Generative AI in the Enterprise – Model Customization

A Scalable and Modular Production Infrastructure with NVIDIA for AI Large Language Model Customization

October 2023

H19825

Design Guide

Abstract

This design guide describes the architecture and design of the Dell Validated Design for Generative AI Model Customization with NVIDIA, a collaboration between Dell Technologies and NVIDIA to enable high performance, scalable, and modular full-stack generative AI model customization solutions for large language models in the enterprise.

Dell Generative AI Solutions

Dell
Validated Design

DELL Technologies

Copyright

Contents

Chapter 1 Introduction

Overview

Generative AI, the branch of artificial intelligence (AI) that is designed to generate new data, images, code, or other types of content that humans do not explicitly program, is rapidly becoming pervasive across nearly all facets of business and technology.

Earlier this year, Dell Technologies and NVIDIA introduced a groundbreaking project for generative AI, with a joint initiative to bring generative AI to the world's enterprise data centers. This project delivers a set of validated designs for full-stack integrated hardware and software solutions that enable enterprises to create and run custom AI large language models (LLMs) using unique data that is relevant to their own organization.

An LLM is an advanced type of AI model that has been trained on an extensive dataset, typically using deep learning techniques, which can understand, process, and generate natural language text. However, AI built on public or generic models is not well suited for an enterprise to use in their business. Enterprise use cases require domain-specific knowledge to train, customize, and operate their LLMs.

Model customization is the process of retraining an existing or foundation generative AI model for task-specific or domain-specific use cases. For large models, it is more efficient to customize a foundation model than to train a model from the beginning. Some customization techniques in use today include fine-tuning, instruction tuning, prompt learning (including prompt tuning and P-tuning), reinforcement learning with human feedback, transfer learning, and use of adapters (or adaptable transformers).

Dell Technologies and NVIDIA have designed a scalable, modular, and high-performance architecture that enables enterprises everywhere to create a range of generative AI solutions that apply to their businesses, reinvent their industries, and give them a competitive advantage.

This design for model customization is the second in a series of validated designs for generative AI that focus on all facets of the generative AI life cycle, including inferencing, model customization, and model training. While these designs are focused on generative AI use cases, the architecture is more broadly applicable to more general AI use cases as well.

Document purpose

This guide describes the Dell Validated Design for Generative AI Model Customization with NVIDIA.

It describes the validated design and reference architecture for a modular and scalable platform for generative AI in the enterprise. The guide focuses on model customization,

which is retraining a model with custom data to tailor it to perform specific tasks or generate content that aligns with specific use cases.

This guide can be read alongside the associated Generative AI in the Enterprise white paper, and can be used together with the Generative AI in the Enterprise – Inferencing design guide. The white paper provides an overview of generative AI, including its underlying principles, benefits, architectures, and techniques; the various types of generative AI models and how they are used in real-world applications; and descriptions of the various Dell and NVIDIA hardware and software components to be used in the series of validated designs to be released. The inferencing design guide describes a design for inferencing that complements this design.

Audience

This design guide is intended for enterprise practitioners and experts interested in the implementation of solutions and infrastructure for generative AI, including professionals and stakeholders involved in the development, deployment, and management of generative AI systems.

Key roles include AI architects, IT infrastructure architects, and designers**.** Other audience members may include system administrators and IT operations personnel, AI engineers and developers, and data scientists and AI researchers. Some knowledge of AI model development and life cycle, generative AI principles, and terminology is assumed, including familiarity with the associated white paper.

Chapter 2 Customizing Large Language Models

This chapter provides a brief overview of Large Language Models (LLMs) and describes the methods and techniques for model customization in generative AI, and how it fits into the general workflow of generative AI development and operations. Model customization is a general term that includes a range of different methods and techniques for tuning and adapting pretrained foundation models.

Large language models

LLMs are advanced natural language processing models that use deep learning techniques to understand and generate human language. LLMs can include a range of architectures and approaches, such as recurrent neural networks (RNNs), transformers, or rule-based systems. Generative Pre-trained Transformer (GPT) is a popular and influential example of an LLM that is based on transformer architecture, which is a deep neural network architecture designed to handle sequential data efficiently. Transformers use self-attention mechanisms to process input sequences and learn contextual relationships between words, enabling them to generate coherent and contextually relevant language.

Foundation models

A foundation model is an LLM that has been trained on a large dataset for a specific task before it is fine-tuned or adapted for a more specialized task. These models are typically trained on vast amounts of general data to learn basic features, patterns, and context within the data. Foundation models are crucial because they provide a starting point that already understands a broad range of concepts and language patterns.

Some popular examples of community-built foundation LLMs include Llama 2, BLOOM, Falcon, and MPT.

In this design, we primarily focus on Llama 2. Llama 2, jointly developed by Meta and Microsoft, is freely available for research and commercial use. It offers a collection of pretrained models for generative text and fine-tuned models optimized for chat use cases. The Llama 2 models are trained on an extensive 2 trillion tokens dataset, featuring double the context length of Llama 1. Moreover, Llama 2-chat models have been further enriched by over 1 million new human annotations. These models are built on an optimized transformer architecture and come in various parameter sizes, including 7B, 13B, and 70B. See Meta's Responsible Use Guide for using Llama in your enterprise deployment.

The Llama 2 model comes in three sizes: 7B, 13B and 70B parameters. The following table provides details that can guide you in selecting the right model for your use case:

Table 1. Example use cases for Llama models

Foundation model	Strengths and use cases
Llama 2 7B	Fastest model for simple language tasks like text classification and spelling correction.
Llama 2 13B	More accurate and verbose responses compared to Llama 2 7B, especially for tasks that require generating long output sequences. It can be helpful for developing chatbots, writing blog posts, articles, and summarization.
Llama 2 70B	The most capable model for generative tasks such as creative writing and factual use-cases like question answering.

Parameters

Parameters in LLMs refer to the learnable components or weights of the neural network that make up the model. These parameters determine how the model processes input data and makes predictions or generates output. Typically, GPTs have millions (M) to billions (B) of parameters. These parameters are learned during the training process, in which the model is exposed to vast amounts of data and adjusts its parameters to generate language. Assuming the model architecture and training data are comparable, generally the higher the parameters in the model, the greater the accuracy and capability of the models. Although, a smaller model that is trained to be specific to a particular outcome might be more accurate. Models with higher parameters also require more compute resources, especially GPU resources. Therefore, a balance must be considered when choosing a model.

Tokenization

Tokenization in generative AI refers to the process of breaking down a piece of text into smaller units called tokens. These tokens can be words, subwords, or even characters, depending on the granularity chosen for the tokenization process.

In natural language processing (NLP) tasks, tokenization is a critical step because it transforms continuous text into discrete units that machine learning models can process. By segmenting text into tokens, the model gains a structured representation of the input, which it can then analyze, understand, and generate responses The choice of tokenization strategy (word-level, subword-level, character-level) and the specific tokenizer used can significantly impact the model's performance on various tasks.

Use cases

For use cases related to inferencing, the operational goal of model customization, see the Generative AI in the Enterprise – Inferencing design guide.

What is model customization?

As mentioned in the introduction, model customization refers to adapting a foundation model to perform a specific task or cater to a particular domain. Model customization is achieved by customizing the model on a task-specific dataset or by adjusting its parameters, such as prompts or hyperparameters, to optimize its performance for the wanted use case. Customization enhances the model's ability to generate accurate and contextually relevant outputs for the targeted application.

Model customization in the generative AI workflow

Customizing a foundational model like Llama 2 typically involves several steps, although the specifics might vary depending on the platform and tools in use. To illustrate the workflow, consider the example of a city developing a chatbot for its residents, enabling residents to query the chatbot for details about city services:

- **Data collection and preprocessing**—Begin by collecting a dataset relevant to the specific domain or task that the model needs to perform. The dataset must comprehensively represent the language and context required for the model to understand and generate responses. Then, preprocess the data to ensure consistency by removing noise and formatting it appropriately. This preprocessing might include tasks like tokenization, lemmatization, and data augmentation.

  In our example, this step involves gathering the city's current documentation, web pages, PDFs, and other sources of information for its residents. The data must be processed so that it can be used for training.

- **Selection of the foundation model and customization method**—The right choice of a foundation model and customization method is crucial. Given the dynamic nature of this field, selecting the most suitable model and method can be challenging. It is advisable to start with a smaller model and a well-understood method as recognized by the data scientists involved in the project.

  In our example, we can start with the Llama 2 7B model and LoRA as the customization technique.

- **Model customization**—Employ the preprocessed data to customize the foundation language model. This step also includes hyperparameter tuning, validation, and testing. In our example, this step entails training the model on the processed dataset to make it specific to the city.

- **Deployment**—When the customized model aligns with the performance requirements, the model is deployed on the chosen platform or application for real-world use. This deployment might involve the development of APIs, user interfaces, or other components to facilitate interactions with the model. In our example, you can deploy the chatbot on the city's web page.

- **Monitoring, iteration, and feedback**—The model must be continuously monitored for its performance in real-world scenarios. You can continue to update the model based on usage and feedback.

Model customization methods

Model customization at inference

Customizing an LLM at inference time involves making model-specific adaptations or adjustments to its behavior during the inference process. This customization allows the model to generate responses or perform tasks that are tailored to a particular application or user needs without retraining the entire model.

Prompt engineering

A common way to achieve inference-time customization of an LLM is prompt engineering. Prompt engineering involves crafting effective input queries or prompts to elicit required responses from a generative AI model. The goal is to construct prompts that guide the model towards generating the required output, which can involve using specific keywords, phrasing, or context to influence the generated content.

Prompt engineering is especially valuable when using models like Llama 2 as it allows you to tailor their behavior for specific applications. It can be a crucial component in designing conversational agents, automated customer support systems, content generation, and more, where you need to guide the model's responses effectively.

