AI without GPUs: A Reference Architecture for VMware Private AI with Intel

VMware AI/ML

Table of Contents

AI without GPUs: A Reference Architecture for VMware Private AI with Intel

Introduction

Executive Summary

In today's rapidly evolving digital landscape, the demand for AI Everywhere is on the rise, and organizations are looking for optimal solutions to support their AI workloads. VMware Private AI™ with Intel supports Xeon 4th Gen CPUs with Advanced Matrix Extensions (AMX) and VMware® Cloud Foundation™ offers a comprehensive and scalable collaboration for unlocking AI Everywhere. This collaboration combines the latest advancements in CPU technology, software-defined architecture, and AI-optimized hardware to deliver unparalleled performance, security, and scalability.

Generative artificial intelligence (GenAI), especially in the form of Large Language Models (LLMs), is at the forefront of technological innovation, offering human-like creativity, reasoning, and language understanding. Organizations across the globe recognize its potential, but implementing LLMs, particularly in regulated industries, brings about unique challenges. On-premises deployment in the private cloud offers a strategic solution, allowing organizations to retain complete control over data and comply with industry regulations. This fosters trust and transparency, ensuring that sensitive information and Intellectual property are securely protected within enterprise environments.

IT organizations can now use VMware Private AI™ with Intel for running GenAI models. This architectural approach for AI services enables privacy and control of corporate data, choice of open-source and commercial AI solutions, quick time-to-value, and integrated security and management.

Utilizing VMware Private AI™ with Intel, we can democratize GenAI for all enterprises and providing the following advantages:

- Get the flexibility to run a range of AI solutions for your environment: Intel, open-source, and independent software vendors.
- Deploy with confidence, knowing that VMware has partnerships with Intel and other partners, all of whom are respected technology leaders.
- Achieve great performance in your model with vSphere and VMware Cloud Foundation's Intel Xeon 4th Gen CPU integrations.
- Augment productivity by building private chatbots, eliminating redundant tasks, and building intelligent process improvement mechanisms.
- By leveraging the latest advancements in CPU technology, Intel Xeon 4th Gen CPUs with AMX offer a cost-effective solution for AI Everywhere, reducing the need for specialized AI hardware.

This Reference Architecture serves as an essential guide, providing insights into the architecture design, implementation, and best practices for LLM fine-tuning and inference. By embracing GenAI through the on-premises deployment of LLMs on VMware infrastructure, enterprises can unlock the full potential of this revolutionary technology in a responsible and ethical manner, significantly boosting innovation and driving sustainable growth.

Document Scope and Objectives

This Reference Architecture offers readers a detailed approach to implementing best in class AI infrastructure that leverages VMware Private AI™ with Intel for Generative AI workloads. A key focus of this Reference Architecture is to provide effective guidance on preparing, deploying, and automating virtual infrastructures fit for LLM fine-tuning and inference tasks. These tasks normally require the availability of hardware accelerators, such as GPUs from Virtual Machines (VMs) or containers. With the introduction of Intel Xeon 4th Gen Scalable CPUs, customers can unleash the power of the built-in AMX accelerator in every CPU core, allowing customers to tap into the latest in CPU advancements and extend these services to all workloads.

Additionally, this Reference Architecture presents real world examples of LLM fine-tuning and inference as the means for testing and validating the platform. The goal is to establish a scalable, high-performance, and production-ready architecture for GenAI tailored to meet the demands of enterprise-level applications.

Whether you are a seasoned AI practitioner or just beginning your journey of GenAI, this Reference Architecture serves as a technical resource, providing step-by-step instructions, best practices, and real-world insights that will help you run GenAI projects on top of VMs or Tanzu Kubernetes hosted by VMware Cloud Foundation and vSphere with Tanzu. By following the guidelines

presented in this Reference Architecture, you can confidently embark on the path toward AI Everywhere within your Data Center, developing state-of-the-art GenAI solutions that drive innovation and help deliver measurable business outcomes for your organization and your customers.

Architecture Design

High-level VMware Private AI™ Architecture Overview

A high-level overview of the VMware Private AI™ is depicted below starting with the infrastructure components up to the LLM application layer and models. This architecture represents the broader Private AI strategy that VMware announced last year at VMware Explore '23. We will address a more specific architecture that was developed in collaboration with Intel and advocate for the combined hardware and software ecosystem to help customers realize AI Everywhere.

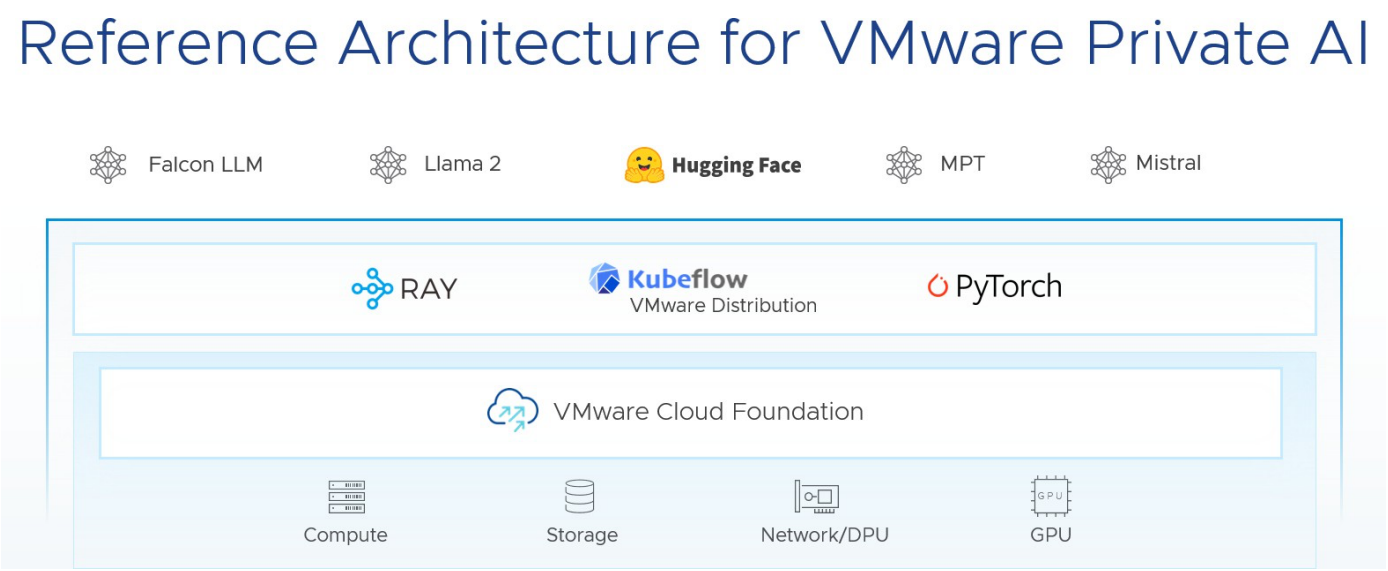


Figure 1: Solution High-Level Architecture Overview

Figure 1 illustrates the stacking of the software components used in LLM Operations starting from the base platform and all the way up to the AI workload.

Customers can leverage the solution for two primary LLM workflows – fine-tuning of a model and inferencing at scale. Both workflows demand more computing capacity than traditional deep learning workloads regarding LLMs. Inference also requires accelerated resources depending on application needs or number of users. By leveraging VMware Cloud Foundation and Intel 4th Generation Xeon processors with AMX, customers can unleash GenAI applications everywhere they run VMware Cloud Foundation and Intel CPUs in their Data Centers. The use of mainstream systems that are well known and scale efficiently in data centers allows customers to drive TTV (Time-to-Value) because they aren't having to rationalize the use of GPU accelerated systems as they explore and cut their teeth on AI use cases like GenAI. Leveraging accelerated CPU also enables customers to bypass the Data Center logistics to achieve GPU provisioning at scale until it is deemed necessary. Let's explore how Private AI with Intel is bringing GenAI workloads to the Enterprise space.

VMware Private AI™ with Intel

VMware Private AI™ with Intel represents a strategic fusion of hardware and software expertise tailored to meet the evolving demands of AI deployments within organizations. Anchored by VMware Cloud Foundation and fortified with cutting-edge acceleration capabilities with Intel AMX instructions, this architectural framework empowers enterprises to harness the transformative potential of AI while safeguarding data privacy and ensuring regulatory compliance. By leveraging VMware's strategic partnerships with leading AI providers, deployments are bolstered with confidence and reliability.

VMware Private AI™ with Intel is a collaboration between VMware and Intel to deliver a jointly validated AI stack that enables customers to use their existing general-purpose VMware and Intel infrastructure and open-source software to simplify building and deploying AI models. The combination of VMware Cloud Foundation and Intel's AI software suite coupling Intel Xeon processors with built-in AI accelerators delivers a validated and benchmarked AI stack for model fine-tuning and inferencing to accelerate

scientific discovery and enrich business and consumer services. This solution helps customers develop AI solutions faster and with a focus on getting workloads into production by providing a flexible, secure, and high-performance infrastructure for AI model training and deployment. The solution is designed to reduce total cost of ownership and address concerns of environmental sustainability by enabling the fine-tuning of task-specific models in minutes to hours versus days and weeks, and the inference of large language models using the customer's private corporate data. The solution is supported by several best-in-class server OEM companies that support 4th Gen Xeon CPUs with Intel Advanced Matrix Extensions (Intel AMX) and VMware by Broadcom is enabling these CPU features as a native component of the VMware Cloud Foundation stack so customers can focus on solving their business problems and focus less on the infrastructure enablement and hardware/software alignment.

Below you will find a VMware Cloud Foundation architecture that outlines the power of the solution and its components. One key attribute to take note of is the ability to run traditional virtual machines alongside Kubernetes workloads, both of which can support Intel's integrated acceleration with AMX. This provides customers the flexibility to maintain existing workloads as they move to a more cloud native approach and enables them to maintain the resiliency and high availability they have grown to trust with VMware products.

A couple of other key benefits are:

- Simplified management
- Enhanced scalability
- Improved security
- Optimized resource usage

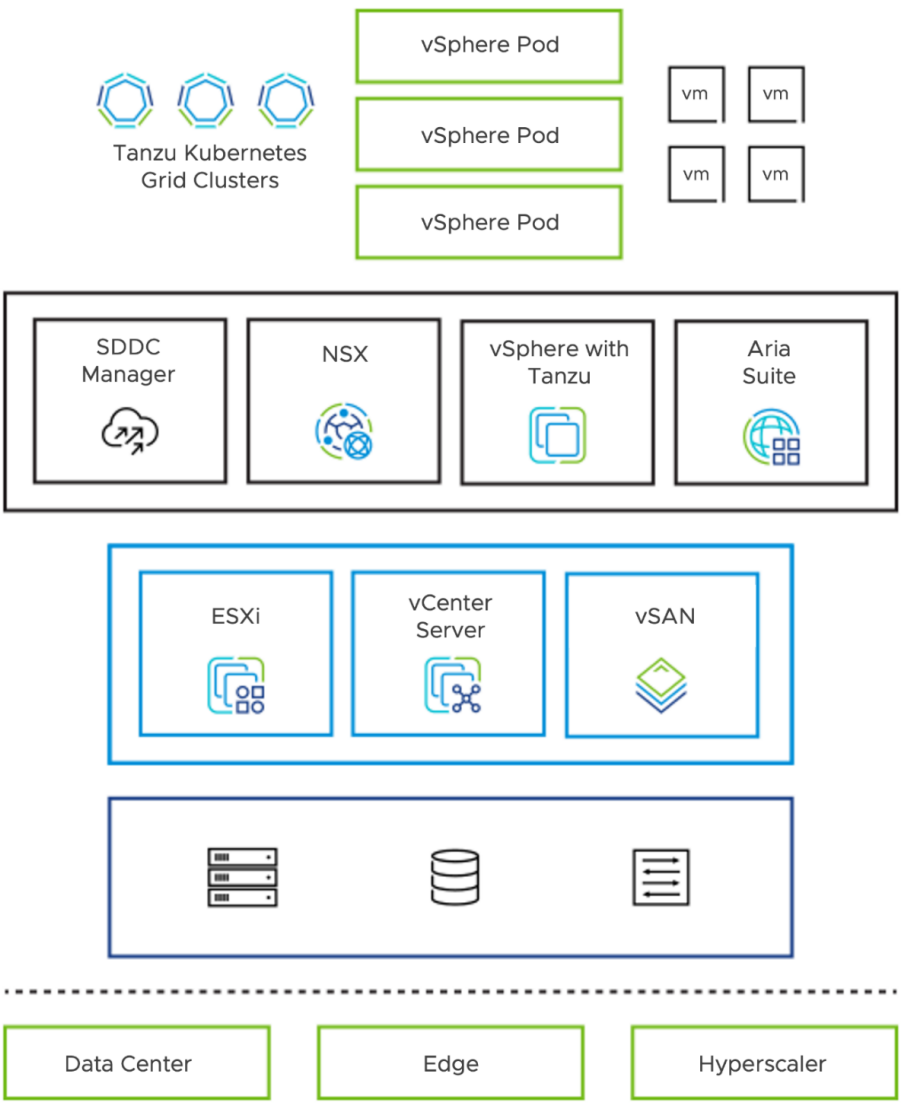


Figure 2: VMware Cloud Foundation Architecture

VMware vSphere IaaS Control Plane Overview

VMware vSphere IaaS control plane transforms traditional virtualization infrastructure into a robust platform for containerized workloads. VMware Tanzu Kubernetes Grid™ facilitates the creation and management of Tanzu Kubernetes Cluster (TKC) natively within vSphere, seamlessly integrating Kubernetes capabilities with the reliable features of vSphere. With vSphere's networking, storage, security, and VMware vSphere High Availability (vSphere HA) features, organizations achieve better visibility and operational simplicity for their hybrid application environments.

