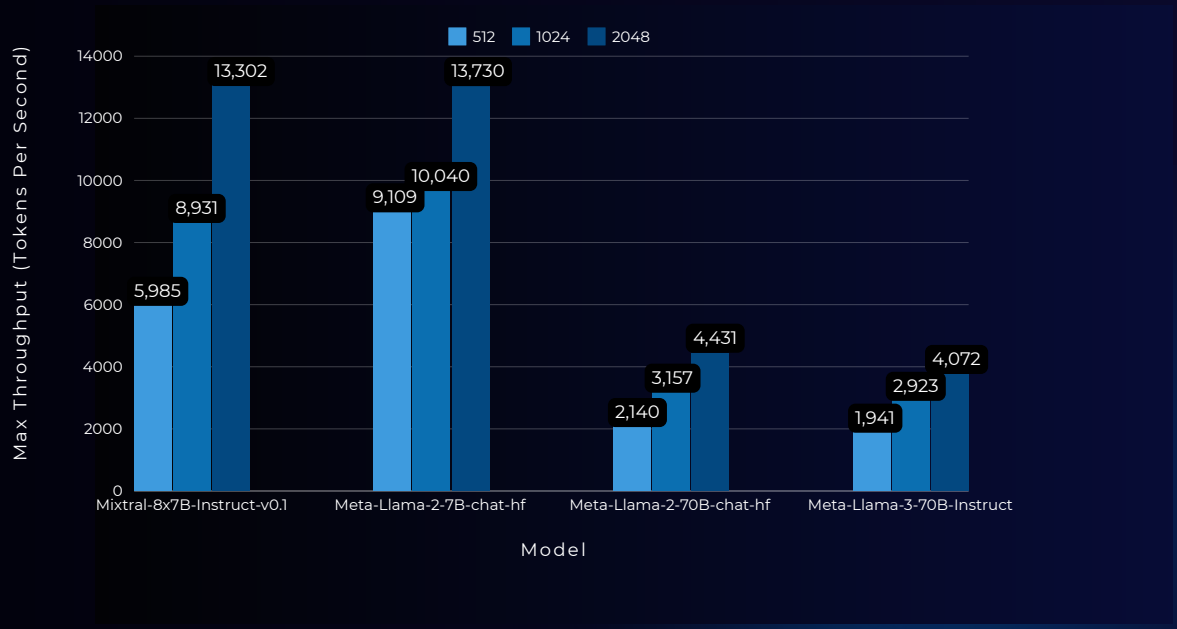
Note: These tests were performed using vLLM release v0.5.3.post1 which was the latest version available when the tests were initiated.

Performance Results

We measured performance by collecting throughput in tokens per second, a widely-used metric used to evaluate how quickly a deployed model can respond to requests and generate outputs in real-world applications.