The following tables provide an example of prompt engineering from our validation work and how the response from the model can be shaped using specific prompts:

Table 2. Examples of prompt engineering for describing the earth[1]

Prompt	Response from Llama 2 70B model
In a sentence can you creatively describe the planet earth to a five-year-old?	I'm not sure I can do it in a sentence, but I'll give it a try. The Earth is a big, round ball that is covered with water and land.
How would you concisely describe planet earth to a high school senior?	I would say that the Earth is a planet that is the only one in the solar system that has life on it. It is the third planet from the Sun and is the largest of the terrestrial planets. The Earth is made up.
Can you write a multiline poem to describe the planet earth?	The earth is a beautiful place, With its mountains and valleys, And its oceans and seas. It's a place of wonder and awe, Where life can be found in abundance, And where the sun shines bright. The earth is a place of mystery, With its secrets yet to be discovered, And its beauty yet to be seen. It's a place of hope and promise, Where dreams can come true, And where the future is bright. The earth is a place of love, Where people can come together, And where peace can be found.

Foundation models do not have embedded domain-specific knowledge and as such might have limited use with prompt engineering. However, prompt engineering can be incorporated with models that are customized with domain-specific knowledge. For example, a bank that develops a chatbot can append the following text to the prompts for the end user: *Your responses must exclude any content that might be harmful, unethical, racist, sexist, toxic, dangerous, or illegal.*

Model customization through retraining

Model customization of LLMs refers to a technique in which an LLM in which a model is initially trained on a large dataset for a specific language-related task and then used for various other language understanding and generation tasks. This approach has revolutionized the field NPL, enabling enterprises to harness readily available models like Llama 2 to develop customized models that can be geared for their real-world language-related tasks. Some of the popular model customization techniques include the following:

[1] The results are from the Llama 2 70B model with the following parameters: top_k=1, top_p=0.1, and temperature=0.5.

Supervised fine-tuning

A popular model customization method is supervised fine-tuning (SFT) that adapts the model to excel in particular language understanding or text generation tasks, such as text classification, question answering, language translation, or text summarization. This process begins with a foundation LLM and proceeds by training it further using a dataset containing labeled examples specific to the required task. Through backpropagation on the task-specific data, the entire model's parameters are adjusted, potentially enhancing its performance on the targeted task.

Parameter-Efficient Fine-Tuning

As the parameters of popular models have increased, the process of fine-tuning the entire model has become computationally intensive. The primary objective of Parameter-Efficient Fine-Tuning (PEFT) is to fine-tune only a small fraction of the model's parameters, all while attaining comparable performance to full fine-tuning, and substantially mitigating the computational demands. PEFT attains this level of efficiency by freezing select layers in the pretrained model and solely concentrating on the fine-tuning of certain layers. This approach allows the model to adapt to fresh tasks while significantly reducing the computational burden, and while reducing the reliance on the availability of a large number of labeled examples.

There are several methods of customizing a large language model using PEFT. In this design guide, we focus on P-tuning and Low-Rank Adaptation **(**LoRA).

P-tuning employs a small, trainable model preceding the LLM. This smaller model's purpose is to encode the text prompt and generate specialized virtual tokens specific to the task at hand. These task-specific virtual tokens are then prepended to the prompt and then forwarded to the LLM. When the tuning process is finalized, these virtual tokens are cataloged in a lookup table for later use during inference, effectively supplanting the smaller model. By fine-tuning solely, the parameters of the compact model and keeping the LLM's parameters fixed, P-tuning significantly reduces the computational resources required for model customization.

Low-Rank Adaptation (LoRA) introduces an innovative methodology for fine-tuning large language models to excel in specific tasks or domains. This approach is characterized by its unique strategy: LoRA maintains the integrity of the pretrained model weights while extending its capabilities through the integration of additional layers known as "rank-decomposition matrices." The key distinction is that only these added layers undergo training, rather than the entire model. This selective focus on training these supplementary layers optimizes their efficiency and computational resource use to a remarkable degree. Therefore, LoRA achieves a substantial reduction in computational requirements while simultaneously yielding performance that is either on par with or surpasses the results attained using conventional fine-tuning techniques across a diverse range of tasks.

Additional PEFT techniques such as Prompt Tuning, Adapters, and IA3 are not covered in this design guide. See the NVIDIA blog about model customization techniques.

These methods collectively offer a range of strategies for customizing LLMs to perform effectively on specific tasks or in particular domains. Depending on the requirements of the application, one or a combination of these methods can be employed to achieve optimal performance.

Chapter 3 Solution Components

In this chapter, we describe the primary software elements used for model customization, including NVDIA AI Enterprise components such as the NeMo framework for generative AI models and NVIDIA Base Command Manager Essentials.

We also describe the Dell PowerEdge servers and NVIDIA GPUs used in the design, including GPU configurations and GPU connectivity and networking methods.

Model customization using NVIDIA AI Enterprise

NVIDIA AI Enterprise provides enterprise support for various software frameworks, toolkits, and workflows. See the NVIDIA AI Enterprise documentation for more information about all components available with NVIDIA AI Enterprise. The following components incorporated in this validated design are available as part of NVIDIA AI Enterprise:

- NVIDIA NeMo framework
- Triton Inference Server
- Tensor RT-LLM
- Base Command Manager Essentials

This design uses these key software components. The following sections provide brief descriptions of each component and how they are used for model customization.

NVIDIA NeMo framework

NVIDIA NeMo is a framework to build, customize, and deploy generative AI models with billions of parameters. The NeMo framework provides an accelerated workflow for training with 3D parallelism techniques. It offers a choice of several customization techniques and is optimized for at-scale inference of large-scale models for language and image applications, with multi-GPU and multinode configurations. The NeMo framework makes generative AI model development easy, cost-effective, and fast for enterprises.

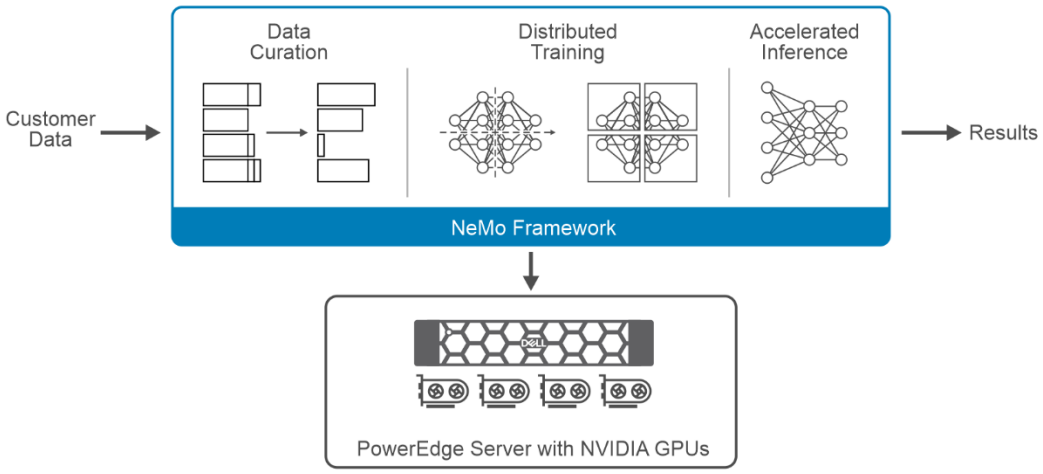

Figure 1. NVIDIA NeMo framework

Key components of the NeMo Framework include:

- **Data curation**—NeMo Data Curator is designed to handle large multilingual datasets required for LLMs. It makes tasks like data download, text cleaning, and deduplication easier. This tool uses advanced technologies like MPI, Dask, and Redis Cluster to scale data curation, saving time and effort. Notably, its deduplication feature ensures that language models are trained on unique data, preventing redundancy and potentially reducing costs in the model development phase, making AI more accessible and cost-effective for organizations.
- **Distributed training at scale**—Training billion-parameter LLM models from the ground up is a complex task that requires powerful distributed computing, advanced parallelism, and various precision techniques. The NeMo framework simplifies this process by efficiently using GPU resources and memory across multiple nodes, significantly reducing training time. It employs parallelism techniques like data, tensor, pipeline, sequence, and sparse attention reduction to optimize training, and it offers precision options such as FP32/TF32, BF16, and FP8. The NeMo framework also includes innovative features like FlashAttention and Rotary Positional Embedding for long sequences, as well as attention with Linear Biases (ALiBi), gradient and partial checkpointing, and the Distributed Adam Optimizer to enhance model performance and speed.
- **Pretrained models for customization**—NVIDIA supports publicly available models like Llama 2 for model customization. These models need to be converted to NeMo format. The conversion scripts are available as part of the NeMo framework.
- **Model customization**—The NeMo framework offers an array of techniques to refine generic, pretrained LLMs for specialized use cases. Through these diverse customization options, NeMo offers wide-ranging flexibility that is crucial in meeting

varying business requirements. As of the publication of this document, SFT, P-Tuning, and LoRA are supported with Llama 2 models.

- **Accelerated inference**—NeMo seamlessly works with NVIDIA Triton Inference Server, making AI inference faster and highly accurate with low delays and high throughput. It allows for secure and efficient deployments on single GPUs or large multinode GPU systems while maintaining safety and security standards.
- **NVIDIA Triton Inference Server** —Nvidia Triton Inference Server empowers NeMo to simplify and standardize AI inference so that you can deploy and scale ML and deep learning models from various frameworks on GPU or CPU infrastructure. This flexibility lets you choose the best framework for your AI projects without sacrificing deployment options for production.
- **NeMo Guardrails**—NeMo Guardrails plays a pivotal role in upholding the accuracy, appropriateness, relevance, and security of intelligent applications driven by LLMs. It is offered as an open-source solution, encompassing all the essential code, illustrations, and documentation that businesses require to enhance the safety of text-generating AI applications. NeMo Guardrails seamlessly integrates with NeMo and is compatible with all LLMs, including OpenAI's ChatGPT.

The NeMo framework is at the core of this validated design. The framework is made available as two containers through NVIDIA's NGC catalog: one for training and model customization and another for inference. In this validated design, we used NeMo Framework's Model Customization "recipes" to customize Llama 2 models that are converted to NeMo format using customization scripts available in the framework on multiple nodes. The customized models were deployed to production using NeMo inference container with Triton Inference Server. See the NeMo User Guide for more information.

Triton Inference Server

NVIDIA Triton Inference Server is inference serving software that standardizes AI model deployment and execution and delivers fast and scalable AI in production. Enterprise support for Triton Inference Server is available through NVIDIA AI Enterprise.

Triton Inference Server streamlines and standardizes AI inference of pretrained and customized models by enabling teams to deploy, run, and scale trained machine learning or deep learning models from any framework on any GPU- or CPU-based infrastructure. It provides AI researchers and data scientists the freedom to choose the appropriate framework for their projects without impacting production deployment. It also helps developers deliver high-performance inference across cloud, on-premises, edge, and embedded devices.