Table 1: VMware vSphere AI Capabilities

vSphere with Tanzu – AI/ML Capabilities	Description	
Intel AMX support	VMware vSphere 8 and Tanzu now support AMX instruction sets in the Tanzu Kubernetes Resources (TKR) out of the box. Making it fast and easy to spin up Tanzu Kubernetes Clusters with AMX enabled CPU workers.	
Device groups	In VMware Cloud Foundation 5.1.1, device groups enable admins to group certain hardware devices like Network Interface Card (NIC) and accelerated CPUs to enhance the performance. Compatible vendor device drivers are required and subject to vendor release.	
Simplified hardware consumption with device groups	Device groups leverage the existing **Add New PCI Device** to VM workflows. VMware vSphere Distributed Resource Scheduler™ (DRS) and vSphere HA are aware of device groups and will place VMs appropriately to meet the device groups' requirements.	

As mentioned earlier, vSphere IaaS control plane also helps organizations to run application components in containers or VMs on a unified platform, streamlining workload management and deployment. By converging containers and VMs, IT teams can leverage the benefits of both paradigms while maintaining a cohesive and scalable infrastructure.

Now let's take a deep look at the VMware Private AI™ with Intel architecture and ecosystem.

VMware Private AI™ with Intel Architecture

VMware Private AI™ with Intel is a robust suite designed to boost AI workloads in various industries. In partnership with VMware, Intel delivers the virtual drivers and AMX instruction sets that enable the Kubernetes Resources and Virtual Machines for optimal performance of AI workloads running on VMware Cloud Foundation and the TKG Service.

The architecture below calls out the Intel AI software suite that enables customers to quickly adopt optimized tools and frameworks to help them kickstart their AI efforts. But before we talk more about the platform it makes sense to understand what a GenAI stack is made up of and what each element represents to help deliver AI solutions.

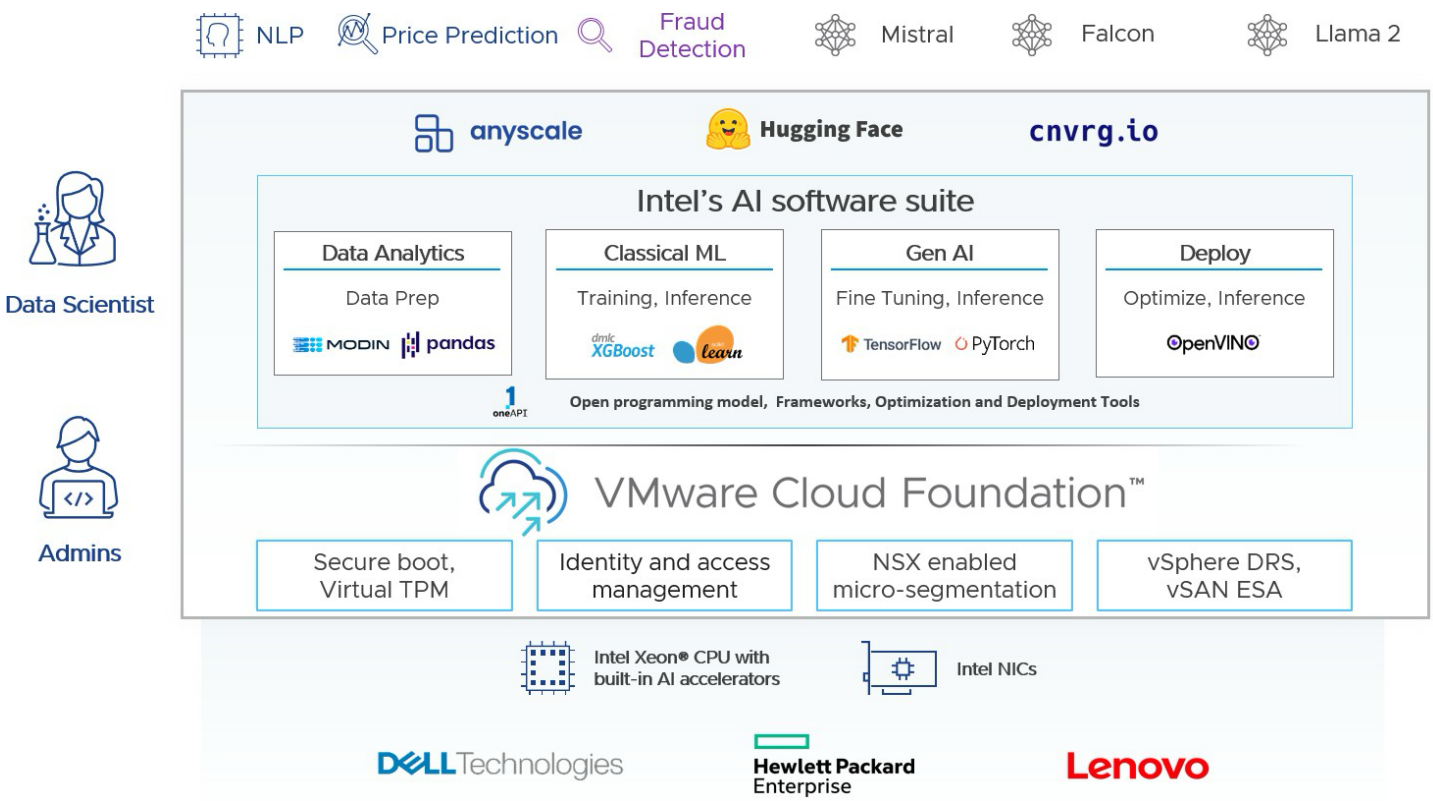


Figure 3: VMware Private AI™ Foundation **with Intel on VMware Cloud Foundation with Tanzu**

Generative AI Stack Overview

LLMs have revolutionized natural language processing (NLP) tasks, enabling machines to understand, generate, and interact with human language in a remarkably human-like manner. These models, such as Llama-2 & 3, GPT-4, MPT, Mistral and Falcon have gained popularity due to their ability to process vast amounts of text data and produce coherent and contextually relevant responses. Behind the scenes, these models rely on intricate operations, components, and processes that work together harmoniously to achieve their impressive capabilities. The main elements of LLM operations can be summarized as follows:

- **Deep Learning (Transformers) Neural Nets**: LLMs are built upon complex neural network architectures based on the transformer architecture. These models consist of multiple layers of self-attention mechanisms and feed-forward neural networks with billions of neurons and parameters that need to get trained over terabytes of data.

- **Hardware accelerators:** LLMs are very computationally demanding and require specialized hardware to achieve optimal performance. LLM training and inference processes often rely on high-performance in-chip accelerators that support AMX, RDMA networking and high-speed storage to handle immense computational loads. Most of this reference architecture provides detailed instructions on how to set up these acceleration technologies from VMware Cloud Foundation and Tanzu Kubernetes so you can run LLM workloads on VMs and containers without needing a dedicated GPU/s.

- **ML software stack**: Multiple open-source software choices provide tools to work with LLMs and GenAI. For this reference architecture, we have strived to make sound selections of open-source ML software that can help our customers in their journey to adopt AI as a central piece of their app development. The main open-source ML software components used in this document are:

  - Hugging Face ▫ Transformers. *Hugging Face* (HF) is a popular platform where the ML community collaborates on models, datasets, and applications. They are authors of one of the most adopted opensource *PyTorch* implementations of the NLP Transformers architecture. In our examples, we use many of their technologies which dramatically simplifying the LLM fine-tuning process.

  - Kubeflow is an end-to-end ML platform for Kubernetes; it provides components for each stage in the ML lifecycle, from exploration to training and deployment.

  - Intel optimized frameworks: Intel provides optimizations to extract better performance on Intel HW through *Intel Extensions for Pytorch (IPEX)* and quantization through the *Intel Neural Compressor*.

- **Pre-training tasks:** The first stage of LLM development involves pre-training on massive amounts of text data from the

internet. During this phase, the model learns to predict the next word in a sentence given the context of the preceding words. This process helps the model build a foundation of language understanding and grammar. The *Hugging Face Models* repository provides access to over 285k language and computer vision ML models that can be used for many types of tasks. Pre-training LLMs is a difficult and expensive task. For instance, pre-training the *Falcon-40B LLM* required 384 x A100 40GB GPUs running in P4d (AWS) instances and took over 2 months and based on data shared by Meta, it took 1.5M GPU hours to train the recently released Llama3 8b model and they used over 24,000 H100 80GB GPUs from NVIDIA. Given the high cost and complexity of pre-training tasks, it is more convenient to leverage an open source pre-trained LLM that works with the licensing permissions (commercial, research, and others) and the use cases you have in mind.

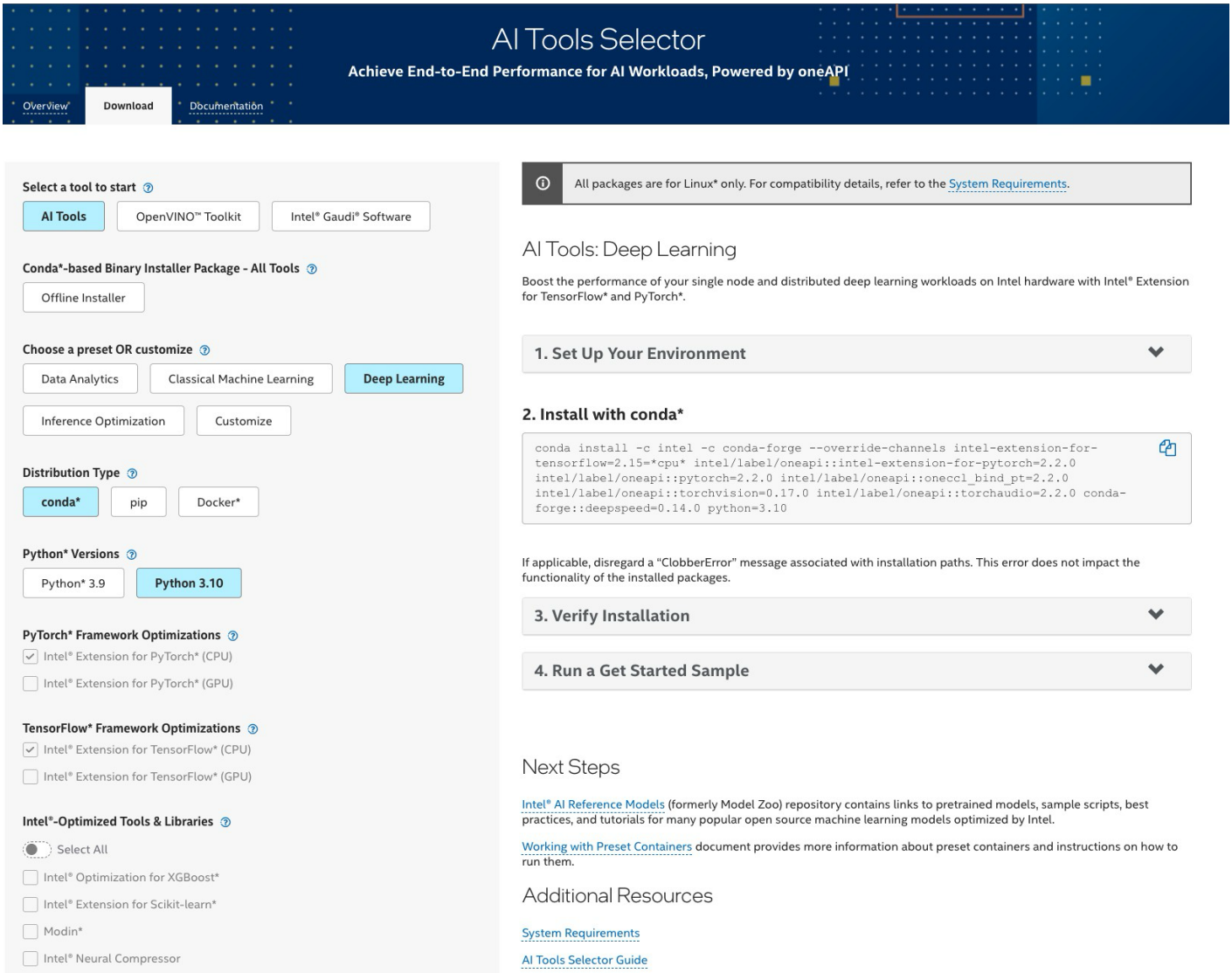
- Fine-tuning tasks: After pre-training, the model can be fine-tuned on specialized datasets for specific tasks. This process adapts the general language model to perform more specialized tasks such as text generation, translation, sentiment analysis, or question-answering. Fine-tuning is crucial to tailoring the model's capabilities to the desired application. Domain specific models enable organizations to achieve business outcomes faster than trying to create their own model and are typically easier to host on-premises because they can be smaller. In the later sections of this document, we provide a complete example of how to fine-tune the Llama-2-7b-chat LLM with an open-source dataset that is specific to finance (finance-alpaca).