Inference (vLLM Serving) · Max Throughput in Tokens Per Second

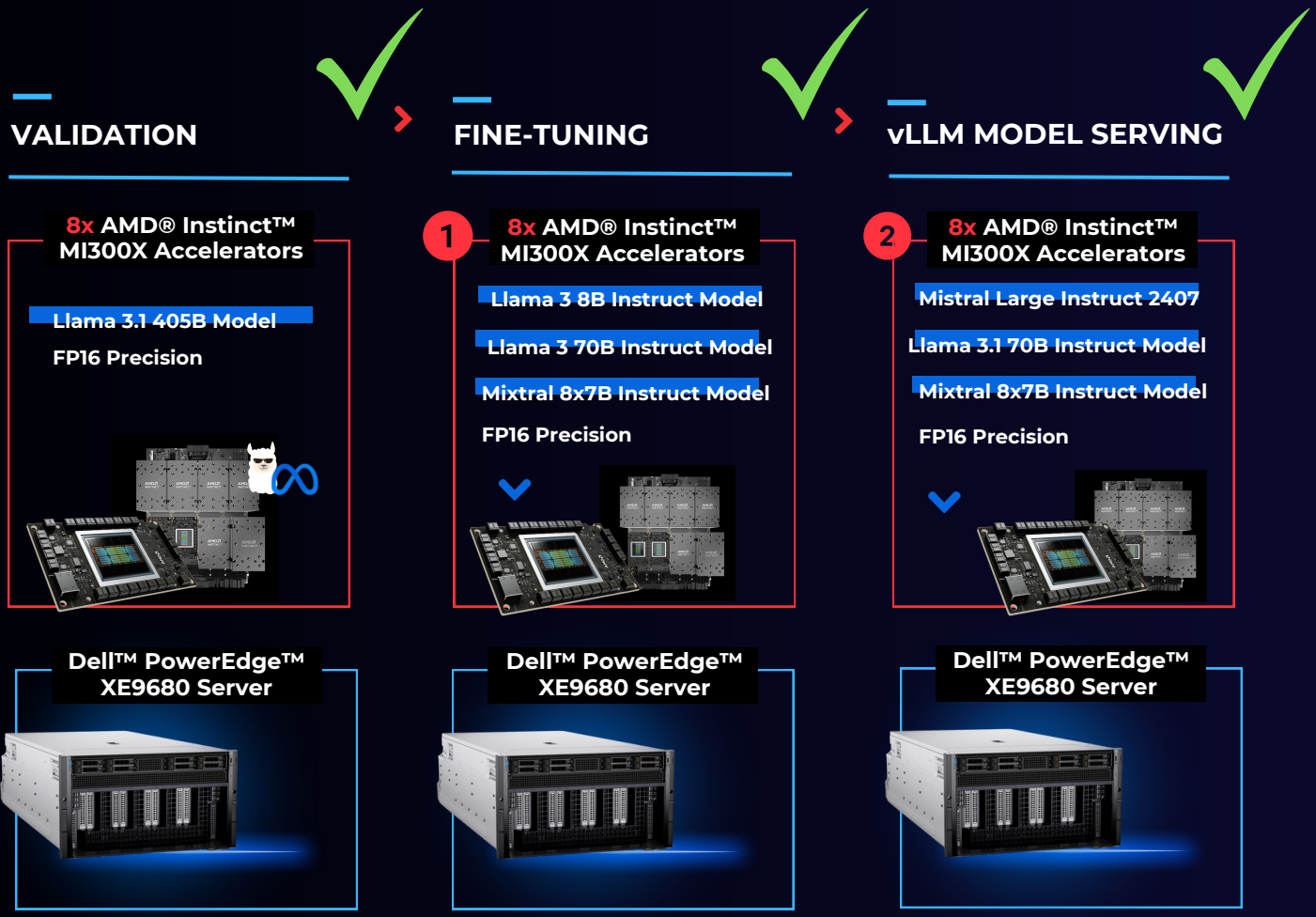
▸ Precision: FP16, Concurrent Requests: 512,1024,2048



This chart demonstrates how throughput scales with the number of concurrent requests. For enterprises looking to deploy workflows based on agents or serve batch-based applications, achieving high token throughput at high concurrency is crucial. It ensures these workflows can efficiently handle large volumes of input data while maintaining high reliability, parallelism and performance - factors that are often more critical than basic token latency.

Summary

We measured performance by collecting throughput in tokens per second, a widely-used metric used to evaluate how quickly a deployed model can respond to requests and generate outputs in real-world applications.



Dell PowerEdge XE9680 server with AMD Instinct MI300X accelerators provides a powerful platform for fine-tuning and vLLM model serving, meeting the needs of enterprises looking to develop and deploy state of the art generative AI solutions.

Benchmarking Methodology

In this section, we provide a more detailed overview of our methodology. If you'd like to replicate the results presented in this blog, reach out to us at contact@metrum.ai.

Start with Dell PowerEdge XE9680 Server configurations as such.

OS: Ubuntu 22.04.4 LTS
Kernel version: 5.15.0-94-generic
Docker Version: Docker Version 26.1.2
AMD ROCm version: 6.1.1
Server: Dell PowerEdge XE9680 Server
GPU: 8x AMD Instinct MI300X Accelerators

Fine-tuning

For those interested in replicating a similar performance testing of LoRA fine-tuning on AMD GPUs, AMD provides comprehensive documentation and a detailed guide to help you get started. For a step-by-step walkthrough to reproduce LoRA fine-tuning, refer to this AMD ROCm blog.