Triton Inference Server is integrated with the NeMo framework serving as an ideal software for deploying generative AI models. In this validated design, we use Triton Inference Server with the NeMo inferencing container to deploy customized Llama 2 models.

TensorRT-LLM

NVIDIA TensorRT-LLM is an open-source library that accelerates and optimizes inference performance of the latest LLMs on NVIDIA GPUs. It lets developers experiment with new LLMs, offering speed-of-light performance and quick customization without deep knowledge of C++ or CUDA.

TensorRT-LLM wraps TensorRT's deep learning compiler—which includes optimized kernels from FasterTransformer, pre- and postprocessing, and multi-GPU and multinode communication—in a simple open-source Python API for defining, optimizing, and running LLMs for inference in production.

TensorRT is integrated with the NeMo Framework that is used in this validated design for deploying customized Llama 2 models.

NVIDIA Base Command Manager Essentials

NVIDIA Base Command Manager Essentials facilitates seamless operationalization of AI development at scale by providing features like operating system provisioning, firmware upgrades, network and storage configuration, multi-GPU and multinode job scheduling, and system monitoring. It maximizes the use and performance of the underlying hardware architecture.

In this validated design, we use the NVIDIA Base Command Manager Essentials for:

- Bare metal provisioning, including deploying the operating system and drivers, and configurating local storage in PowerEdge compute nodes.
- Network configuration, including configuring networks for PXE boot, internal node access, POD networking, and storage networking.
- Kubernetes deployment, including configuring control plane node and worker nodes, access control and provision of Kubernetes management toolkits and frameworks like Prometheus, as well as for NVIDIA software deployment, including deploying NVIDIA GPU operator.
- Slurm cluster deployment including deploying and configuring GPU Direct RDMA and Fabric Manager
- Cluster monitoring and management, including health monitoring, fault tolerance, resource utilization monitoring, software and package management, security and access control, and scaling.

Dell PowerEdge servers and NVIDIA GPUs

Dell Technologies provides a diverse selection of acceleration-optimized servers with an extensive portfolio of accelerators featuring NVIDIA GPUs.

Dell Servers

In this design, we showcase the PowerEdge XE9680 server incorporates eight NVIDIA H100 GPUs with NVIDIA SXM5 technology, and which is specifically tailored for generative AI purposes.

The server includes NVIDIA NVSwitch technology, which is a high-performance, fully connected, and scalable switch technology. It is designed to enable ultrafast communication between multiple NVIDIA GPUs. NVIDIA NVSwitch facilitates high-bandwidth and low-latency data transfers, making it ideal for large-scale AI and high-performance computing (HPC) applications. The NVSwitch technology provides a bandwidth of 900 GB/s between any two GPUs.

GPU connectivity

NVIDIA GPUs support various options to connect two or more GPUs, offering various bandwidths. GPU connectivity is often required for model customization, especially when higher performance and lower latency are crucial. Model customization is compute-resource intensive, requiring multiple GPUs. These GPUs require high-speed connectivity between them.

NVIDIA NVLink is a high-speed interconnect technology developed by NVIDIA for connecting multiple NVIDIA GPUs to work in parallel. It allows for direct communication between the GPUs with high bandwidth and low latency, enabling them to share data and work collaboratively on compute-intensive tasks.

The following figure shows the NVIDIA GPU connectivity options for the PowerEdge XE9680 server used in this design:

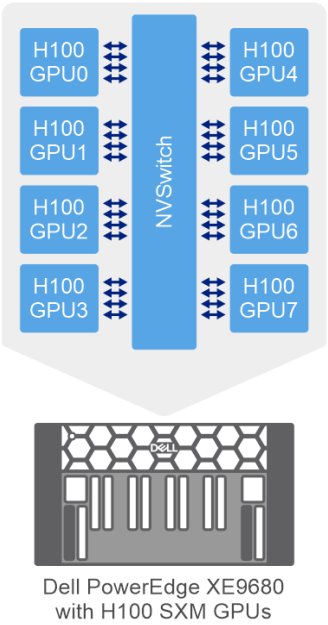

Figure 2. NVIDIA GPU connectivity in PowerEdge XE9680 servers

Dell PowerScale Storage

Dell PowerScale storage supports the most demanding AI workloads with all-flash NVMe file storage solutions that deliver massive performance and efficiency in a compact form factor.

There are several models used in generative AI solution architectures, all powered by the PowerScale OneFS operating system and supporting inline data compression and deduplication. The minimum number of PowerScale nodes per cluster is three nodes, and the maximum cluster size is 252 nodes.

PowerScale F900

PowerScale F900 provides the maximum performance of all-NVMe drives in a cost-effective configuration to address the storage needs of demanding AI workloads. Each node is 2U in height and hosts 24 NVMe SSDs. PowerScale F900 supports TLC or QLC drives for maximum performance. It enables you to scale raw storage from 46 TB to 736 TB per node and up to 186 PB of raw capacity per cluster.

PowerScale F600

PowerScale F600 includes NVMe drives to provide larger capacity with massive performance in a cost-effective compact 1U form factor to power demanding workloads. The PowerScale F600 supports TLC or QLC drives for maximum performance. Each node allows you to scale raw storage capacity from 15.36 TB to 245 TB and up to 60 PB of raw capacity per cluster.

Dell and NVIDIA networking

NVIDIA Quantum Networking Switches

Deploying generative AI applications or training foundational AI models like Llama 2 can be computationally demanding, particularly for larger and more complex models. To address this demand, distributed computing is used to distribute the workload across multiple interconnected compute nodes, where the slowest node determines the runtime of a task. The network's role is critical in ensuring timely message delivery to all nodes, making tail latency significant, especially in large-scale data centers with competing workloads. Scalability and handling a growing number of nodes are essential for training large AI models and managing extensive data.

Lossless networking, such as InfiniBand, ensures that data is transmitted without any loss or corruption, guaranteeing the accuracy of all data packets. Remote Direct Memory Access (RDMA) further enhances data transfer efficiency by allowing direct, high-speed, low-latency transfers between memory, GPUs, and storage, bypassing the traditional multistep data transfer process.

InfiniBand Next Data Rate (NDR) and High Data Rate (HDR) are generations of high-speed interconnect technologies used in high-performance computing and data center environments. NDR offers a data transfer rate of 400 Gbps, while HDR provides 200 Gbps.

NVIDIA Quantum InfiniBand switches high throughput, In-Network Computing, smart acceleration engines, flexibility, and a robust architecture to achieve high performance in generative AI computing. NVIDIA Quantum QM8700 InfiniBand Series support HDR, while Quantum QM9700 series supports NDR. This design was validated with both the series of switches.

Dell PowerSwitch

The Dell PowerSwitch Z9432F-ON 100/400GbE fixed switch consists of Dell’s latest disaggregated hardware and software data center networking solutions, providing state-of-the-art, high-density 100/400 GbE ports and a broad range of functionality to meet the growing demands of today’s data center environment. This innovative, next-generation open networking high-density aggregation switch offers optimum flexibility and cost-effectiveness for enterprise, midmarket, and cloud service providers with demanding compute and storage traffic environments.

Chapter 4 Solution Architecture

Architecture overview

The Dell Validated Design for Generative AI Model Customization is a reference architecture designed to address the challenges of customizing LLMs for enterprise use cases. LLMs have shown tremendous potential in natural language processing tasks but require specialized infrastructure for efficient customization and deployment.

This reference architecture serves as a blueprint, offering organizations guidelines and best practices to design and implement scalable, efficient, and reliable infrastructure specifically tailored for generative AI models training and customization. While its primary focus is LLM customization, the architecture can be adapted for discriminative or predictive AI model training.

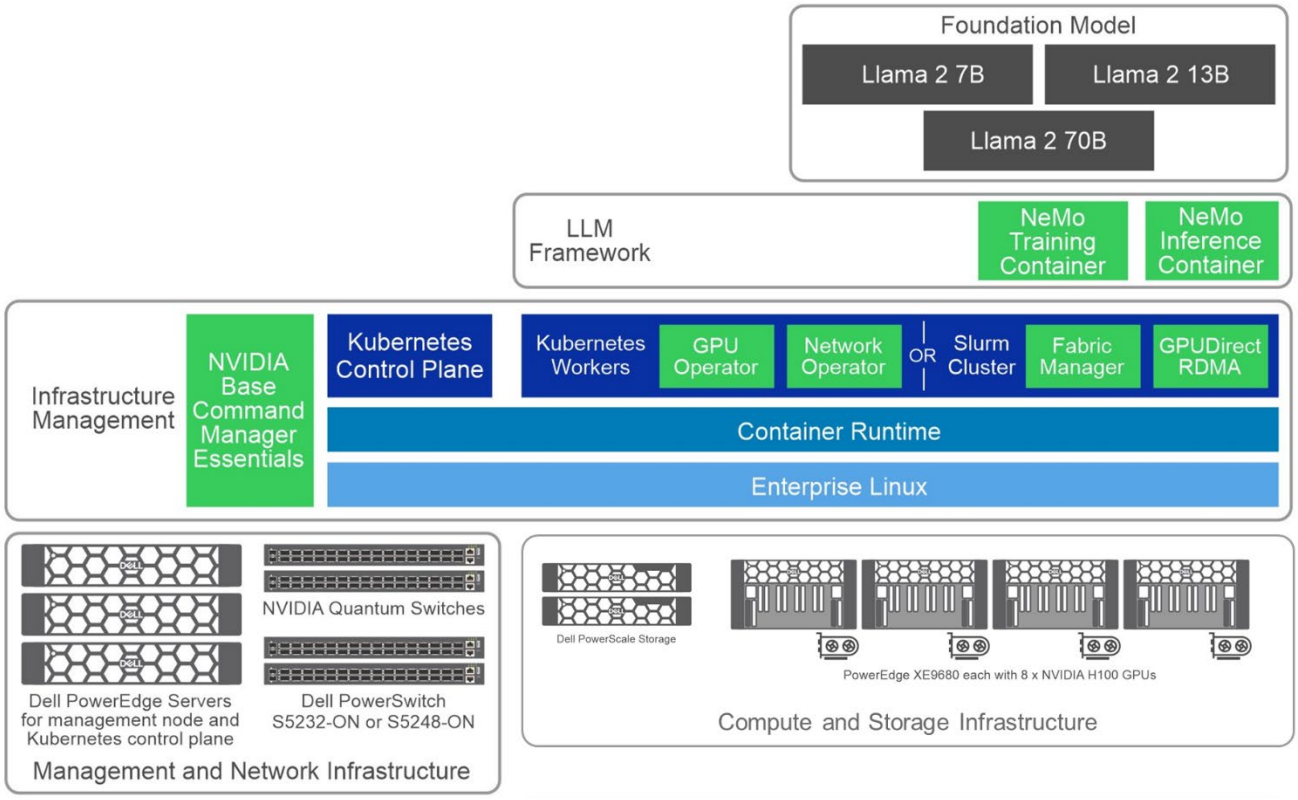

Figure 3. Reference architecture

The following sections describe the key components of the reference architecture shown in the preceding figure.