- **Inference (prompt completion) tasks:** After the LLM is pre-trained and fine-tuned, it enters the inference stage, where it processes users' prompts and generates completions in real-time. The model utilizes the previously learned information to make predictions and generate coherent and contextually relevant text. Later in this document we provide a full example on how to serve (inference) prompt completions using a fine-tuned model and a chatbot frontend that leverages the fine-tuned model mentioned above.

- **Ethical Considerations:** As LLMs become more powerful, ethical concerns about their potential misuse and bias have gained prominence. Efforts are being made to address issues related to fairness, transparency, and responsible AI practices in language model deployment. For more information, refer to How to Build Trustworthy AI with Open Source

Intel AI Software Suite & Tool Selector

One of the challenges customers face when developing AI related applications is compiling a flexible and standard set of tools that are optimized to maintain performance and allow a fast and easy path to consuming these resources to help with governance and consistency. Intel's AI software suite is packaged with end-to-end open-source software and optional licensing components to enable developers to run full AI pipeline workflows from data preparation to fine-tuning to inference, accelerate building multi-node scaling, and deploying AI on enterprise IT infrastructure. The open oneAPI framework enables processors and hardware
accelerator-agnostic software development, allowing developers to write code once and run it across different architectures, eliminating the need for multiple code bases and specialized languages. Intel's Transformer Extensions (ITEX) and PyTorch Extension's (IPEX) deep integration with developers favorite open-source Hugging Face libraries provide automated optimization recipes for fine-tuning and compressing models for efficient inference. To get started download the AI Tools from *AI Tool Selector.*


AI Tools Selector
Achieve End-to-End Performance for AI Workloads, Powered by oneAPI
Overview
Download
Documentation
Select a tool to start
AI Tools
OpenVINO™ Toolkit
Intel® Gaudi® Software
Conda*-based Binary Installer Package - All Tools
Offline Installer
Choose a preset OR customize
Data Analytics
Classical Machine Learning
Deep Learning
Inference Optimization
Customize
Distribution Type
conda*
pip
Docker*
Python* Versions
Python* 3.9
Python 3.10
PyTorch* Framework Optimizations
Intel® Extension for PyTorch* (CPU)
Intel® Extension for PyTorch* (GPU)
TensorFlow* Framework Optimizations
Intel® Extension for TensorFlow* (CPU)
Intel® Extension for TensorFlow* (GPU)
Intel®-Optimized Tools & Libraries
Select All
Intel® Optimization for XGBoost*
Intel® Extension for Scikit-learn*
Modin*
Intel® Neural Compressor
All packages are for Linux* only. For compatibility details, refer to the System Requirements.
AI Tools: Deep Learning
Boost the performance of your single node and distributed deep learning workloads on Intel hardware with Intel® Extension for TensorFlow* and PyTorch*.
1. Set Up Your Environment
2. Install with conda*
conda install -c intel -c conda-forge --override-channels intel-extension-for-tensorflow=2.15=*cpu* intel/label/oneapi::intel-extension-for-pytorch=2.2.0 intel/label/oneapi::pytorch=2.2.0 intel/label/oneapi::oneccl_bind_pt=2.2.0 intel/label/oneapi::torchvision=0.17.0 intel/label/oneapi::torchaudio=2.2.0 conda-forge::deepspeed=0.14.0 python=3.10
If applicable, disregard a "ClobberError" message associated with installation paths. This error does not impact the functionality of the installed packages.
3. Verify Installation
4. Run a Get Started Sample
Next Steps
Intel® AI Reference Models (formerly Model Zoo) repository contains links to pretrained models, sample scripts, best practices, and tutorials for many popular open source machine learning models optimized by Intel.
Working with Preset Containers document provides more information about preset containers and instructions on how to run them.
Additional Resources
System Requirements
AI Tools Selector Guide

Physical Infrastructure Design

When it comes to AI/ML workloads, the hardware infrastructure requirements can vary depending on the specific task, dataset size, model complexity, or performance expectations. However, there are some general recommendations for hardware infrastructure for AI/ML workloads shown in the following table.

Table 2: Physical Infrastructure Design

Category	Hardware	Description	Example of Optimal Configuration
	Intel *VMware Compatibility Guide - Intel Xeon*		2 x Intel Xeon 6448H (Sapphire Rapids) CPU with 32 cores. 2.4 GHz Base, 4.1 GHz Max Turbo Or… 2 x Intel Xeon 8462Y+ (Sapphire Rapids) CPU with 32 cores. 2.8 GHz Base, 4.1 GHz Max Turbo Or… 2 x Intel Xeon 8490H (Sapphire Rapids) CPU with 60 cores. 1.9 GHz Base, 3.5 GHz Max Turbo
			2TB RAM per node, depending on the configuration. Follow OEM recommendations when populating DIMM slots.
	VMware vSAN *VMware vSAN Hardware Quick Reference Guide*		For more information, refer to: ● *Design Considerations for Storage Controllers in vSAN* ● *Design Considerations for Flash Caching Devices in vSAN* ● *Consider NVMe or SAS SSD for vSAN Capacity devices* *vSAN Storage Design Considerations*
	Management Network *VMware Compatibility Guide - NICs*	10 Gb/s onboard NIC 25 Gb/s or above Ethernet NIC Host baseboard management controller (BMC) with RJ45.	NIC: Intel E810 Series *Intel E810 10/25GbE Details*
Network Adapter	Workload Network *VMware Compatibility Guide – NICs with SRIOV and RoCE v2*	LLM inference and fine-tuning within a single host (non-multinode) is compatible with standard 25 Gb Ethernet. For fine-tuning models larger than 40B parameters, efficient multi-node communication requires low latency and necessitates 100 Gb/s or higher RDMA network for optimal performance.	NIC: Intel E810 *Intel E810 100GbE Details*
	NAS/Object Storage *VMware Compatibility Guide – NAS*	For inferencing, it might be possible to store the model on a local storage. For fine tuning larger models, it might be necessary to use NAS or SAN or object storage solutions.	

Keep in mind that the configuration above provides optimal configuration for inferencing and fine-tuning LLMs and is in line with

the industry architectures being implemented today. The requirements for your environment might be different.

There is no one-size-fits-all solution when it comes to AI/ML hardware, as different tasks and projects may demand unique configurations. Organizations and individuals involved in AI/ML work should carefully analyze their specific requirements to determine the most suitable hardware setup, considering the computational power, memory capacity, storage capabilities, and networking resources necessary to achieve optimal results. Flexibility and scalability in hardware choices are crucial to meet the evolving demands of AI/ML workloads efficiently. Consult your *OEM* to determine the proper solution.

We recommend using the *VMware Compatibility Guide vSAN ESA Ready Node Configurator* to select a chassis with certified network cards and NVMe drives, then choose Sapphire Rapids CPUs and add more RAM. By selecting a chassis with vSAN ESA-certified components you will be able to deploy VMware Cloud Foundation and vSAN-ESA on your cluster. Also see *Support for ReadyNode Emulated Configurations in vSAN ESA*.

Hardware Configuration Considerations

Xeon 4th Gen Sapphire Rapids CPUs

For performance set up the BIOS following the **tuning recommendations of the hardware OEM**, with a few caveats:

- Be sure to leave virtualization enabled in the BIOS.
- If there are settings for performance of virtualized environments, turn them on.
- Leave Logical Processors enabled.
- If there are two almost identical performance settings and one is "per watt", use the one that is not "per watt" if you want to maximize performance. Use the "per watt" setting if you want to sacrifice some performance to save on power costs.
- If there are hardware settings for Sub NUMA Clustering (SNC), we have observed that leaving this feature disabled for smaller core count CPUs is ok. ESXi will handle NUMA affinity without any special BIOS settings. With higher core count CPUs (32core +) we have observed that setting SNC=2 can provide benefits and should be a design consideration when planning your implementation.

AMX Configuration Guidance in vSphere

Sizing an environment for GenAI is crucial for ensuring optimal performance and scalability, as it directly impacts the efficiency and effectiveness of AI-driven solutions. The right sizing approach involves determining the appropriate infrastructure resources, such as computing power, memory, and storage, to support the deployment and operation of GenAI models and applications.

One of the primary challenges in sizing an environment for GenAI lies in accurately estimating the computational and storage requirements of AI workloads. GenAI applications often involve complex deep learning models that demand significant computational resources, especially during training and inferencing phases. Estimating these requirements involves careful analysis of factors such as the size of the dataset, model architecture, training algorithms, and expected usage patterns.

Additionally, scalability is a critical consideration in sizing GenAI environments, as AI workloads may vary in intensity over time and across different use cases. Sizing for scalability involves designing a flexible infrastructure that can dynamically adjust resources to accommodate fluctuations in demand, ensuring consistent performance and responsiveness under varying workloads.

Furthermore, ensuring compatibility and integration with existing IT infrastructure poses another challenge in sizing GenAI environments. Integrating AI solutions into existing systems and workflows requires careful planning and consideration of factors such as data interoperability, security, compliance, and operational dependencies.

To learn more about configuration guidance for your environment and to gain some insight about performance numbers please refer to the following document.

Configuration Guidance for VMware Private AI with Intel

Network Adapter Configuration Consideration

The adapter type to be used in VMware stacks can be selected as follows with virtualization features and performance.

Table 10: Network Adapter Configuration

Category	Paravirtual		Passthrough		
Network adapter type	VMXNET3	PVRDMA	DirectPath IO	Dynamic DirectPath IO	SRIOV
vMotion/ Snapshot/Suspend & Resume supported?	Yes	Yes	No	No	No
HA/DRS supported?	Yes	Yes	No	Yes* (DRS initial placement in vSphere 7)	Yes* (DRS initial placement in vSphere 8U2)
Virtual Distributed Switch required?	Yes	Yes	No	No	Yes
Multi-VMs sharing?	Yes	Yes	No. Exclusive to 1 VM	Yes. But exclusive to 1 VM once powered on.	Yes
RDMA capability	No. TCP only	RoCE only	IB or RoCE	IB or RoCE	IB or RoCE
Tanzu VM Class supported?	Yes	No	No	Yes	Yes

We selected an VMXNet3 adapter for this Reference Architecture to ensure full HA and DRS support to allow customers the most resilient setup possible. If your workload requires lower-latency you could also decide to use PVRDMA and leverage RoCEv2 (RDMA over Converged Ethernet) or SR-IOV w/ RDMA. It is important to consider the caveats to each so please review the table above to help guide your decision making.

Network Design

Figure 4 presents the network design of the GenAI deployment on VMware Cloud Foundation, which is segmented into two domains: the management domain and the workload domain.

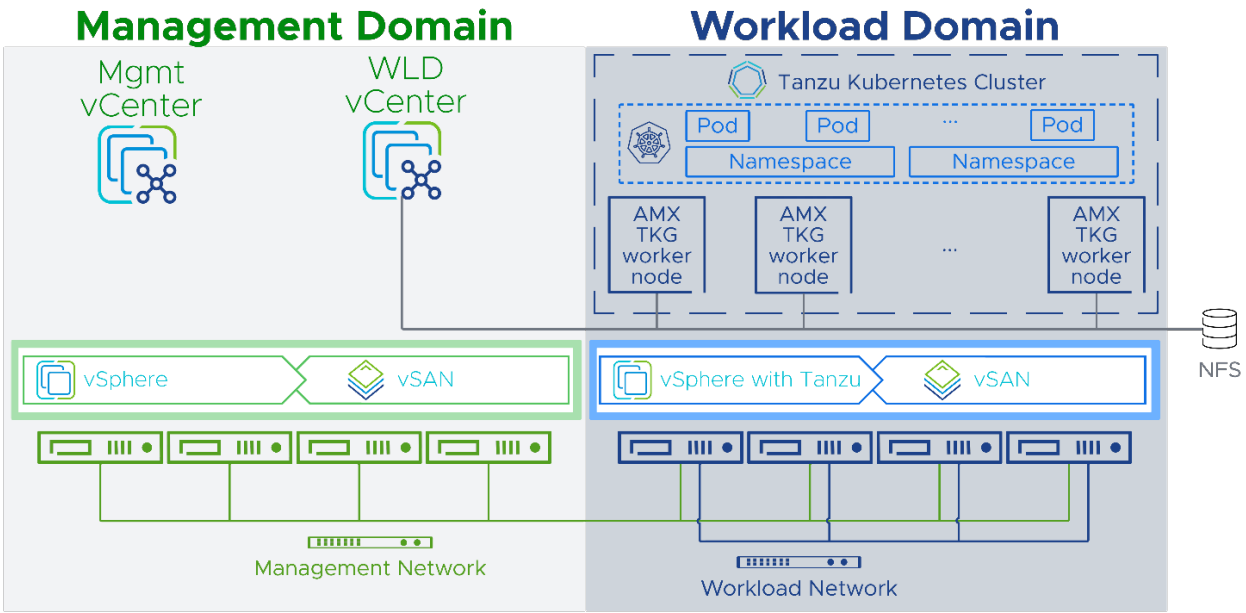


Figure 4: Network Design

The workload domain manages the ESXi hosts within the workload domain where the VMware Tanzu Supervisor Cluster resides, facilitating TKCs on top.