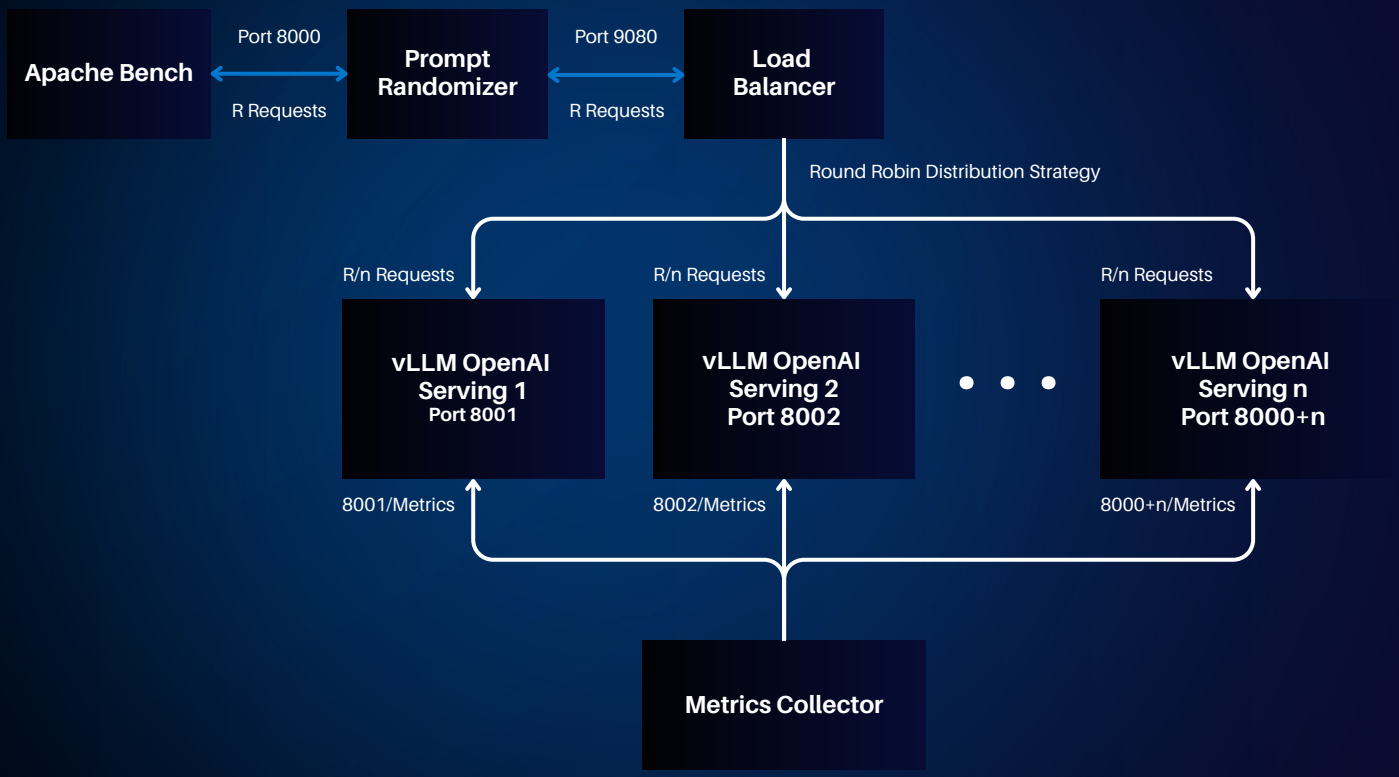
AMD provides a ready-to-use script for fine-tuning, available in their GitHub repository.

vLLM Model Serving

Our performance benchmarking architecture for a vLLM (v0.5.3.post1) model serving scenario on multiple GPUs is built around four major components:

1. Apache Bench
2. A Prompt Randomizer
3. A Load Balancer
4. Multiple vLLM Serving Replicas

See next page for diagram.



The load balancer (e.g. nginx, haproxy, traefik or similar) handles load balancing between the input requests from Apache Bench and the multiple vLLM serving replicas deployed on the server. Each vLLM serving replica will be configured to load the model on multiple GPUs given the required tensor parallelism. The count of replicas will fall in the range of values (1, 2, 4, 8) and will be based on the following factors:

1. Number of LLM Model Parameters
2. Inference Precision
3. GPU Memory Capability

Through a prompt randomizer, Apache Bench will access the load balancer at port 9080. The prompt randomizer will randomly select prompts from the list and the load balancer will then distribute the request among the deployed vLLM serving replicas using a round-robin strategy.

Each vLLM serving replica will expose ports ranging from 8001 to 8000+n, where n is the number of vLLM serving replicas deployed. These ports can be accessed by the Metrics Collector module to retrieve vLLM production metrics via the */metrics* endpoint.

Running vLLM Server

vLLM offers a comprehensive guide for running vLLMs on AMD GPUs. For detailed instructions and setup information, please refer to the vLLM documentation.

Resources

Dell product image: Dell.com, https://www.dell.com/
AMD product image: AMD library, https://www.amd.com/en/partner/resources/resource-library.html

***DISCLAIMER - Performance varies by hardware and software configurations, including testing conditions, system settings, application complexity, the quantity of data, batch sizes, software versions, libraries used, and other factors. The results of performance testing provided are intended for informational purposes only and should not be considered as a guarantee of actual performance.