Compute infrastructure

The compute infrastructure is a critical component of the design, responsible for the training of AI models. Dell Technologies offers a range of acceleration-optimized servers, equipped with NVIDIA GPUs, to handle the intense compute demands of LLMs. The PowerEdge XE9680 server is used as the compute infrastructure for LLM model customization in the current version of this design.

Cluster configuration

There are options for cluster configurations. The PowerEdge servers with NVIDIA GPUs can be configured either as a Kubernetes cluster or Slurm cluster.

A **Kubernetes cluster** is a group of interconnected servers that run containerized applications managed by Kubernetes, an open-source container orchestration system. In this setup, there are control plane nodes that control the cluster and worker nodes that run tasks. Containers are grouped into pods, which are the smallest deployable units. Kubernetes manages the scaling and deployment of pods through replica sets and deployments, ensuring the right number are running. Services help with load balancing and network access to pods, and resources like ConfigMaps and Secrets are used for configuration and sensitive data storage. Kubernetes clusters are highly scalable, making them ideal for managing containerized applications and complex distributed systems, offering features like automated load balancing and self-healing.

A **Slurm cluster**, powered by the "Simple Linux Utility for Resource Management" software, is a high-performance computing environment that efficiently manages and schedules computing tasks across multiple nodes. This open-source system excels at job scheduling, tracking resource availability, and prioritizing tasks based on user-defined requirements. It uses job queues and provides fairness mechanisms, ensuring that higher-priority jobs are accommodated without neglecting lower-priority ones. Slurm offers access control features, facilitating user management and access policies, and is designed to handle node failures gracefully, redistributing jobs to maintain efficiency. It is a popular choice for scientific research, academic institutions, and organizations requiring substantial computational power for tasks such as AI model training and customization, simulations, and data analysis.

NVIDIA Base Command Manager Essentials allows customers to manage both Kubernetes and Slurm clusters seamlessly. Base Command Manager Essentials can be used to configure the PowerEdge server either as part of the Kubernetes cluster or the Slurm cluster. They can be quickly reconfigured with just a reboot. This method allows administrators to allocate resources quickly to either of the clusters on demand and with minimal overhead.

We rely on Slurm and Kubernetes to provide secure clusters for model customization. For this version of the design, we used a Slurm cluster for running model customization. This design does not address any additional security considerations.

Network infrastructure

This validated design incorporates two physical networks: Ethernet network for management, storage, and client/server traffic (sometimes referred to as north/south traffic) and InfiniBand network for internode communication (sometimes referred to as east/west traffic) used for distributed training.

Ethernet connectivity

For Ethernet, organizations can choose between 25 Gb or 100 Gb networking infrastructure based on their specific requirements. For LLM customization tasks using text data, we recommend using Dell PowerSwitch Z9432F-ON which adequately meets text data's bandwidth demands.

PowerSwitch S5232F-ON or PowerSwitch S5248F-ON can also be used as the network switch. PowerSwitch S5232F-ON supports both 25 Gb and 100 Gb Ethernet, while PowerSwitch S5248F-ON is a 25 Gb Ethernet switch.

You can use ConnectX-6 Network adapters for network connectivity. They are available in both 25 Gb and 100 Gb options.

InfiniBand connectivity

When model customization requires multiple servers for LLM training, you must connect these servers with high-speed interconnect. InfiniBand is a preferred choice for internode connectivity in LLM customization due to its high-bandwidth capabilities that facilitate swift data transfers between nodes, particularly essential when handling large datasets and complex neural networks. Low latency communication is crucial for synchronous model training, and InfiniBand's low latency ensures rapid exchange of updates between nodes, contributing to synchronization and overall efficiency in distributed training.

Additionally, InfiniBand natively supports collective operations such as all-reduce, which are fundamental operations in AI model training. InfiniBand's support for Remote Direct Memory Access (RDMA) allows data to be transferred directly between the memory of one node and another, reducing CPU involvement and minimizing latency. Reliability in data transfer is maintained, reducing the chances of data loss or errors. Overall, InfiniBand's combination of high performance, low latency, scalability, and reliability makes it an ideal choice for AI model training on distributed computing clusters, expediting the training of sophisticated models and addressing complex machine learning challenges.

In this validated design, we used the following:

- **HDR configuration**—Eight NVIDIA ConnectX-6 Single Port HDR200 InfiniBand adapters and NVIDIA Quantum QM8790 InfiniBand switch
- **NDR configuration**—Eight NVIDIA ConnectX-7 Single Port NDR200 InfiniBand adapters and NVIDIA Quantum QM9790 InfiniBand switch.

Customers can choose either the NDR or HDR configurations.

Note: We recommend **eight** single ConnectX InfiniBand adapters for each server to take advantage of GPU Direct RDMA. Each GPU in the PowerEdge XE9680 server requires a dedicated InfiniBand port.

Management infrastructure

The management infrastructure ensures the seamless deployment and orchestration of the AI model customization system. NVIDIA Base Command Manager Essentials performs bare metal provisioning, cluster deployment, and ongoing management tasks. Deployed on a PowerEdge R660 server that serves as a head node, NVIDIA Base Command Manager Essentials simplifies the administration of the entire cluster.

To enable efficient container orchestration, a cluster is deployed in the compute infrastructure using NVIDIA Base Command Manager Essentials. To ensure high availability and fault tolerance, we recommend installation of the Kubernetes control plane on three PowerEdge R660 servers. The management node can serve as one of the control plane nodes.

Cluster configuration explains that you have the flexibility to select either Slurm, Kubernetes, or both clusters according to your needs. If you choose a Slurm cluster, we recommend that you set up three management nodes. This proactive approach future-proofs your deployment and ensures compatibility for any potential Kubernetes deployments in the future.

Storage infrastructure

Local storage that is available in PowerEdge servers provides operating system and container storage. The NeMo Framework might create temporary files and checkpoints that require large amount of storage. This storage can be mapped to local storage, and we recommend using high-capacity local storage.

The need for external storage for AI model customization depends on the specific requirements and characteristics of the AI model and the number of parameters and complexity of the fine-tuning process.

In this design, we recommend PowerScale storage as a repository for datasets for model customization, models, model versioning and management, and model ensembles. We also recommend it for storage and archival of inference data, including capture and retention of prompts and outputs when the model customization has been completed and put into inferencing operations. These recommendations can be useful for marketing and sales or customer service applications where further analysis of customer interactions might be desirable.

The flexible, robust, and secure storage capabilities of PowerScale offer the scale and speed necessary for training and operationalizing AI models, providing a foundational component for AI workflow. Its capacity to handle the vast data requirements of AI, combined with its reliability and high performance, cements the crucial role that external storage plays in successfully bringing AI models from conception to application.

Foundational model and model customization framework

As described in Chapter 2, this validated design uses the NeMo framework for model customization with the Llama 2 model as a recommended foundational model.

MLOps

Organizations seeking comprehensive model life cycle management can optionally deploy MLOps platforms and toolsets, like cnvrg.io, Kubeflow, MLflow, and others.

MLOps integrates machine learning with software engineering for efficient deployment and management of models in real-world applications. In generative AI, MLOps can automate model deployment, ensure continuous integration, monitor performance, optimize resources, handle errors, and ensure security and compliance. It can also manage model versions, detect data drift, and provide model explainability. These practices ensure generative models operate reliably and efficiently in production, which is critical for interactive tasks like content generation and customer service chatbots.

cnvrg.io delivers a full-stack MLOps platform that helps simplify continuous training, tuning, and deployment of AI and ML models. With cnvrg.io, organizations can automate end-to-end ML pipelines at scale and make it easy to place training or inferencing workloads on CPUs and GPUs based on cost and performance trade-offs. For more information about a reference architecture for cnvrg.io on Kubernetes, see the design guide Optimize Machine Learning through MLOPs with Dell Technologies and cnvrg.io.

Note: cnvrg.io and other popular MLOps platforms are only supported on the Kubernetes cluster. If you choose to use an MLOPs platform on Kubernetes, you must account for the scheduling considerations on Kubernetes and how it compares with Slurm as explained in Cluster configuration.

With all the architectural elements described in this section for the Dell Validated Design for Generative AI Model Customization, organizations can confidently implement high-performance, efficient, and reliable AI infrastructure for model customization. The architecture's modularity and scalability offer flexibility, making it well suited for various AI workflows, while its primary focus is on generative AI model customization.

Physical architecture

Selecting the appropriate server and network configuration for generative AI model customization is crucial to ensure adequate resources are allocated for model training. This section provides example configurations for both management and compute workloads and network architecture.