For the management network, each network type below in Table 3 is associated with a specific VLAN:

Table 3: Management Network

Network Type	Description	Comments
ESXi Management	ESXi management traffic	At least 1Gb/s
vSAN	vSAN traffic	vSAN ESA: at least 25 Gb/s
vMotion	vMotion traffic	At least 10Gb/s
VM	VM-to-VM traffic	Depends on use case or requirements
Tanzu Management	Supervisor Cluster traffic	This network can either be shared with the ESXi management port group or kept separate.

Given this architecture is covering only inference and fine-tuning, the management network's aggregate bandwidth at 25 Gbps is deemed satisfactory. We recommend using *Network I/O control* to reserve network bandwidth for the above different services. This approach helps mitigate potential conflicts or disruptions caused by other VMware services or VM traffic.

The workload network that resides on the workload cluster is configured with dedicated switch and network adapters for optimal performance.

Table 4: Workload Network

Network Type	Description	Comments
Tanzu Frontend	It provides load balanced virtual IP Ranges for TKCs	This network can either be shared with the Tanzu workload network or kept separate.
Tanzu Workload	It provides connectivity to TKCs in the vSphere Namespaces.	For LLM inferencing tasks and fine-tuning tasks within a single host, leveraging an existing network infrastructure featuring 25 Gb Ethernet is sufficient to accommodate the bandwidth requirements of textual data. For the requirement to fine-tune smaller (7b-15b) models the substantial demand for information exchange (including weights) necessitates the adoption of RDMA networking (RoCE) with 100 Gb or higher bandwidth for optimal performance.

For more information about how to design the vSphere with Tanzu infrastructure, refer to: *Developer Ready Infrastructure for VMware Cloud Foundation*.

Note: During the installation of NICs or HCAs (Host Channel Adapters) on servers, it is crucial to consider PCIe generation and lane compatibility on the servers' motherboard. This alignment ensures optimal data transfer speed for the devices. Neglecting this alignment can result in suboptimal performance and reduced number of PCIe lanes to achieve maximum throughput.

GenAI models can be sizable, with billions of parameters and intermediate outputs. Thus, the models require large and shared storage capacity. The need for external storage for AI model inference depends on the specific requirements and characteristics of the AI model or the deployment environment.

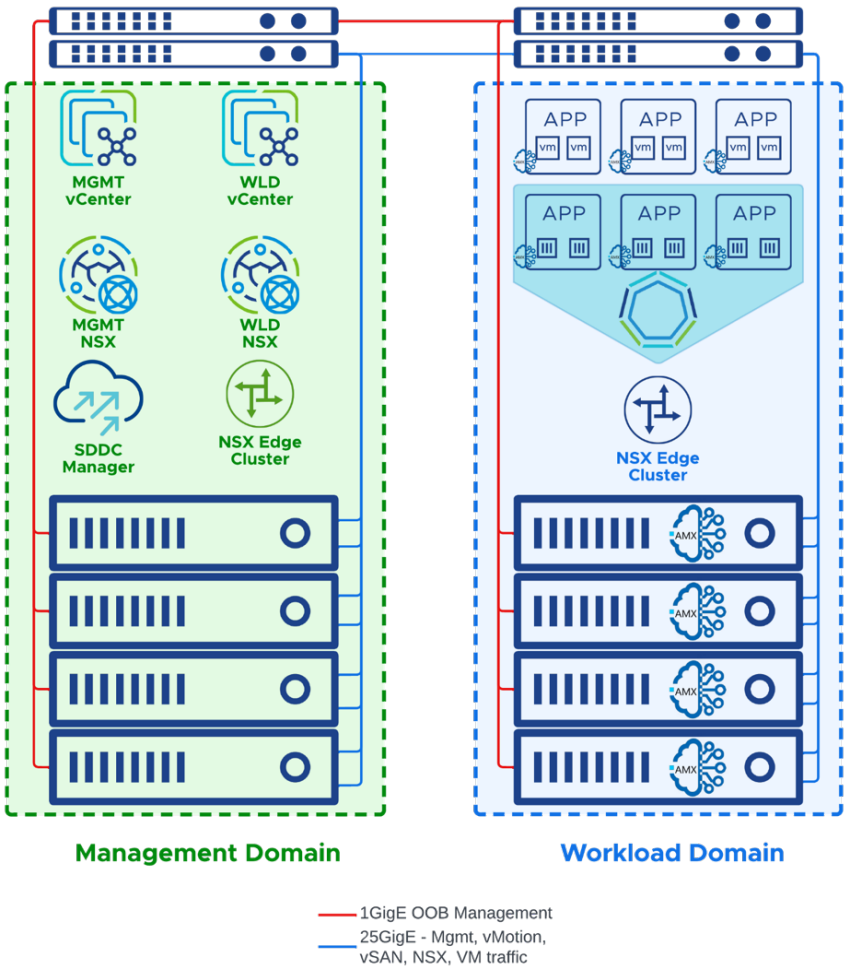
- External storage can be used as a repository for models, model versioning and management, model ensembles, and for storage and archival of inference data.
- For fine-tuning tasks, it is recommended to use a dedicated storage network to connect to a robust external storage to store the training data and intermediate outputs during training. This network helps to optimize data access and reduce latency for storage operations, which can offer the scale and speed necessary for operationalizing AI models, providing a foundational component for AI workflow. This also enables administrators to govern who, what, and how the data is being used to comply with

data governance and compliance.

VMware Cloud Foundation Deployment

VMware Cloud Foundation can be deployed in two architectures: Standard & Consolidated

The Standard Architecture consists of a Management Domain, which is an independent vSphere cluster with its own vCenter and NSX Manager cluster, where the management components will be physically separated from the workload VMs. The management components consist of SDDC Manager, all vCenter Servers for both management and workloads, NSX Manager clusters for both management and workloads, and NSX Edge cluster(s) for management only. Additional VI Workload Domains can be deployed consisting of their own compute cluster(s), NSX Management cluster(s) (hosted in the Management Domain), separate vCenter(s), and a separate NSX edge cluster(s), leveraging vSAN as the storage. Though vSAN is required for the Management Domain, Workload Domains support the option to leverage vSAN, vVols (vVols supports FC, NFS, and iSCSI), NFS, iSCSI or Fibre Channel as the shared storage in the cluster.



Each "domain" is a physically separate vSphere cluster. This ensures there is no resource contention for the management components of the environment.

This Reference Architecture focuses on the Standard Architecture with a VI Workload Domain consisting of Intel's 4th Generation Xeon processors with AMX. Every node in each domain has two physical 25GigE uplinks, cabled to two top of rack (ToR) switches to provide redundant links for high availability (HA), and a single out-of-band (OOB) management interface cabled to a separate switching infrastructure for OOB management. These two 25GigE links carry all the traffic for the vSphere hosts, as well as VM traffic, including GENEVE encapsulated traffic for the NSX overlay network. Since the VI Workload Domain is comprised of vSphere hosts with Intel's 4th Generation Xeon processors with AMX, every VM created in the Workload Domain can take advantage of the accelerator, whether it be in containers or virtual machines.

Software Resources

The table below demonstrates the software resources we used in our validation.

Table 8: Software Resources

Software	Purpose	Version
VMware Cloud Foundation	Efficiently manage VM and container workloads at scale. Deliver cloud benefits to on-premises, full-stack hyperconverged infrastructure (HCI) deployments.	5.1.1 *We upgraded vCenter Server to *8.0 Update 1b* by following *this doc* to avoid a known issue with vGPU and vMotion.
Tanzu Kubernetes Release (TKR)	A Tanzu Kubernetes release provides the Kubernetes software distribution signed and supported by VMware	v1.26.5---vmware.2-fips.1-hwekernel.1

VMware Environment Preparation

To properly prepare VMware environments, plan the following items:

- **Assessment and Planning**: Begin by assessing your organization's requirements, including computing resources, storage, networking, and application needs. Plan the virtual infrastructure, accordingly, considering factors such as scalability, redundancy, and disaster recovery. It is crucial to understand the workload demands and design a flexible architecture to accommodate future growth.
- **Hardware Selection and Compatibility***:* Choose hardware components that are compatible with VMware's virtualization platform. Ensure that the server hardware, storage devices, and networking equipment are on VMware's Hardware Compatibility List (HCL) to guarantee smooth operations and optimal performance.
- **Networking Setup***:* Create virtual networks and VLANs to segregate traffic and enhance security. Consider using VMware NSX-T provided by VMware Cloud Foundation or VMware's distributed switches for more streamlined network management and monitoring. Implement proper firewall configurations, configure network security policies, and isolate sensitive workloads as required.
- **Backup and Disaster Recovery:** Establish a robust backup and disaster recovery strategy to protect critical data and ensure business continuity. Consider utilizing VMware's Data Protection tools or integrate with third-party backup solutions.
- **Performance Monitoring and Optimization***:* Consider utilizing tools like vCenter Server and *VMware Aria Operations* to identify performance bottlenecks and make informed decisions to optimize resource utilization and maintain optimal performance.

VMware Cloud Foundation Environment Preparation summarizes the high-level steps of preparing the VMware environment with VMware Cloud Foundation.

VMware Cloud Foundation Environment Preparation

1. Familiarize with the VMware Cloud Foundation design and concepts. The *VMware Cloud Foundation Design Guide* contains requirements and recommendations for the design of each component of the SDDC.
2. Ensure all servers and network switches are supported and meet VMware Cloud Foundation requirements by consulting the *VMware Compatibility Guide*.
3. Based on the chosen design options, configure the network and services (DNS, DHCP, NTP), then complete the *VMware Cloud Foundation Planning and Preparation Workbook*.
4. Prepare the ESXi hosts for VMware Cloud Foundation by installing the appropriate ESXi version and configuring the systems. Details on how to install and configure the systems are found in the *Prepare ESXi Hosts for VMware Cloud Foundation* in the VMware Cloud Foundation Deployment Guide.
5. Deploy the VMware Cloud Foundation management domain by deploying the VMware Cloud Builder and using the VMware Cloud Foundation Planning and Preparation Workbook. Cloud Builder creates the management cluster, deploys the SDDC manager, then creates the vCenter and NSX-T for the management domain. More detail about the process can be found in the *VMware Cloud Foundation Deployment Guide*.
6. Once the VMware Cloud Foundation management domain deployment is complete, use the SDDC manager to add additional hosts and created an additional VI workload domain. More detail about the process of adding and creating VI

workload domains can be found in the *VMware Cloud Foundation Administration Guide*.

For those looking to help script or automate their VCF deployment, PowerShell scripts can be found using the following link: *https://vmware.github.io/power-validated-solutions-for-cloud-foundation/*

Virtual Infrastructure Design

Designing a VMware virtual infrastructure requires consideration of several factors to ensure optimal availability, manageability, performance, recoverability, and security (AMPRS), which are the key design factors of VMware's design methodology.

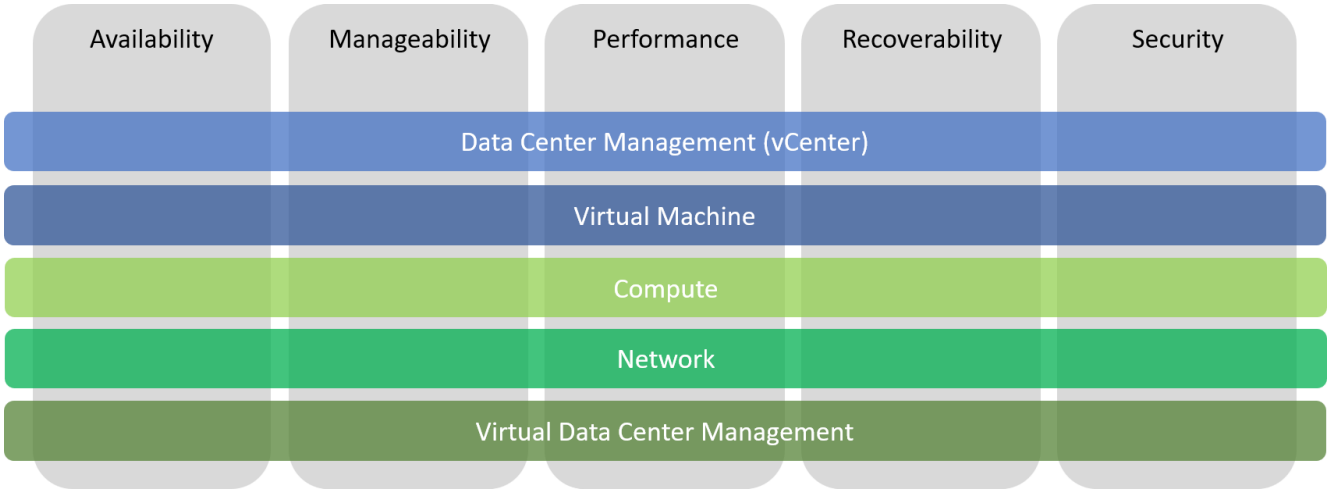


Figure 5: Virtual Infrastructure Design

Refer to our existing documentation, design guides and VMware Validated Solutions to help guide a design process that follows VMware's best practices, design decisions and operational procedures. For more information, see the following links *VMware Validated Solutions* and *Cloud Platform Tech Zone*.

Tanzu Kubernetes Grid Design

Tanzu Kubernetes Grid is a powerful and versatile solution that empowers organizations to seamlessly deploy and manage Kubernetes clusters on top of VMware vSphere. Tanzu Kubernetes Grid offers a consistent and scalable Kubernetes experience, simplifying the complexities of container orchestration, and enabling enterprises to focus on delivering value through their AI/ML workloads. Tanzu Kubernetes Grid provides a robust foundation for running AI/ML workloads at scale, leveraging Kubernetes' strengths in orchestrating distributed and resource-intensive applications.

In the AI/ML space, Tanzu Kubernetes Grid plays a critical role by providing a reliable and resilient platform for deploying and scaling AI/ML workloads and applications. With the explosive growth of data and the increasing demand for sophisticated AI-driven solutions, organizations require a stable and efficient infrastructure to process and analyze large amounts of information. Tanzu Kubernetes Grid ensures that AI/ML workloads can be seamlessly deployed and managed on Kubernetes clusters, taking advantage of Kubernetes' advanced capabilities for automating resource allocation, load balancing, and fault tolerance.

Verify the Tanzu Kubernetes Grid *documentation* before designing and implementing your solution. Understanding the Tanzu Kubernetes Grid design, planning, and implementation documentation is crucial as it provides a comprehensive understanding of its functionality and deployment best practices. The documentation outlines Tanzu Kubernetes Grid concepts, architecture components, security, deployment options, maintenance, and troubleshooting. Understanding the architecture helps in customizing and optimizing Tanzu Kubernetes Grid, ensuring security measures are in place, planning upgrades and migrations, and facilitating smooth integrations with other services.

Enabling vSphere IaaS Control Plane

Once you have VMware Cloud Foundation deployed, we recommend you enable vSphere IaaS control plane by following the steps in the following section.

Enabling vSphere IaaS Control Plane using VMware Cloud Foundation

In VMware Cloud Foundation deployments, it is recommended to follow the vSphere IaaS control plane solution workflow built into SDDC Manager to streamline the enabling of vSphere IaaS control plane over NSX-T based networks. This solution integration reliably and efficiently manages the complete lifecycle management of all VMware Cloud Foundation components within its framework that results in consistent operations across all deployments.

The list below summarizes the high-level steps to enable VMware Cloud Foundation with Tanzu Services using the Workload Management solution. Refer to *VMware Cloud Foundation for VMware Tanzu* in the *VMware Cloud Foundation Administration Guide* for more information.

1. *Create a Subscribed Content library*
   Please refer to the section below to get the appropriate repo URL.
2. *Enable Workload Management*
3. *View Workload Management cluster details*
4. *Update Workload Management license*

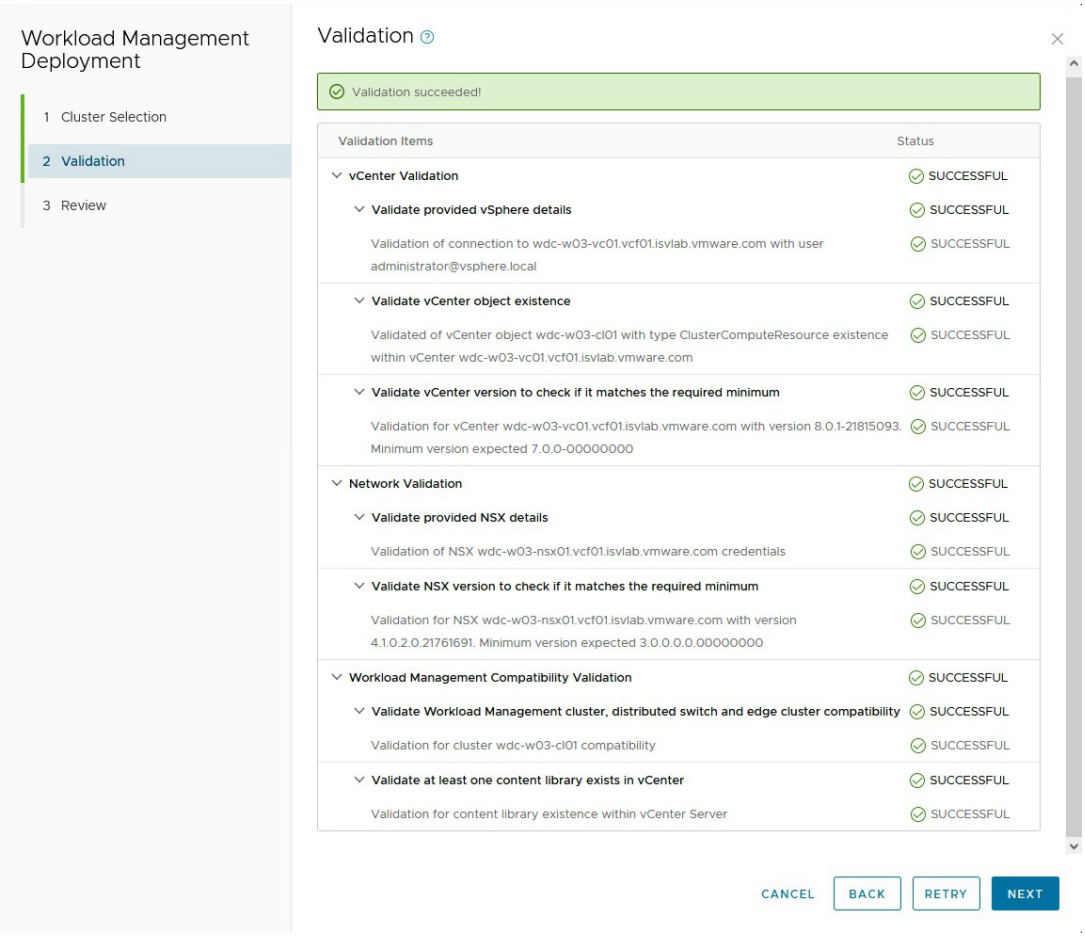


Figure 16: SDDC Manager Validating Tanzu Prerequisites

After you follow the Workload Management solution workflow to enable the vSphere IaaS control plane you can create a Kubernetes workload cluster that can run workloads that use AMX instructions.

Using Tanzu with AMX-CPUs

To take advantage of AMX instructions in Sapphire Rapids CPUs, Tanzu worker nodes have two requirements:

- Worker nodes must use Hardware Version 20 or higher (ESXi 8.0 and later).
- Worker nodes must be running a Linux kernel that supports AMX instructions, so either a kernel 5.15 or higher, or an earlier kernel that had the AMX instructions backported.

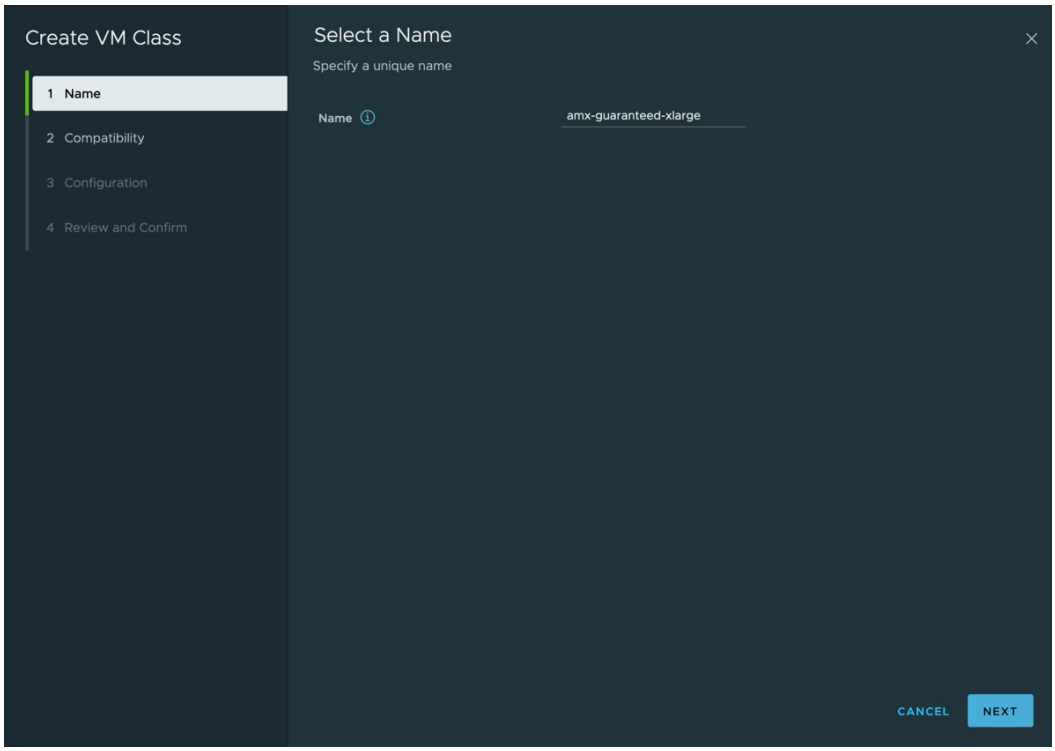
Which capabilities of the underlying hardware are virtualized in vSphere is determined by the hardware version (HW version) of the guest VM. The AMX instructions are virtualized in HW version 20, so if you want to access AMX instructions in vSphere you need to use HW version 20 on your VMs.

This reference architecture is based on vCenter 8.0.2 and ESXi 8.0u2, and ESXi 8.0u2 supports HW version 21, so that's what we'll

be using.

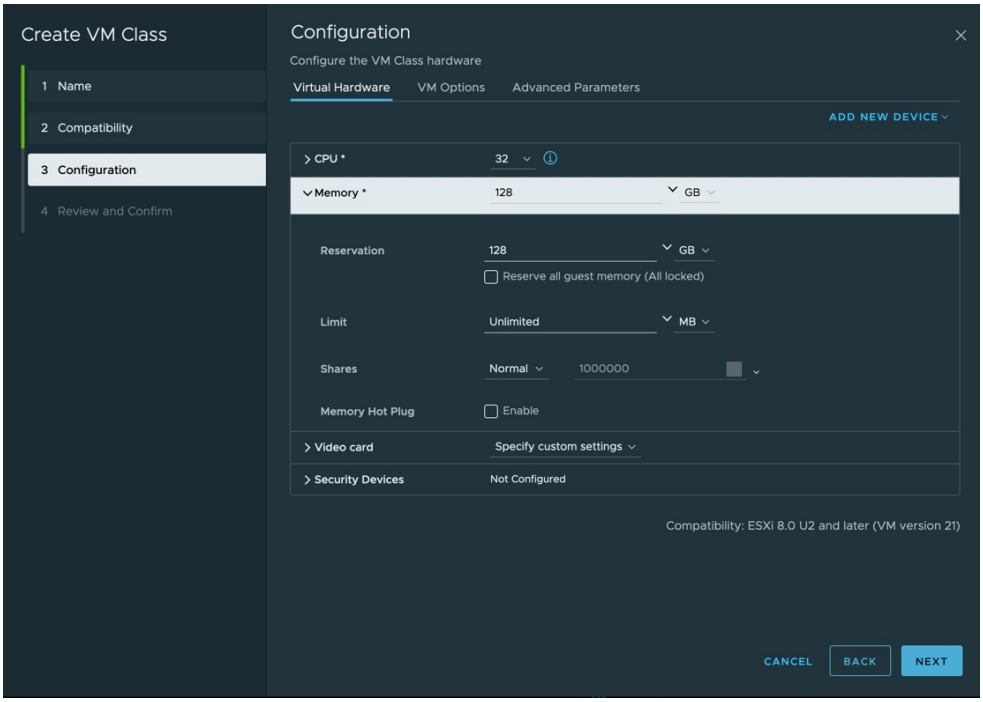
To create worker nodes with HW version 21 you'll need to create a custom VM Class: vSphere Client > Workload Management > [Your TKC] > VM Service Card > Manage VM Classes > Manage VM Classes (again) > Create VM Class.