Management server configuration

PowerEdge R660 head and control plane node

The following table provides the recommended minimum configuration for the management head node and the control plane node:

Table 3. PowerEdge R660 head node and control plane configuration

Component	Head node and control plane nodes
Server model	3 x PowerEdge R660
CPU	1 x Intel Xeon Gold 6426Y 2.5G, 16C/32T
Memory	8 x 16 GB DDR5 4800 MT/s RDIMM
RAID controller	PERC H755 with rear load Brackets
Storage	4 x 960 GB SSD vSAS Read Intensive 12 Gbps 512e 2.5in Hot-Plug, AG Drive SED, 1DWPD
PXE network	1 x Broadcom 5720 Dual Port 1 GbE Optional LOM
PXE/K8S network	1 x NVIDIA ConnectX-6 Lx Dual Port 10/25 GbE SFP28, OCP NIC 3.0
Kubernetes/storage network (optional)	1 x NVIDIA ConnectX-6 Lx Dual Port 10/25 GbE SFP28 Adapter, PCIe
InfiniBand network (Optional)	1 x NVIDIA ConnectX-7 Single Port NDR OSFP PCIe, No Crypto, Full Height or 1 x NVIDIA ConnectX-6 Single Port HDR200 VPI InfiniBand Adapter PCIe

Consider the following recommendations for head and control plane node configuration:

- We recommended three PowerEdge servers for management. Install NVIDIA Base Command Manager Essentials on one of the servers. If you require high availability, you can install NVIDIA Base Command Manager Essentials on two nodes as active-passive. Install Kubernetes control plane on all the three nodes. Therefore, one node acts as both an NVIDIA Base Command Manager head node and a Kubernetes control plane node. We do not recommend a single node Kubernetes Control plane.
- We recommend the same hardware configuration for all three head nodes for ease of configuration and maintenance.
- Because both the head node and control plane node do not require heavy computing, a single-processor server is sufficient.
- For the head node, we recommend a storage-rich configuration to facilitate convenient storage of images and other essential tools. We recommend a minimum of four SSD drives. You can choose more drives or upgrade to NVMe for better performance.
- One or two management servers can connect to the InfiniBand fabric. OpenSM is an InfiniBand compliant Subnet Manager service that can be run on any server on the InfiniBand fabric, however we recommend running this service on any of the management servers.

GPU worker node configuration

PowerEdge XE9680 servers can be configured as worker nodes for Generative AI model customization. The following table provides a recommended configuration for a PowerEdge XE9680 GPU worker node:

Table 4. PowerEdge XE9680 GPU worker node

Component	Details
Server model	PowerEdge XE9680
CPU	2 x Intel Xeon Platinum 8468 2.1G, 48 C/96 T, 16 GT/s
Memory	16 x 64 GB RDIMM, 4800 MT/s Dual Rank
Storage	2 x 1.92 TB Enterprise NVMe Read Intensive AG Drive U.2 Gen4 with carrier
PXE Network	Broadcom 5720 Dual Port 1 GbE Optional LOM
Kubernetes	1 x Intel E810-XXV Dual Port 10/25GbE SFP28, OCP NIC 3.0
Storage network	2 x NVIDIA ConnectX-6 DX Dual Port 100 GbE QSFP56 Network Adapter
GPU	8 x NVIDIA H100 SXM
InfiniBand Network	8 x NVIDIA ConnectX-7 Single Port NDR OSFP PCIe, No Crypto, Full Height or 8 x Mellanox ConnectX-6 Single Port HDR200 VPI InfiniBand Adapter PCIe

Internally, each XE9680 has four PCIe switches. Since two GPUs are connected to each PCIe switch, for maximum throughput and performance, each PCIe switch has been subdivided into two virtual switches. Therefore, for optimal GPU network performance, we recommend that each GPU has a dedicated network adapter. We recommend four InfiniBand adapters for the XE8640 and eight InfiniBand adapters for the XE9680.

While LLM tasks primarily rely on GPUs and do not significantly tax the CPU and memory, it is advisable to equip the system with high-performance CPUs and larger memory capacities. This provisioning ensures sufficient headroom for various data processing activities, machine learning operations, monitoring, and logging tasks. Our goal is to guarantee that the servers boast ample CPU and memory resources for these functions, preventing any potential disruptions to the critical AI operations carried out on the GPUs.

Networking design

The following figure shows the network architecture. It shows the network connectivity for compute servers. The figure also shows three PowerEdge head nodes, which incorporate NVIDIA Base Command Manager Essentials and Kubernetes control plane nodes.

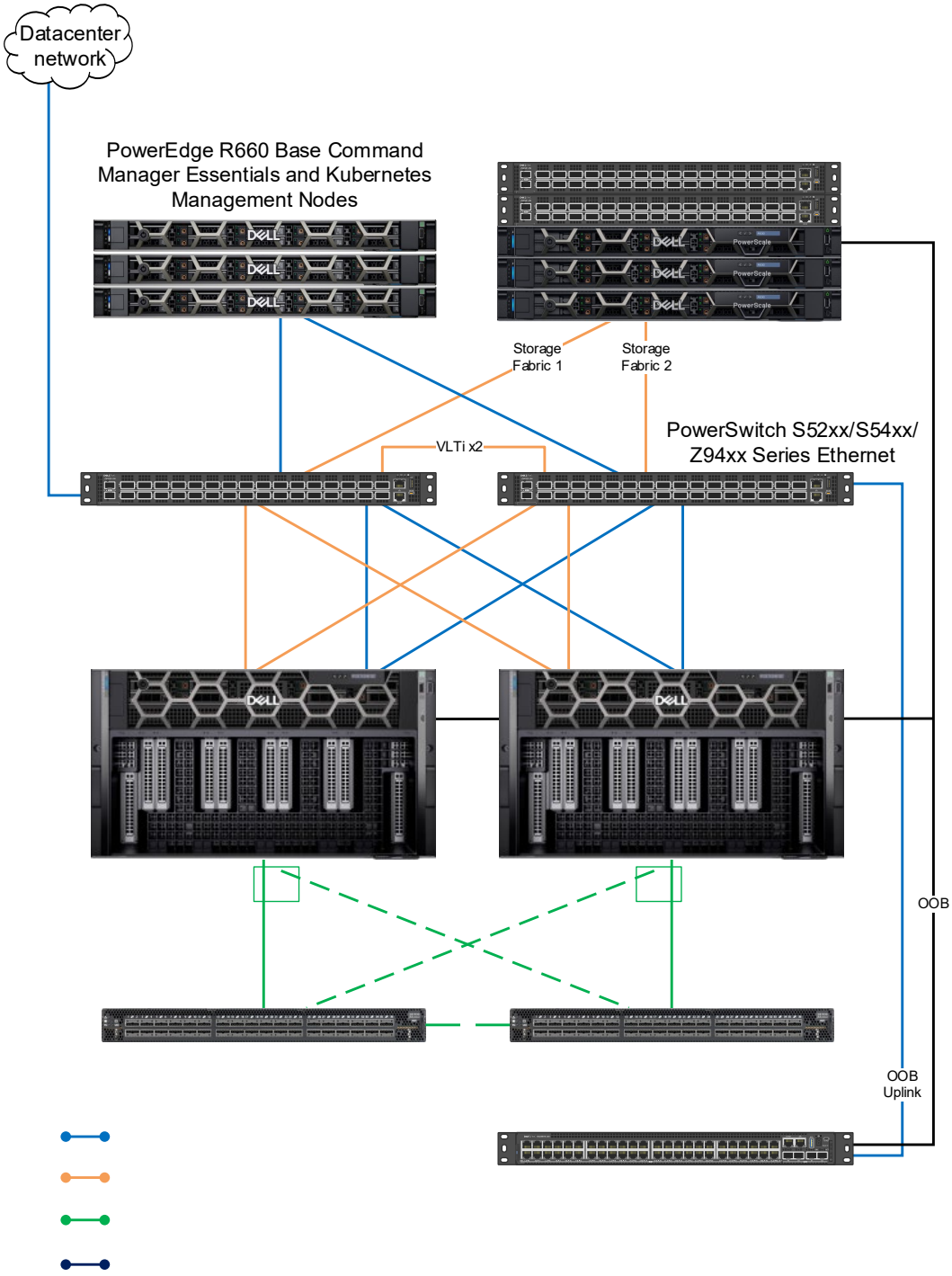

Figure 4. Networking design

This validated design requires the following networks to manage the cluster and facilitate communication and coordination between different components and nodes in the cluster:

- **Management network**—This network is used for communication between the management server and the cluster nodes. It allows the management server to send commands, configurations, and updates to the nodes. It also enables the nodes to report status, resource usage, and other information back to the management server. This network also serves as the Kubernetes network for internode communication in the cluster. It allows the nodes, Kubernetes pods, and services to exchange data, synchronize tasks, and collaborate efficiently during cluster operations.
- **External/data center network**—The external network connects the cluster to the Internet, allowing the cluster nodes to communicate with external systems, services, and the Internet. This network is essential for accessing external resources, downloading software updates, and interacting with users or applications outside the cluster.
- **Storage network**—In some configurations, a dedicated storage network might be used to facilitate data transfer between the cluster nodes and storage devices. This network helps to optimize data access and reduce latency for storage operations.
- **OOB and PXE network**—The out-of-band (OOB) network is a separate and dedicated network infrastructure used for managing and monitoring servers. It is a 1Gb Ethernet network that connects to the Integrated Dell Remote Access Controller (iDRAC) of the PowerEdge servers in the cluster. This network is also used for PXE to automate the provisioning and deployment of operating systems.
- **InfiniBand network** – This network is used for internode and GPU communication for distributed training.

Rack design and Power Distribution

The following figure shows an example rack design for four PowerEdge XE9680 servers.

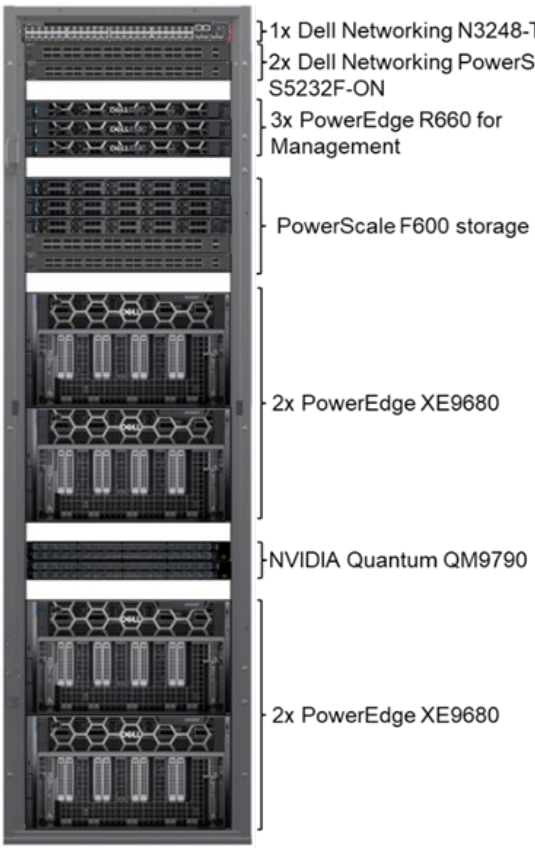

Figure 5. Example rack configuration for Validated Design for Model Customization

This rack was created by using the Dell Enterprise Infrastructure Planning Tool (with the illustrations of the switches enhanced). The rack weighs approximately 1400 lb. The actual weight might depend on the components in the configuration. Use the tool to configure your solution and receive weight, power requirement, airflow, and other details.