Create the amx-guaranteed-large VM Class

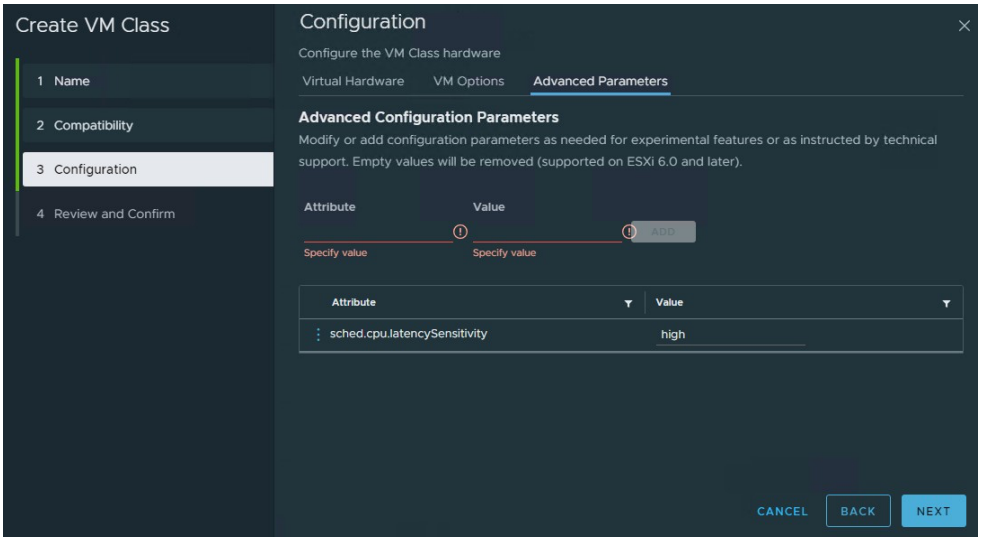


Select the HW Version / Compatibility Version

Set the number of CPUs and memory. Reserve all memory.

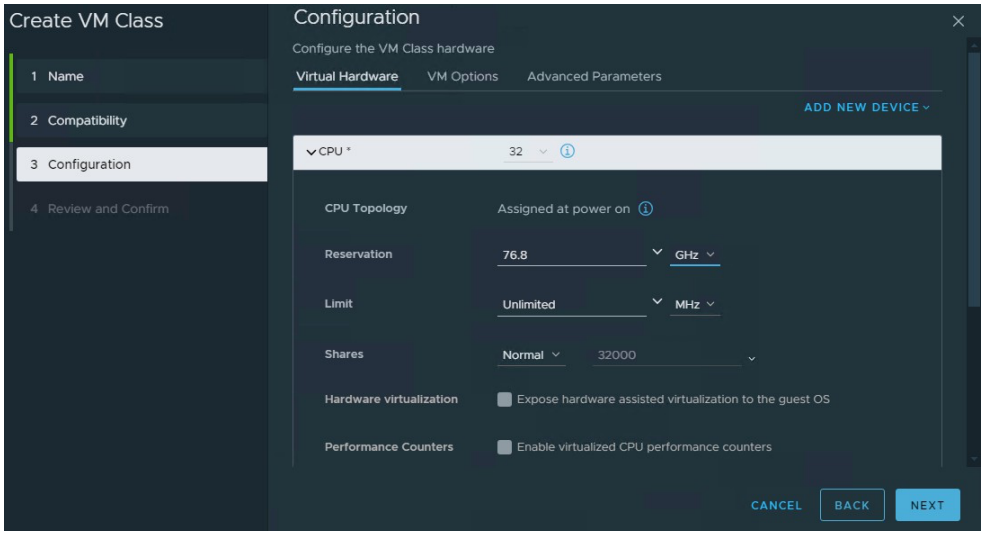


Optionally add an Advanced Parameter to set VM latency sensitivity to high.

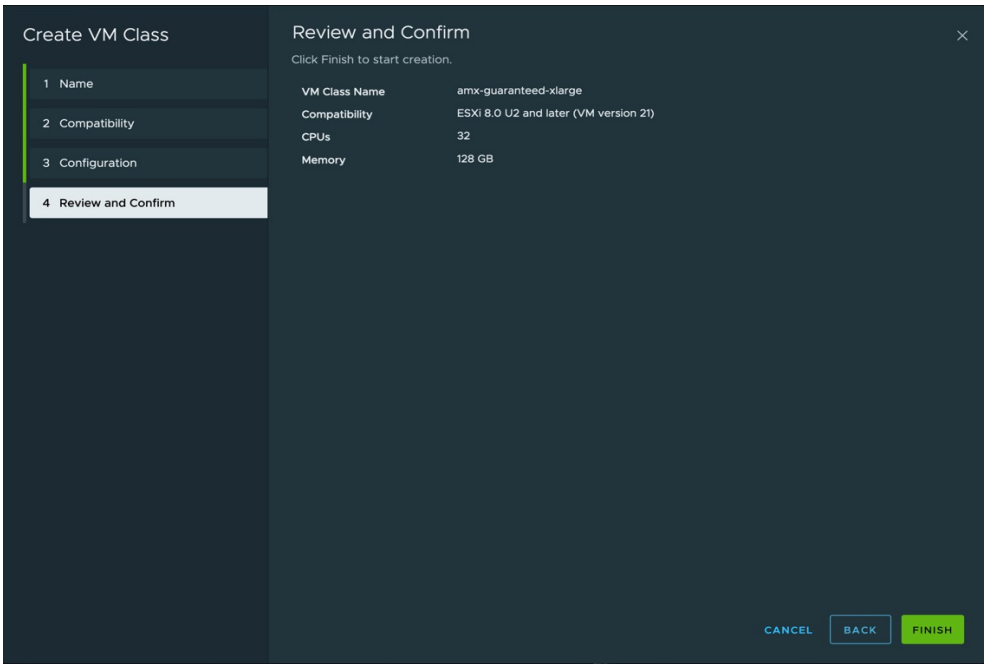


VMware latency sensitivity is a feature that optimizes the VM processor scheduling by granting vCPUs exclusive access to physical CPUs, thus making them inaccessible to ESXi and thereby lowering CPU ready time and alleviating interruptions from any vmkernel operations. You must add a new attribute of ***sched.cpu.latencySensitivity*** with a value of ***high*** to the Advanced Parameters tab to enable this setting.

When setting latency sensitivity to high, you must reserve 100% of both the RAM and vCPU allocation in MHz or GHz, and you must leave four (4) cores available for ESXi. In the instance of the 6448H and 8462Y+, they each have 32 cores, thus 64 physical cores in a dual socket configuration, so the maximum reservation per VM would be 60 cores but entered as GHz calculated from the CPU's base frequency. As an example, a VM with 60 vCPUs on the 6448H CPU would require a reservation of 144 GHz, or 168 GHz on the 8462Y+. You may also create VMs with fewer vCPU counts to allow other VMs to run on the same physical host.



Review and confirm, click Finish.



We're going to use two different VM classes, guaranteed-small (pre-defined) and amx-guaranteed-xlarge (the custom class we just created). The following tables provide information about these two VM classes that are used for this reference architecture as well as how Tanzu Kubernetes Grid nodes are configured.

Table 5: VM Class Configuration

VM Class Name	Type	CPU	RAM
guaranteed-small	Pre-defined	2	4GiB (100% Reservation)
amx-guaranteed-xlarge	Custom	32	128GiB (100% Reservation)

Table 6: Tanzu Kubernetes Grid Node Configuration

Role	Replicas	Storage Class	VM Class	Tanzu Kubernetes Release (TKR)
Control Plane	3	tanzu-storage-policy	guaranteed-small	v1.26.5---vmware.2-fips.1-hwekernel.1
Worker Nodes	4	tanzu-storage-policy	amx-guaranteed-xlarge	v1.26.5---vmware.2-fips.1-hwekernel.1

Add a new Content Library

To stand up a Tanzu Kubernetes Cluster (TKC) that supports AMX you need to make sure that you are using a Tanzu Kubernetes Release (TKR) image that contains kernel 5.16 or later.

The latest TKR that supports this kernel is in a separate content library from the mainstream Tanzu TKRs, so first you will need to add a new content library (vSphere Client > Content Libraries > Create) with the subscription URL: *https://wp-content.vmware.com/v2/labs/hwe/lib.json*. Once you have created this content library the v1.26.5---vmware.2-fips.1-hwekernel.1 TKR will be available for deploying new TKCs.

Update the Tanzu Kubernetes Grid and VM Services with the new Content Library

Add the new Content Library to the Tanzu Kubernetes Grid Service: vSphere Client > Workload Management > Namespaces > [your namespace] > Configure > Tanzu Kubernetes Grid Service EDIT > Content Library EDIT > Select the new library.

Add the new Content Library to the VM Service: Update the vSphere Client > Workload Management > Namespaces > [your namespace] > Summary > VM Service card > Manage Content Libraries > Check the new library > OK.

Deploy the new TKC

The following example is a YAML file named `developer-01.yaml` which deploys a TKC called `developer-01` in a namespace called `developers` for this reference architecture environment.

```
apiVersion: run.tanzu.vmware.com/v1alpha3
kind: TanzuKubernetesCluster
metadata:
  name: developer-01
  namespace: developers
  annotations:
    run.tanzu.vmware.com/resolve-os-image: os-name=ubuntu
spec:
  topology:
     controlPlane:
       replicas: 3
       vmClass: amx-guaranteed-xlarge
       storageClass: vsan-default-storage-policy
       tkr:
         reference:
           name: v1.26.5---vmware.2-fips.1-hwekernel.1
    nodePools:
    - name: worker
      replicas: 4
      vmClass: guaranteed-8xlarge
      storageClass: vsan-default-storage-policy
      volumes:
        - name: containerd
          mountPath: /var/lib/containerd
          capacity:
            storage: 160Gi
      tkr:
        reference:
          name:  v1.26.5---vmware.2-fips.1-hwekernel.1
```

To deploy the TKC:

```
$ kubectl config use-context developers
Switched to context "developers".
$ kubectl apply -f developer-01.yaml
tanzukubernetescluster.run.tanzu.vmware.com/developer-01 configured
```

Once your cluster is up and running you can deploy AI/ML workloads that use AMX to the cluster.

Test the connectivity to the workload cluster

To test the connectivity to the workload cluster, refer to the vSphere documentation *Workflow for Provisioning TKG Clusters on Supervisor Using Kubectl* to download and install the Kubernetes CLI tools for vSphere, then use the *kubectl* commands to monitor the cluster provisioning, and log into the workload cluster to check its resources.

More information on provisioning clusters

For more information see the product documentation *Provisioning TKG Clusters on Supervisor* and the VMware Private AI™ Foundation sample *YAML file* for provisioning a workload cluster.

Running LLM Tasks on vSphere IaaS Control Plane Kubernetes

Overview

After you have deployed and configured vSphere and Tanzu Kubernetes infrastructure, your environment is ready to run different types of tasks related to LLMs and GenAI. It should be noted that you can also run these workloads on traditional VMs as well while using the same solution and infrastructure. In the next section we provide two working examples: LLM fine-tuning and LLM inference. The intention of the working examples is to provide you with a procedure to verify the reference architecture deployment works appropriately; and provide examples that show how the progression works from fine-tuning a model to quantizing the model for performance to using the fine-tuned model weights in a chatbot. These are essential tasks in the LLM development lifecycle and will help you with your GenAI journey.

Task 1: LLM Fine-tuning

Introduction to Fine-tuning a Model

For many customers the thought of trying to create a model from scratch is overwhelming and may be too costly. An alternative to creating your own model is to fine-tune a commercially viable LLM that meets your desired outcomes. There are many models available today but individuals who are looking to leverage an open-source model should make sure they understand how the model was trained, what data was used, and whether they can use it free of charge. In the example below we selected a Llama-2-7b-chat model that is provided by Meta. This model provides the right balance of parameters, size, and accuracy for this fine-tuning exercise. The goal for fine-tuning the model is to provide context specific data, in this case finance data, to help improve the model's accuracy in a specific topic or area of focus.

There are two options for distributed fine-tuning: using a Kubernetes orchestration and using virtual machines. The virtual machine approach gives the maximum performance but requires additional setup considerations, whereas helm charts are supplied to orchestrate on a Kubernetes environment.

Performance considerations

If a Kubernetes orchestration such as Tanzu is used, the optimal approach is to use a 100% guaranteed VMclass with 1 thread per core. You should also account for overhead to ensure the hypervisor and vSAN have resources. We recommend leaving out at least 4 cores and 8GB of memory on each physical host.