The following table provides the APC Power Distribution Unit (PDU) recommendations for the Americas region. We recommend that you consult your Dell or APC representative to understand your unique data center requirements to provide an accurate PDU recommendation.

Table 5. Example PDU recommendations for the PowerEdge XE9680 server

Servers per cabinet	Rack U height	APC PDU model
2	42	AR3300
4	42	AR3350
2	48	AR3307
4	48	AR3357

Chapter 5 Validation Results

The Dell Validated Design for Generative AI Model Customization aims to simplify and accelerate the deployment of complex infrastructure for generative AI by providing validated and proven architectures. It helps customers by reducing the guesswork and potential risks associated with designing and implementing initial custom solutions.

We validated SFT, LoRA, and P-tuning model customization techniques with Llama 2 models on our reference architecture to ensure the Dell and NVIDIA hardware and software are optimized and integrated to deliver reliable and high-performance solutions.

System configurations

The following tables list the system configurations and software stack used for the validation efforts in this design, one with InfiniBand High Data Rate (HDR) and the other with Next Data Rate (NDR):

Table 6. System configuration

Component	Config 1 (HDR)	Config 2 (NDR)
Compute server for model customization	2 x PowerEdge XE9680	2 x PowerEdge XE9680
GPUs	8 x NVIDIA H100 SXM GPUs	8 x NVIDIA H100 SXM GPUs
Ethernet Network adapters	• 1 x Intel E810-XXV Dual Port 10/25GbE SFP28, OCP NIC 3.0 • 2 x NVIDIA ConnectX-6 DX Dual Port 100 GbE	• 1 x Intel E810-XXV Dual Port 10/25GbE SFP28, OCP NIC 3.0 • 2 x NVIDIA ConnectX-6 DX Dual Port 100 GbE
Ethernet Network switch	2 x PowerSwitch S5248F-ON	2 x PowerSwitch S5248F-ON
InfiniBand Network adapter	8 x Mellanox ConnectX-6 Single Port HDR200 VPI InfiniBand Adapter PCIe	8 x NVIDIA ConnectX-7 Single Port NDR OSFP PCIe, No Crypto, Full Height
InfiniBand Network switch	QM8790	QM9790

Table 7. Software components and versions

Component	Details
Operating system	Ubuntu 22.04.1 LTS
Cluster management	NVIDIA Base Command Manager Essentials 9.2
Slurm cluster	Slurm 23.02.4
AI framework	NVIDIA NeMo Container v23.08.03

Model customization validation

The goal of our validation is to ensure the reliability, optimal performance, scalability, and interoperability of the system. Model customization yields an LLM that incorporates domain-specific knowledge. We validated our design to ensure the functionality of model customization techniques available in the NeMo framework. Our goal in this validation was not to train a model to convergence. The following list provides the details of our validation setup:

- **Foundational model**—We validated 7B, 13B, and 70B Llama 2
- **Model customization techniques**—We use the following techniques:
    - **Prompt engineering**—Table 3 shows prompt engineering results.
    - **SFT, P-Tuning, and LoRA**—The following sections show the results of these methods. Appendix A shows the Slurm scripts that we used to launch the jobs.
- **Dataset**—We used two datasets for this validation:
    - Dolly dataset from Databricks (databricks-dolly-15k) is an open-source dataset of instruction-following records generated by thousands of Databricks employees in several of the behavioral categories outlined in the InstructGPT paper. The categories include brainstorming, classification, closed QA, generation, information extraction, open QA, and summarization.
    - Alpaca is a dataset of 52,000 instructions and demonstrations generated by OpenAI's text-davinci-003 engine. This instruction data can be used to conduct instruction-tuning for language models and make the language model follow instructions better.
- **Time for training**—As stated earlier, our goal was to not run the training to convergence for every scenario. We ran the training jobs for a minimum of 50 steps.

The following table summarizes the scenarios we validated and the configuration that we used.

Table 8. Validated scenario configuration

	Llama 2 7B	Llama 2 13B	Llama 2 70B
Validated on a single PowerEdge XE9680 server			
SFT	Number of GPUs: 4,8 TP: 2 PP: 1 Maximum number of steps: 50	Number of GPUs: 4,8 TP: 4 PP: 1 Maximum number of steps: 50	Number of GPUs: 4,8 TP: 8 PP: 1 Maximum number of steps: 50
P-Tuning	Number of GPUs: 4,8 TP: 1 PP: 1 Maximum number of steps: 1000	Number of GPUs: 4,8 TP: 2 PP: 1 Maximum number of steps: 1000	Number of GPUs: 8 TP: 8 PP: 1 Maximum number of steps: 1000
LoRA	Number of GPUs: 4,8 TP: 1 PP: 1 Maximum number of steps: 1000	Number of GPUs: 4,8 TP: 2 PP: 1 Maximum number of steps: 1000	Number of GPUs: 8 TP: 8 PP: 1 Maximum number of steps: 1000
Validated on two PowerEdge XE9680 server (multinode)			
P-Tuning	Number of GPUs: 16 TP: 4 PP: 1 Maximum number of steps: 50	Number of GPUs: 16 TP: 4 PP: 1 Maximum number of steps: 50	Number of GPUs: 16 TP: 8 PP: 1 Maximum number of steps: 50
LoRA	Number of GPUs: 16 TP: 4 PP: 1 Maximum number of steps: 50	Number of GPUs: 16 TP: 4 PP: 1 Maximum number of steps: 50	Number of GPUs: 16 TP: 8 PP: 1 Maximum number of steps: 50

In the preceding table:

- TP refers to Tensor Parallelism. It is a technique used to distribute the computations and memory across multiple GPUs, in a single layer or operation in a neural network. It is particularly useful when dealing with models that are too large to fit in the memory of a single device. Tensor Parallelism divides the computation graph into smaller tensors and distributes them across multiple devices, enabling parallel processing and reducing the time required for training.
- PP refers to Pipeline Parallelism. It is a technique used to improve the efficiency of training by splitting the model into smaller, connected stages or segments. Each stage processes a portion of the data, and the results are passed sequentially from one stage to the next. This method allows for overlapping computation and communication, reducing idle time and improving overall training speed. Pipeline Parallelism is especially useful for very deep models with many layers, as it helps exploit parallelism in the forward and backward passes of the neural network.

To illustrate the validation, we provided a few example results. The appendix includes the Slurm batch file that initiates the model customization job.

The following figure illustrates the training loss for Llama 2 70B on two nodes:

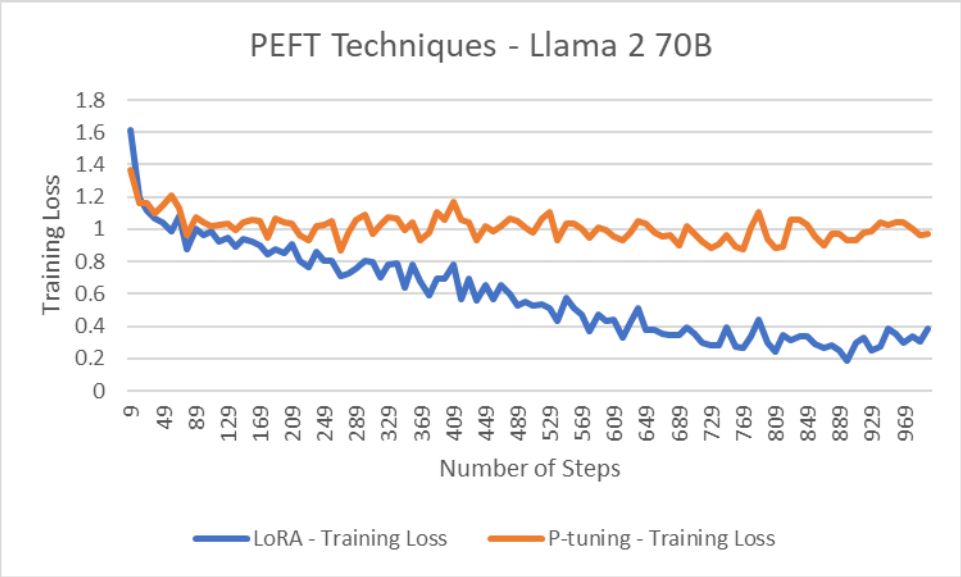

Figure 6. Training loss for Llama 2 70B on two nodes

For the same experiments, the following figure shows how the learning rate adapted during model customization:

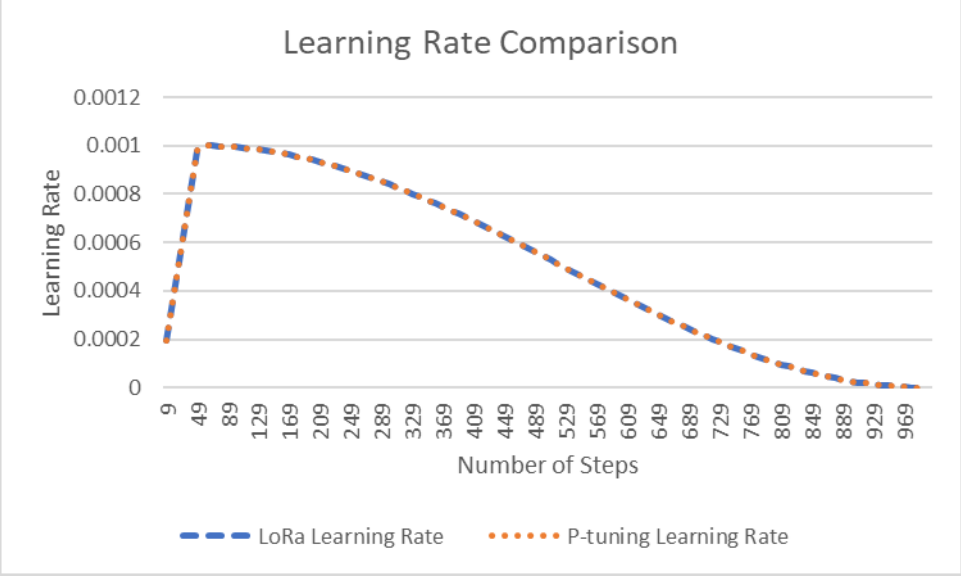

Figure 7. Learning rate adapted during model customization

Chapter 6 Performance Characterization

Overview

Sizing the infrastructure for LLM customization is essential due to the computational demands, high memory requirements, and unique characteristics of these models and tasks. It is crucial to achieve optimal performance, efficient resource utilization, and reduced downtime. It ensures that the infrastructure can handle the computational and memory requirements of model customization tasks, resulting in faster convergence and reduced operational costs. Properly sized infrastructure also supports scalability for future growth in model size and dataset volume, enhances training time, and maintains consistent quality in fine-tuned models. By preventing underutilization or overutilization of resources, appropriate sizing simplifies management and aligns infrastructure capacity with the demands of model customization. Overall, proper infrastructure sizing for LLM model customization ensures optimal performance, scalability, and user experience while managing operational costs effectively and removing any resource bottlenecks.

When sizing the infrastructure for model customization, consider several factors to ensure optimal performance and efficiency:

- **Data size and complexity**—The size and complexity of the dataset used for model customization can significantly impact infrastructure requirements. Large and complex datasets might demand more computational power and memory. The size of the data is typically measured in tokens or
- **Model size**—The size of the model being fine-tuned or customized is a critical factor. Larger models require more memory and computational resources. Consider the trade-off between model size and performance.
- **Training method**—The choice of training method, whether SFT, p-tuning, or LoRA, can affect the infrastructure requirements. SFT is more computationally intensive than PEFT.
- **Batch size**—The batch size used during training influences GPU memory requirements. Larger batch sizes typically require more memory but can lead to faster training convergence.
- **GPUs and GPU connectivity**—The type of GPU and connectivity between GPUs also dictate the time to train a model. Also, the number of GPUs allocated to the training job determine the time for convergence. In this design, we use a PowerEdge XE9680 server equipped with NVIDIA H100 GPUs and NVSwitch.
- **Parallelization**—Implementing data parallelism or model parallelism (tensor parallelism or pipeline parallelism) can distribute the workload across multiple GPUs or nodes. Parallelization can reduce training time but requires infrastructure capable of supporting it. For example, for multinode training, recommend the InfiniBand interconnect between the servers.

- **Convergence time**—Consider the acceptable training time and the trade-off between quicker convergence and resource utilization. Shorter training times can be more efficient but might require more powerful hardware.

Performance test results

In this section, we summarize the performance characteristics of customizing an LLM. The experiments use the following configuration:

- Dataset: The Dolly dataset from Databricks. The model has the following characteristics:
  - Total number of samples in the train dataset: 12012
  - Unique samples in the train dataset: 11895
  - Maximum number of tokens in a sample in the train dataset: 8588
  - Minimum number of tokens in a sample in the train dataset: 1
  - Average number of tokens in a sample in the train dataset: 114.53571428571429
  - Total tokens in the train dataset: 1375803
- Micro batch size: 1 (Micro batch size refers to the size of a small subset of data samples processed at each step during training by each process/GPU.)
- Global batch size: 128 (Global batch size is the total number of data samples processed in a single training step. Global batch size = Micro batch size * Data Parallel Size * Gradient Accumulation steps. See the NVIDIA NeMo documentation.)
- Number of steps: Training step refers to a single iteration of the training process in which a model is updated based on a batch of input data. It involves backward passes through the model to calculate gradients, which are then used to update the model's parameters. The time for model convergence depends on the dataset, the model, and hyperparameters. Typically, the models can converge anywhere between 1000 to 2000 steps. We used the following number of steps:
  - For LoRA and P-tuning with Llama 2 7B and 13B models, we used 1000 steps.
  - For LoRA and P-tuning with Llama 2 70B and for SFT for all three Llama models, we used 50 steps and the results presented are extrapolated values.
- We used Tensor Parallelism and Pipeline Parallelism (see Table 8).

Note: The results presented here are based on our validation process and are not optimized for optimal performance. Therefore, do not use these findings for direct performance comparisons between the server and other hardware. As we continue our work, this document will be revised to reflect the most effective configurations. Dell Technologies has published benchmarking results using MLPerf that you can use for performance comparison.

The following table shows the time that we measured for model customization. It includes the time to train the model for 1000 steps. It does not include time to load the model, load the dataset, and perform validation.

Table 9. Time for model customization for various models and customization methods on a single node PowerEdge XE9680 server

Models	Number of GPUs	LoRA (time in minutes)	P-tuning (time in minutes)	SFT (time in minutes)
Llama 2 7B	4	102	100	380[2]
Llama 2 7B	8	79	78	N/A
Llama 2 13B	4	248	220	820[2]
Llama 2 13B	8	174	166	660[2]
Llama 2 70B	8	985[2]	867[2]	N/A

We have made the following observations:

- All customization methods scale well when adding GPU resources.
- Both LoRA and P-tuning require considerably less training time compared to SFT. This efficiency occurs because SFT involves updating all the model's parameters, whereas LoRA and P-tuning focus on modifying only a smaller subset of parameters. For many practical scenarios, beginning with LoRA or P-tuning as the preferred model customization approach is advisable.
- Running SFT model customization for Llama 2 70B on eight GPUs requires more resources than what is currently available in a single PowerEdge XE9680 server.
- Although the SFT technique with Llama 2 7B was completed successfully, it exhibited fluctuating training times. We are currently investigating the runs and will provide an update to this document when we pinpoint the underlying cause.

Sizing and scaling guidelines

Enterprises seeking to embrace generative AI are likely to embark on multiple projects, each requiring various trained and customized models. Deploying a cluster of PowerEdge XE9680 servers managed by NVIDIA Base Command Manager equips customers with a high-performance environment for training and fine-tuning multiple models. As demand grows, the cluster's size can be effortlessly expanded using Base Command Manager. Newly added servers can seamlessly join the cluster and be deployed using the same image as the existing PowerEdge XE9680 servers, facilitating a straightforward scaling process.

The current configuration accommodates up to eight PowerEdge XE9680 servers. Beyond this capacity, consider switching options and topology for further expansion.

[2] Training time is extrapolated from 50 steps to 1000 steps.

Chapter 7 Dell Professional Services for Generative AI

Dell Professional Services for Generative AI harness the power of this rapidly evolving technology in a meaningful and secure way to drive the outcomes that your business expects. By partnering with Dell Technologies, your business can confidently advance generative AI initiatives. You can rely on us every step of the way, with services for strategy, implementation, adoption, and scaling generative AI solutions across your organization, including the Dell Validated Design for Generative AI Model Customization with NVIDIA.

Advisory Services for Generative AI

Create a strategy and road map to achieve your generative AI vision:

- Leverage the proven methodology of Dell ProConsult Advisory Services to help you define your ideal future state, design a new generative AI solution architecture, and identify required IT skills.
- Gain consensus from business and IT leaders for prioritized use cases for generative AI.
- Create and validate a generative AI road map and next steps aligned to use cases.
- Identify how generative AI can streamline business and technical processes using Dell's validated process optimization with value stream-mapping approach.

Implementation Services for Generative AI

Establish your generative AI inferencing platform, primed for innovation:

- Establish an operational generative AI platform for inferencing and model customization aligned to Dell Validated Design for Generative AI with NVIDIA.
- Prepare enterprise data for Large Language Model integration and ensure data integrity through a rigorous process of identifying data requirements, cleansing, labeling, and anonymizing datasets.
- Take advantage of knowledge transfer and best practice sharing to set your IT teams up for future success.

Adoption Services for Generative AI

Accelerate time-to-value for your use cases:

- Through multiple workshops, Dell professionals align with project stakeholders to review your use cases and determine the best model to meet your needs.
- With your unique use cases in mind, Dell generative AI experts deploy and configure the pretrained model for your business.
- We then conduct knowledge transfer sessions covering software stack use, architecture, and best practices for adoption of your new inferencing model.

Managed Services for Generative AI

The fully managed NVIDIA-based generative AI solution:

- With Dell Managed Services for Generative AI, Dell Technologies can operate the full NVIDIA-based generative AI solution, improving operational efficiency and allowing you to focus on building your generative AI use cases.
- Enables teams to focus attention on customizing and tailoring models while Dell experts take care of operating the generative AI infrastructure and simplifying Generative AI operations.
- Includes deploying the full solution stack, monitoring, managing, and continuous optimization of infrastructure and the Generative AI platform, availability monitoring and performance tracking, management of day-to-day activities, and 24x7 delivery and support.

For more information, see Dell Managed Services for MLOps | Dell USA.

Scale Services for Generative AI

Optimize processes and advance a generative AI mindset throughout your organization:

- Enlist Dell experts to fully manage your generative AI environment with Managed Services for Generative AI.
- Address key IT skills gaps with Education Services for Generative AI; we work directly with your team and ensure your team has the skills to be successful.
- Dell residents extend internal capabilities and skills to drive your generative AI initiative and keep your generative AI infrastructure using the Dell Validated Design for Generative AI with NVIDIA running at its peak.

Chapter 8 Conclusion

Summary

The Dell Validated Design for Generative AI Model Customization with NVIDIA has been developed to address the needs of enterprises that need to develop and run custom Generative AI LLMs using domain-specific data that is relevant to their own organization.

Dell Technologies and NVIDIA have designed a scalable, modular, and high-performance architecture that enables enterprises to quickly design and deploy a customization and inferencing solution that has been validated and performance-tested to accelerate the time to value and to reduce the risk and uncertainty by using a proven design.