If just virtual machines are being deployed, then the optimal approach is to use one monster VM per node with 1 thread per core pinning. We recommend leaving out at least 1 core per socket and 8GB of memory for ESXi's resource needs. Additional information on pinning BKMs is available at: *https://github.com/intel/vmware-platforms-scripts-and-tools/tree/main/pin-cpu*

Obtaining the model

You can download the model from Hugging Face and place it in an accessible location. We are using the chat model for the purposes of demonstrating a chatbot *https://huggingface.co/meta-llama/Llama-2-7b-chat-hf*

Distributed Llama2 fine-tuning using Kubernetes

If you are planning to run the fine-tuning recipe with VMs, please skip to the next

section. Cluster requirements:

- Kubeflow PyTorch Training operator deployed to the cluster
- Kubernetes storage class

Client requirements:

- kubectl command line
- tool Helm command line
  tool
- Access to the Llama 2 model in Hugging Face and a Hugging Face token

Building the container

The first step is to clone the repository.

git clone https://github.com/IntelAI/transfer-learning.git; cd transfer-learning/docker/hf_k8s

The next step is to build the docker container and make it accessible in the cluster. The below command builds the container. Push this container to an accessible location for when the helm chart gets deployed.

```
docker build \
  --build-arg IMAGE_NAME=intel/intel-optimized-pytorch \
  --build-arg IMAGE_TAG=2.1.0-pip-multinode \
  --build-arg PYTHON=python \
  --build-arg PYTHON_VER=3.9 \
  -t intel/ai-workflows:torch-2.1.0-huggingface-multinode-py3.9 .
```

Running the fine-tuning

In the directory templates, a values file is provided for finance fine-tuning. A guide to editing the values file can be found here for reference: *https://helm.sh/docs/chart_template_guide/values_files/*. A short summary of the required edits is provided below:

1. Select the correct tag of the image created as well as the location to fetch the image from (under the image section)

```
image:
  name: <NAME> # Specify the image name that was pushed to docker hub or copied to the nodes
  tag: <TAG> # Specify the image tag that was pushed to docker hub or copied to the nodes
  pullPolicy: IfNotPresent
```

2. Generate or retrieve a Hugging Face token with read access and use the terminal to get the base64 encoding. A token can be obtained here *https://huggingface.co/docs/hub/security-tokens*

echo <TOKEN> | base64

Edit the secret section with this token:

```
secret:
  name: hf-token-secret
  encodedToken: <ENCODED_TOKEN>
```

Some additional values that need to be edited are below:

- distributed.workers should be set to the number of workers that will be used for the job
- If you are using chart/values.yaml for your own workload, fill in either train.datasetName (the name of a Hugging Face dataset to use) or train.dataFile (the path to a data file to use). If a data file is being used, we will upload the file to the volume after the helm chart has been deployed to the cluster.
- resources.cpuRequest and resources.cpuLimit values should be updated based on the number of cpu cores available on your nodes in your cluster
- resources.memoryRequest and resources.memoryLimit values should be updated based on the amount of memory available on the nodes in your cluster
- resources.nodeSelectorLabel and resources.nodeSelectorValue specify a node label key/value to indicate which type of nodes can be used for the worker pods. kubectl get nodes and kubectl describe node <node name> can be used to get information about the nodes on your cluster.

storage.storageClassName should be set to your Kubernetes storage class name (use kubectl get storageclass to see a list of storage classes on your cluster)

Finally, deploy the helm chart.

cd docker/hf_k8s

helm install –namespace <namespace> -f chart/<chart_file.yaml> llama2-distributed ./chart

The logs will indicate the iteration and epoch the process is on. Once the fine-tuning is complete, the weights are present in the output directory inside the pvc. You can access it by using the kubectl cp command. The output location can be edited in the helm chart.

kubectl cp --namespace kubeflow <dataaccess pod name>:/tmp/pvc-mount/output/saved_model .

Distributed Llama2 fine-tuning using VMs

This section talks about fine-tuning using VMs. Refer to the previous section to fine-tune using

Kubernetes. Cluster requirements:

- VMs on each node with passwordless SSH enabled for communication.
- Conda on each VM

Setup the VMs (Do this on each VM)

Create a working directory. This will be where the scripts are stored. The first step is to clone the repository.

git clone https://github.com/IntelAI/transfer-learning.git; cd transfer-learning/docker/hf_k8s

Install the requirements by using the below commands:

conda config --add channels conda-forge

conda config --add channels intel

conda create -n llm -c intel -c conda-forge python=3.9 -y

conda activate llm

conda install gcc=12.3 gxx=12.3 cxx-compiler -c conda-forge -y

conda install mkl mkl-include -y

conda install gperftools jemalloc==5.2.1 -c conda-forge -y

pip install torch==2.1.0 --extra-index-url https://download.pytorch.org/whl/cpu

pip install datasets transformers==4.31.0 Jinja2 accelerate sentencepiece git+https://github.com/huggingface/peft.git evaluate nltk rouge_score protobuf==3.20.1 tokenizers einops

pip install intel_extension_for_pytorch==2.1.0 -f https://developer.intel.com/ipex-whl-stable-cpu

pip install oneccl_bind_pt==2.1.0 -f https://developer.intel.com/ipex-whl-stable-cpu

pip install neural_compressor

conda install impi_rt -y

Now go into the scripts directory.

cd scripts

Next, clone the model from *https://huggingface.co/meta-llama/Llama-2-7b-chat-hf*. Remember this path as <MODEL_PATH>

Install git-lfs from https://github.com/git-lfs/git-lfs/wiki/Installation

git lfs install

git clone https://huggingface.co/meta-llama/Llama-2-7b-chat-hf

Next, get the dataset. You can do this by running the download_financial_dataset.sh file.

export DATASET_DIR= $PWD

bash download_financial_dataset.sh

Create a hosts file. This will contain the list of VM IPs to use for fine-tuning. Additionally, at this stage, you would want to determine which VM to use as the main node to orchestrate the nodes. The hostfile lists each IP in a single line like below, with the main nodes IP being on the first line. (In this case, there are 4 nodes, skip this step if you're using a single VM for fine tuning).

cpu@cpu$ cat hostfile

10.x.x.x # IP of main VM

10.x.x.x # IP of VM 1

10.x.x.x # IP of VM 2

10.x.x.x # IP of VM 3

Inside the finetune.py script, add the following edits just after "import torch.distributed as dist" (skip this step if you're using a single VM for fine tuning)

import oneccl_bindings_for_pytorch

os.environ["MASTER_ADDR"] = "10.x.x.x" #IP of main node

os.environ["MASTER_PORT"] = "29501"

os.environ["RANK"] = str(os.environ.get("PMI_RANK", 0))

os.environ["WORLD_SIZE"] = str(os.environ.get("PMI_SIZE", 1))

dist.init_process_group(backend="ccl", init_method="env://")

Also, we need to set the default prompts for helping the model fine-tune with the finance dataset. These are the default prompts which were used to train the Llama-2 model but adjusted to include the context of being a finance assistant.

export PROMPT_WITH_INPUT=" You are a helpful, respectful, and honest finance assistant. Always answer as helpfully as possible, while being safe. Your answers should not include any harmful, unethical, racist, sexist, toxic, dangerous, or illegal content. Please ensure that your responses are socially unbiased and positive in nature. If a question does not make any sense, or is not factually coherent, explain why instead of answering something in correctly. If you don't know the answer to a question, please don't share false information."

export PROMPT_WITHOUT_INPUT=" You are a helpful, respectful, and honest finance assistant. Always answer as helpfully as possible, while being safe. Your answers should not include any harmful, unethical, racist, sexist, toxic, dangerous, or illegal content. Please ensure that your responses are socially unbiased and positive in nature. If a question does not make any sense, or is not factually coherent, explain why instead of answering something in correctly. If you don't know the answer to a question, please don't share false information."

Run the workload (do this on the main VM only)

Set the following environment settings:

export KMP_BLOCKTIME=1

export KMP_SETTINGS=1

export KMP_AFFINITY=granularity=fine,compact,1,0

export LD_PRELOAD=${CONDA_PREFIX}/lib/libstdc++.so.6

export LD_PRELOAD=${LD_PRELOAD}:${CONDA_PREFIX}/lib/libiomp5.so

export LD_PRELOAD=${LD_PRELOAD}:${CONDA_PREFIX}/lib/libtcmalloc.so

export NPROC_PER_NODE=<NPROC> # Number of processes per node (Number of sockets per node)

export NNODES=<NNODES> # Number of nodes (unnecessary if you're using a single VM for fine tuning)

export MODEL_PATH=<PATH_TO_LLAMA2_MODEL> # Path to Llama2 model

export TRAIN_DATASET=<PATH_TO_DATASET> # Dataset location (including the .json file name)

export OUTPUT_PATH=<PATH_TO_OUTPUT> # Location for output to be saved in

export ADDR=<> # Address of the main node, first line in the hostfile

The following step runs the fine-tuning command and will spin up the instances automatically in each node you specify in the hostfile.

ipexrun cpu --master_addr=$ADDR --hostfile ./hostfile --nprocs_per_node $NPROC_PER_NODE --nnodes $NNODES --memory-allocator auto finetune.py --model_name_or_path $MODEL_PATH --train_file $TRAIN_DATASET --dataset_concatenation --per_device_train_batch_size 8 --gradient_accumulation_steps 1 --do_train --learning_rate 2e-4 --num_train_epochs 3 --logging_steps 10 --save_total_limit 2 --overwrite_output_dir --log_level info --save_strategy steps --save_steps 5000 --output_dir $OUTPUT_PATH --no_cuda --use_ipex --use_cpu --ddp_backend ccl --use_fast_tokenizer False --bf16 True --prompt_with_input \"$PROMPT_WITH_INPUT\" --prompt_without_input \"$PROMPT_WITHOUT_INPUT\"

For single VM fine tuning, run the following command:

ipexrun cpu finetune.py --model_name_or_path $MODEL_PATH --train_file $TRAIN_DATASET --dataset_concatenation --per_device_train_batch_size 8 --gradient_accumulation_steps 1 --do_train --learning_rate 2e-4 --num_train_epochs 3 --logging_steps 10 --save_total_limit 2 --overwrite_output_dir --log_level info --save_strategy steps --save_steps 5000 --output_dir $OUTPUT_PATH --no_cuda --use_ipex --use_cpu --ddp_backend ccl --use_fast_tokenizer False --bf16 True --prompt_with_input \"$PROMPT_WITH_INPUT\" --prompt_without_input \"$PROMPT_WITHOUT_INPUT\"

The logs will indicate the iteration and epoch the process is on. Once the fine-tuning is complete, the weights are present in the output directory $OUTPUT_PATH.

Verifying model accuracy

If you want to verify the accuracy of the newly trained model, you must add the parameters "--do_eval --validation_split_percentage 0.2" when running the training command, as there is no way to verify the accuracy of the model after the fine tuning has been executed. Adding "--per_device_eval_batch_size 8" may also be used to speed up the evaluation process.

Below is a brief summary of the metrics measured in the accuracy evaluation:

- **Loss:** Cross entropy loss is measured in LLMs. The training script measures loss on the training dataset, which is an indicator of how well the training is progressing. Loss is directly co-related to how many mistakes the model makes when predicting the next token. Ideally, you want this to be as low and as close to 0 as possible.
- **Perplexity**: Perplexity is the exponent of loss. This is measured on the validation dataset and is done at the end of training. A perplexity of 1 is ideal. Perplexity can be thought of as how unsure the model is. The higher the perplexity score, the more unsure a model is. This is measured on the evaluation dataset (A split of the original dataset if not specified separately).

To include accuracy evaluation in the fine-tuning command across multiple nodes, run the following command:

ipexrun cpu --master_addr=$ADDR --hostfile ./hostfile --nprocs_per_node $NPROC_PER_NODE --nnodes $NNODES --memory-allocator auto finetune.py --model_name_or_path $MODEL_PATH --train_file $TRAIN_DATASET --dataset_concatenation --per_device_train_batch_size 8 --gradient_accumulation_steps 1 --do_train --learning_rate 2e-4 --num_train_epochs 3 --logging_steps 10 --save_total_limit 2 --overwrite_output_dir --log_level info --save_strategy steps --save_steps 5000 --output_dir $OUTPUT_PATH --no_cuda --use_ipex --use_cpu --ddp_backend ccl --use_fast_tokenizer False --bf16 True --prompt_with_input \"$PROMPT_WITH_INPUT\" --prompt_without_input \"$PROMPT_WITHOUT_INPUT\" --do_eval --validation_split_percentage 0.2 --per_device_eval_batch_size 8

To include accuracy evaluation in the fine-tuning command in a single VM, run the following command:

ipexrun cpu finetune.py --model_name_or_path $MODEL_PATH --train_file $TRAIN_DATASET --dataset_concatenation --per_device_train_batch_size 8 --gradient_accumulation_steps 1 --do_train --learning_rate 2e-4 --num_train_epochs 3 --logging_steps 10 --save_total_limit 2 --overwrite_output_dir --log_level info --save_strategy steps --save_steps 5000 --output_dir $OUTPUT_PATH --no_cuda --use_ipex --use_cpu --ddp_backend ccl --use_fast_tokenizer False --bf16 True --prompt_with_input \"$PROMPT_WITH_INPUT\" --prompt_without_input \"$PROMPT_WITHOUT_INPUT\" --do_eval --validation_split_percentage 0.2 --per_device_eval_batch_size 8

In terms of tips for fine-tuning on a custom dataset, here are a few things to keep in mind when working on it:

- Cleaning the dataset can eliminate some bad samples that can confuse the model.
- The system prompt should be edited to guide the model. This can help reduce the steps needed to understand the dataset & task.
- Adjust the number of epochs based on the dataset. More variety may need larger number of steps.
- Adjusting the Lora parameters can help with complexity in the dataset. Some of these will impact the training time. The various parameters are mentioned here: https://huggingface.co/docs/peft/main/en/conceptual_guides/lora

Now you can move on to the next task that outlines how to use this fine-tuned model and provides details on how to quantize the model to suite your performance needs as well as how to demo the model with a chatbot frontend.