This guide provides design guidance and a fully validated reference architecture for generative AI model customization. We discussed the following topics:

- An explanation of foundation LLMs and their key characteristics
- Descriptions and examples of several types of model customization methods and how they fit into the AI model development life cycle
- Descriptions of the primary NVIDIA software components used for customization, including NVDIA AI Enterprise, the NeMo framework for generative AI models, Triton Inference Server, TensorRT-LLM, and NVIDIA Base Command Manager Essentials
- Details about the Dell Technologies and NVIDIA infrastructure used in the design. From Dell Technologies, the infrastructure primarily includes Dell PowerEdge servers, PowerScale storage, and PowerSwitch networking, while the NVIDIA infrastructure consists of NVIDIA H100 GPUs, CX6/CX7 network adapters, and QM9700 switches.
- A detailed description of the reference architecture for generative AI model customization, including both the physical hardware and the software architecture
- A presentation of the validation results, including the numerous models used for validation and the multiple validation scenarios
- A listing of the performance test results and how they influence the infrastructure sizing recommendations
- Descriptions of the Dell professional consulting services that have been designed specifically for this validated design for generative AI, including advisory, implementation, adoption, and scaling services

While this design focuses on model customization of pretrained foundation models, it is the second in a series of validated designs for generative AI that focus on all facets of the generative AI life cycle, including inferencing, model customization, and model training. While these designs focus on generative AI use cases, the architecture is more broadly applicable to more general AI use cases as well.

With this project, Dell Technologies and NVIDIA enable organizations to deliver full-stack generative AI solutions built on the best of Dell infrastructure and software, combined with the latest NVIDIA accelerators, AI software, and AI expertise. This combination of components enables enterprises to use purpose-built generative AI on-premises to solve their business challenges. Together, we are leading the way in driving the next wave of innovation in the enterprise AI landscape.

We value your feedback

Dell Technologies and the authors of this document welcome your feedback on the solution and the solution documentation. Contact the Dell Technologies Solutions team by email.

Chapter 9 References

The following links provide additional information about the solution design and components in this guide.

Dell Technologies documentation

The following Dell Technologies documentation provides additional and relevant information.

- Dell Generative AI Solutions (Dell.com/ai)
- Dell Info Hub for AI
- White Paper – Generative AI in the Enterprise
- Design Guide – Generative AI in the Enterprise - Inferencing
- GPU Optimization with Run:ai Atlas

NVIDIA documentation

The following NVIDIA documentation provides additional and relevant information:

- What is NeMo?
- NVIDIA AI Enterprise documentation
- Optimizing Inference on Large Language Models with NVIDIA TensorRT-LLM, Now Publicly Available
- Blog - Selecting Large Language Model Customization Techniques
- NeMo User Guide

Other documentation

The following documentation provides additional and relevant information:

- Dolly Dataset
- Llama 2: Open Foundation and Fine-Tuned Chat Models
- Meta's Responsible Use Guide for Llama 2

Appendix A Sample Scripts

This appendix contains the sample scripts we used for the validation efforts described in Chapter 5.

Supervised Fine Tuning with Llama 2 7B

The following example is of the Slurm batch file that we used for validation of SFT with Llama 2 7B:

```
#!/bin/bash

# Parameters
#SBATCH --error=%j.err
#SBATCH --gpus-per-node=8
#SBATCH --job-name=singlenode-sft
#SBATCH --nodes=1
#SBATCH --nodelist=node015
#SBATCH --ntasks-per-node=8
#SBATCH --output=%j.out
#SBATCH --partition=slr-9680-hdr
#SBATCH --time=6-00:00:00

TRAIN_DS="[/dataset/data/training.jsonl]"
VALID_DS="[/dataset/data/validation.jsonl]"
TEST_DS="[/dataset/data/test.jsonl]"
VALID_NAMES="[databricks-dolly-15k]"
CONCAT_SAMPLING_PROBS="[1.0]"
TP_SIZE=2
PP_SIZE=1
MODEL="/models/llama2-7b-bf16-tp1.nemo" #llama2-13b-bf16-tp4.nemo" llama2-7b-bf16-
tp1.nemo

export CUDA_LAUNCH_BLOCKING=1
module load docker/20.10.25

srun --container-mounts=/powerscale-
share/prem_sft/results/${SLURM_JOBID}/:/results,/dev/infiniband/:/dev/infiniband,/
powerscale-share/llm/datasets/databricks-dolly-15k/:/dataset,/powerscale-
share/llm/models/NeMo-models/llama2-NGC-v2.0/:/models --container-
image="docker://nvcr.io/ea-bignlp/ga-participants/nemofw-training:23.08.03" bash -
c "python /opt/NeMo/examples/nlp/language_modeling/tuning/megatron_gpt_sft.py \
trainer.precision=bf16 \
trainer.devices=8 \
trainer.num_nodes=1 \
trainer.val_check_interval=0.5 \
```

```
trainer.max_steps=50 \
model.restore_from_path=${MODEL} \
model.micro_batch_size=1 \
model.global_batch_size=128 \
model.tensor_model_parallel_size=${TP_SIZE} \
model.pipeline_model_parallel_size=${PP_SIZE} \
model.megatron_amp_O2=True \
model.sequence_parallel=True \
model.activations_checkpoint_granularity=selective \
model.activations_checkpoint_method=uniform \
model.optim.name=distributed_fused_adam \
model.optim.lr=5e-6 \
model.answer_only_loss=True \
model.data.train_ds.file_names=${TRAIN_DS} \
model.data.validation_ds.file_names=${VALID_DS} \
model.data.test_ds.file_names=${TEST_DS} \
model.data.train_ds.concat_sampling_probabilities=${CONCAT_SAMPLING_PROBS} \
model.data.train_ds.max_seq_length=2048 \
model.data.validation_ds.max_seq_length=2048 \
model.data.train_ds.micro_batch_size=1 \
model.data.train_ds.global_batch_size=128 \
model.data.validation_ds.micro_batch_size=1 \
model.data.validation_ds.global_batch_size=128 \
model.data.test_ds.micro_batch_size=1 \
model.data.test_ds.global_batch_size=256 \
model.data.train_ds.num_workers=0 \
model.data.validation_ds.num_workers=0 \
model.data.test_ds.num_workers=0 \
model.data.validation_ds.metric.name=loss \
model.data.test_ds.metric.name=loss \
exp_manager.exp_dir=/results \
exp_manager.create_checkpoint_callback=True \
exp_manager.checkpoint_callback_params.save_nemo_on_train_end=True"
```

LoRA with Llama 2 70B on two nodes

The following example is of the Slurm batch file that we used for validation of LoRA with Llama 2 70B:

```
#!/bin/bash

# Parameters
#SBATCH --error=%j.err
#SBATCH --gpus-per-node=8
#SBATCH --job-name=peft-lora
#SBATCH --nodes=2
#SBATCH --nodelist=node0[12-13]
#SBATCH --ntasks-per-node=8
#SBATCH --output=%j.out
#SBATCH --partition=defq
#SBATCH --time=6-00:00:00

TRAIN_DS="[/datasets/training.jsonl]"
VALID_DS="[/datasets/validation.jsonl]"
TEST_DS="[/datasets/test.jsonl]"
VALID_NAMES="[databricks-dolly-15k]"
RESTORE_PATH="/home/user/helix_output/"
CONCAT_SAMPLING_PROBS="[1.0]"
TP_SIZE=8
PP_SIZE=1
MODEL="/models/llama2-70b-bf16.nemo" #"/models/llama2-7b-bf16-tp1.nemo"
#"/models/llama2-13b-bf16-tp4.nemo"  #   #/models/llama2-70b-bf16.nemo

export HYDRA_FULL_ERROR=1
export NCCL_IB_HCA=mlx5_0,mlx5_3,mlx5_10,mlx5_11,mlx5_4,mlx5_5,mlx5_6,mlx5_9
export NCCL_IBEXT_DISABLE=1
export NCCL_DEBUG=INFO
export NCCL_IGNORE_CPU_AFFINITY=1

export MASTER_PORT=$(expr 10000 + $(echo -n $SLURM_JOBID | tail -c 4))
export WORLD_SIZE=$(($SLURM_NNODES * $SLURM_NTASKS_PER_NODE))
echo "WORLD_SIZE="$WORLD_SIZE

master_addr=$(scontrol show hostnames "$SLURM_JOB_NODELIST" | head -n 1)
export MASTER_ADDR=$master_addr
echo "MASTER_ADDR="$MASTER_ADDR

module load docker
module load cuda-dcgm/3.1.3.1
module load cuda12.2/toolkit/12.2.1

srun --container-
mounts=/dev/infiniband/:/dev/infiniband,/home/user/:/workspace,/powerscale-
share/llm/models/NeMo-models/llama2-NGC-v2.0/:/models,/powerscale-
```

```
share/llm/docker_tmp/tmp70b/:/tmp,/powerscale-
share/llm/docker_tmp/var70b:/var,/powerscale-
share/user/results/${SLURM_JOBID}:/results,/powerscale-
share/llm/datasets/databricks-dolly-15k/data/:/datasets --container-
image="nvcr.io/ea-bignlp/ga-participants/nemofw-training:23.08.03"  bash -c
"export CUDA_VISIBLE_DEVICES=0,1,2,3,4,5,6,7 && nvidia-smi topo -m && nvidia-smi
&& python
/opt/NeMo/examples/nlp/language_modeling/tuning/megatron_gpt_peft_tuning.py \
trainer.devices=8 \
trainer.num_nodes=2 \
trainer.precision=bf16 \
trainer.val_check_interval=30 \
trainer.max_steps=50 \
model.megatron_amp_O2=False \
++model.mcore_gpt=True \
exp_manager.create_wandb_logger=False \
exp_manager.resume_if_exists=True \
exp_manager.explicit_log_dir=/results \
exp_manager.resume_ignore_no_checkpoint=True \
exp_manager.create_checkpoint_callback=True \
exp_manager.checkpoint_callback_params.monitor=validation_loss \
exp_manager.checkpoint_callback_params.save_best_model=False \
exp_manager.checkpoint_callback_params.save_nemo_on_train_end=True \
model.tensor_model_parallel_size=${TP_SIZE} \
model.pipeline_model_parallel_size=${PP_SIZE} \
model.micro_batch_size=1 \
model.global_batch_size=8 \
model.restore_from_path=${MODEL} \
model.data.train_ds.num_workers=0 \
model.data.validation_ds.num_workers=0 \
model.data.test_ds.num_workers=0 \
model.data.train_ds.file_names=${TRAIN_DS} \
model.data.train_ds.concat_sampling_probabilities=[1.0] \
model.data.validation_ds.file_names=${VALID_DS} \
model.peft.peft_scheme='lora' \
model.data.train_ds.max_seq_length=2048 \
model.data.validation_ds.max_seq_length=2048" #\
# model.optim.lr=0.001 \
# model.peft.p_tuning.virtual_tokens=10"
```