Task 2: LLM Inference

Inference of a finance fine-tuned Llama2 LLM

Welcome to the world of fine-tuned model inferencing, where AI meets specialization to deliver tailored solutions for diverse tasks and domains. In today's rapidly evolving landscape of artificial intelligence, fine-tuned models have emerged as a powerful tool, enabling machines to understand and respond to human inputs with unprecedented accuracy and relevance. By training models on domain-specific data and fine-tuning their parameters, developers can optimize performance for a wide range of applications, like natural language understanding in chatbots. In the example below we will outline how to take the fine-tuned model and deploy it with a VM that will provide a user friendly chatbot experience so you can interact with the LLM to determine if it now has a better finance acumen.

Performance considerations

If a Kubernetes orchestration such as Tanzu is used, the optimal approach is to use a 100% guaranteed VM class with 1 thread per core. You should also account for overhead to ensure the hypervisor and vSAN have resources. We recommend leaving out at least 4 cores and 8GB of memory on each physical host.

If just virtual machines are to be deployed, then optimal approach is to pin the VM with 1 thread per core pinning. We recommend leaving out at least 1 core per socket and 8GB of memory for ESXi's resource needs. Additional information on pinning BKMs is available at: *https://github.com/intel/vmware-platforms-scripts-and-tools/tree/main/pin-cpu*

Note: **For quantization, at least 140GB of memory is recommended in this example. For inference, at least 50GB of memory is recommended.**

Setup for using Kubernetes

Refer to the section on fine-tuning using Kubernetes to build the container if not done already. Once the container is cloned, deploy a pod with the container and access to the Kubernetes storage class defined for fine-tuning. Make sure to expose port 65535 (or any other port that you want to access the inference from)

export PEFT_PATH =<PATH_TO_PVC>/output # Location for weights from finetuning

Clone the model from *https://huggingface.co/meta-llama/Llama-2-7b-chat-hf. Store this in your PVC.*

export MODEL_PATH=<PATH_TO_PVC>/<MODEL_DIR> # Path to Llama2 model

Setup using VMs

Refer to the section on setting up the VM for fine-tuning if not done already. You would need to activate the environment if needed.

conda activate llm

Setup the following variables:

export MODEL_PATH=<PATH_TO_LLAMA2_MODEL> # Path to Llama2 model

export PEFT_PATH =<PATH_TO_OUTPUT> # Location for weights from finetuning

Setting environment variables

Setup the following environment variables. Set $NUM_CORES according to the VM/pod specification

export KMP_BLOCKTIME=1

export KMP_SETTINGS=1

export KMP_AFFINITY=granularity=fine,compact,1,0

export LD_PRELOAD=${LD_PRELOAD}:${CONDA_PREFIX}/lib/libiomp5.so

export LD_PRELOAD=${LD_PRELOAD}:${CONDA_PREFIX}/lib/libtcmalloc.so

export OMP_NUM_THREADS=$NUM_CORES

Quantization of model

Quantization needs to be performed once to get the model weights to INT8 precision for better performance. Since INT8 is an 8-bit precision compared to BF16's 16 bit precision, a conversion and calibration is needed to ensure accuracy. If you want to use the BF16 model as is, skip to the next section.

First, get the scripts to quantize from *https://github.com/intel/intel-extension-for-pytorch/tree/v2.1.0%2Bcpu*

git clone https://github.com/intel/intel-extension-for-pytorch; cd intel-extension-for-pytorch; git checkout v2.1.0+cpu; cd examples/cpu/inference/python/llm/

Add the following lines to the quantize script inside single_instance/run_llama_quantization.py on line 119 (Just after the model load outside the if block):

import os

from peft import PeftModel

PEFT_PATH = os.environ.get('PEFT_PATH', './output')

user_model = PeftModel.from_pretrained(user_model, PEFT_PATH)

user_model = user_model.merge_and_unload()

Now run the quantization script. Set the output directory to where to save the weights.

Note: Do not pin this using numactl as it can cause the OS to kill the process.

export OUTPUT_DIR = <OUTPUT_LOC> # path to store weights after quantization

python single_instance/run_llama_quantization.py -m $MODEL_PATH --ipex-smooth-quant --dataset "NeelNanda/pile-10k" --output-dir $OUTPUT_DIR --int8-bf16-mixed

This will download the dataset and then run the quantization. This process can take some time and on completion, a best_model.pt will be saved inside the OUTPUT_DIR specified.

Note: You do not have to quantize on every VM/pod, you can just copy the weights after quantizing on one system over if you use the same dependency versions.

Running the inference

Clone the inference script from *https://github.com/intel/vmware-platforms-scripts-and-tools/tree/main/llm-demo*

git clone https://github.com/intel/vmware-platforms-scripts-and-tools; cd vmware-platforms-scripts-and-tools/llm-demo

Now setup the additional requirements for inference. This is going to install gradio and termcolor.

pip install -r requirements.txt

To run the inference server, run it as below. Numactl will perform pinning within the virtual environment and is recommended but can fail in docker environments. In this case, a simplified numactl by removing the "-m 0" can be tried or can be ignored completely by removing "numactl -m 0 -C 0-$NUM_CORES". **For quantized model**, use OUTPUT_DIR from previous step.

numactl -m 0 -C 0-$NUM_CORES python infer_server.py -n $MODEL_PATH -q $OUTPUT_DIR

For BF16 model use the command below. PEFT_PATH is the weights from finetuning:

numactl -m 0 -C 0-$NUM_CORES python infer_server.py -n $MODEL_PATH -p $PEFT_PATH

By default, the inference runs on port 65535. To run it on a different port, use the -port argument. To access this interface, go to
<IP_of_pod/vm>:<port> and the chatbot will be available to use. The first interaction can take a bit of time to warmup and fully load the model.

Note: To get smaller responses, the following instruction has been included in infer_server.py "Answer in 200 words or less". This can be removed to get larger responses but that takes more time to complete.

For additional tuning recommendations for 4th Gen Xeons for AI use cases please refer to this Intel site:

https://www.intel.com/content/www/us/en/developer/articles/technical/tuning-guide-for-ai-on-the-4th-generation.html

Extending the finetuning and inference

This paper details a mechanism to fine-tune the Llama2 model using a finance dataset. However, this approach can be extended for any dataset (and possibly even model) with appropriate modifications. A summary of the changes required at each step are detailed below:

Fine-tuning dataset

Any dataset in the alpaca format can be used to fine-tune the model. Make sure to do any appropriate pre-processing to clean the data for better results.

Fine-tuning script & process

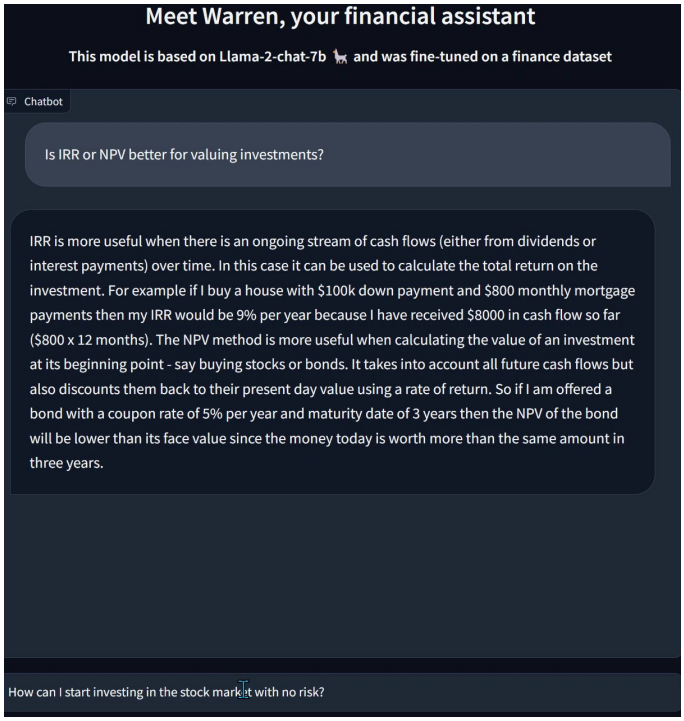
While the process to fine-tune is the same, the prompt template needs to be adjusted based on the model and dataset. These are the prompt_with_input and prompt_without_input options. These guide the model on what context to assume when trying to answer to the dataset.

Quantization

Use the new weights that were generated for quantization.

Inference

Now you can use the updated model weights. You can also edit the title on infer_server.py to show a more relevant message on the top of the chatbot frontend instead of "Meet Warren…". Below is a screenshot of what you should see once you navigate to the IP or the VM and the proper port (65535 by default Ex. http://YourIP:65535).


Meet Warren, your financial assistant
This model is based on Llama-2-chat-7b and was fine-tuned on a finance dataset
Chatbot
Is IRR or NPV better for valuing investments?
IRR is more useful when there is an ongoing stream of cash flows (either from dividends or interest payments) over time. In this case it can be used to calculate the total return on the investment. For example if I buy a house with $100k down payment and $800 monthly mortgage payments then my IRR would be 9% per year because I have received $8000 in cash flow so far ($800 x 12 months). The NPV method is more useful when calculating the value of an investment at its beginning point - say buying stocks or bonds. It takes into account all future cash flows but also discounts them back to their present day value using a rate of return. So if I am offered a bond with a coupon rate of 5% per year and maturity date of 3 years then the NPV of the bond will be lower than its face value since the money today is worth more than the same amount in three years.
How can I start investing in the stock market with no risk?

Conclusion

Properly designing a VMware virtual infrastructure for running LLM operations like fine-tuning and inference, is a critical consideration in achieving optimal performance and scalability. The demand for AI to help with support assistance, code generation and traditional Machine Learning use cases is driving a new wave of opportunity and VMware and Intel are committed to helping customers **unlock AI Everywhere**. Inference is a vital and practical application of LLMs, and appropriately sizing the infrastructure ensures that the computational resources align with the demands of the tasks. Under-provisioning can lead to sluggish performance and slow inference times, hampering user experience and efficiency. Conversely, an over-provisioned infrastructure can handle the required computations quickly, but it may lead to wasteful excess capacity, translating into higher costs without proportional benefits.

By properly designing the virtual infrastructure, organizations can strike a delicate balance, optimizing the utilization of resources without unnecessary waste. This ensures that the necessary computational power is available for demanding fine-tuning processes and real-time inference tasks while avoiding the costs associated with underutilized resources. This balance not only contributes to reducing operating expenses but also promotes a more responsive and agile environment, enabling organizations to adapt to changing needs and scale their operations seamlessly. With the powerful in-chip acceleration provided by 4th & 5th Gen Xeons organizations can also scale based on mainstream systems that are well known and don't require special cooling and power.

VMware Private AI™ with Intel allows enterprise customers to explore AI use cases with less operational overhead, faster implementation, increased developer agility, privacy and confidence to help drive real business outcomes that will easily flow into production and start to impact your customers and business.

We would like to be closer to the customers and partners interested in implementing this reference architecture for GenAI. We encourage you to visit the companion GitHub repositories for this document and provide your feedback. Our goal is to keep updating this document to make it useful for everyone interested in running their GenAI workloads on VMware Infrastructure.

References

- VMware Cloud Foundation AI/ML Solutions
- *VMware Cloud Foundation Documentation*
- *VMware vSphere Documentation*
- *vSphere with Tanzu Documentation*
- *VMware vSAN Design Guide*
- *VMware Validated Solutions*
- *vSphere Storage*
- *vSphere Network I/O control*
- *VMware Compatibility Guide*
- *VMware Aria Operations*
- *vSphere Lifecycle Manager*
- *Developer Ready Infrastructure for VMware Cloud Foundation*
- How to Build Trustworthy AI with Open Source
- *RoCE SR-IOV Setup and Performance Study on vSphere 7.x*
- *Enabling vSphere with Tanzu with VMware Cloud Foundation tuning guide for AI on 4th Gen Xeon*
- *https://github.com/intel/intel-extension-for-pytorch/tree/v2.1.0%2Bcpu/examples/cpu/inference/python/llm*
- *https://github.com/intel/vmware-platforms-scripts-and-tools*
- *How to Debug AI Model Performance on Intel® CPUs*

About the Authors

The following people wrote the original content of this solution reference architecture.

VMware by Broadcom

Chris Gully, Master Solutions Architect | VCF Partner Engineered

Solutions Earl C. Ruby III, R&D Engineer | VCF

Luke Huckaba, Master Solutions Architect | VCF Partner Engineered Solutions

Intel

Anitha Suresh, Director of Engineering

Anirudh Lakshmanan, Software Engineer

Patryk Wolsza, Cloud Systems Architect

Special Thanks

The authors of this paper would like to thank all the reviewers and additional contributors for their collaboration and assistance to help get this paper published.

Feedback

Your feedback is valuable.

To comment on this paper, contact Broadcom Product Marketing at: intel-pai-ra-feedback.pdl@broadcom.com