VULTR

Deploying an NVIDIA Dynamo-Based RAG Application with NetApp FlexCache and NVIDIA GPUs on Vultr Cloud

Table of Contents

Introduction 6

NVIDIA GPUs - B200 and GH Series 8

NVIDIA Nemotron 9

NVIDIA Dynamo 10

Hybrid Storage Architecture Using NetApp FlexCache 12

Architecture Overview 14

Hybrid Data Architecture 14

Compute and AI Services Architecture 15

NVIDIA Dynamo - Inference Orchestration Framework 16

Retrieval Augmented Generation (RAG) Pipeline 17

Deployment Overview 19

Deployment Workflow 19

Prerequisites 20

Deployment Scenario 21

Deployment Modes 23

Prepare FlexCache Volume access for Data Ingestion 25

Step 1 | Configure volume export/security settings (NFS/SMB) 25

Step 2 | Create Destination NetApp FlexCache Volume 27

Step 3 | Mount the NetApp FlexCache Volume with Junction Path 29

Step 4 | Modify the volume with export-policy 30

Step 5 | Mount on Vultr Compute 32

Create SVM, Volume and Mount point For Qdrant VectorDB 35

Step 1 | Create SVM on NetApp ONTAP for Qdrant 35

Step 2 | Create data LIF on NetApp ONTAP 36

Step 3 | Create a volume for Qdrant database 39

Step 4 | Set Junction Path for Qdrant Volume 40

Step 5 | Mount on Vultr Compute 45

Deploy Qdrant Vector DB on prod mount 46
Step 1 | Pre validate docker and qdrant volume mount 46
Step 2 | Verify storage Mount points 47
Step 3 | Create a storage directory 48
Step 4 | Deploy Qdrant Container 48
Deploy a DEV/TEST FlexClone volume for Qdrant database 49
Create a DEV FlexClone volume for Qdrant database 49
Create a TEST FlexClone volume for Qdrant database 54
Setting up NVIDIA Dynamo for serving LLMs 54
Inference Serving Modes 54
Deployment Overview 57
Model Repository Access and Hugging Face Authentication 62
Host Preparation 64
Install NVIDIA Container Toolkit 67
NVIDIA GPU Cloud (NGC) Authentication 71
Clone NVIDIA Dynamo repository 73
Pull vLLM Container Image (CUDA Version Match) 75
Configure Model Cache Directories 77
Deploy Dynamo Infrastructure Services 78
Deploy LLM with Aggregated Serving on NVIDIA Dynamo 82
Aggregated Serving Deployment Configurations 83
Configuration 1: Deploy Aggregated Serving with Single Worker (TP=4) 83
Configuration 2: Deploy Aggregated Serving with Dual Workers (TP=2) 94
Alternative Configuration: Distributed Multi-Container Deployment 109
Distributed Architecture Overview 110
Component Distribution 110
Communication Flow 111
Deployment Steps (Informational) 111
Operational Considerations 113
Deploy LLM with Disaggregated Serving on NVIDIA Dynamo 114

Model Selection for Disaggregated Mode 114
Memory Implications (Critical) 114
Architectural Characteristics 115
Stop Aggregated Worker (If Running) 115
Create Disaggregated Launch Script 117
Pre-download the model to avoid race conditions 119
Clean Up Previous Deployments 120
Deploy Disaggregated Workers as a Background Daemon 121
Monitor Disaggregated Workers Initialization 123
Service Endpoint Information 126
Verify Disaggregated Workers 127
Deploy Embedding Model using vLLM 133
Service Architecture (Embedding & Other Auxiliary Models) 133
Setup vLLM Cache Directory 134
Deploy Embedding Worker Container 135
Monitor Embedding Model Initialization 137
Service Endpoint Information 141
Verify Embedding Model Endpoint 143
Deploy Reranker Model using vLLM 144
Deploy Reranker Worker Container 145
Monitor Reranker Model Initialization 146
Service Endpoint Information 151
Verify Reranker Model Endpoint 153
Deploy Document Parser Model using vLLM 155
Deploy Document Parser Worker Container 156
Monitor Document Parser Model Initialization 158
Service Endpoint Information 162
Verify Document Parser Model Endpoint 164
Ingestion Pipeline for Qdrant 168
Ingestion Pipeline Overview 168

Document Synchronization Workflow 169
Vector Upsert with Metadata into Qdrant 171
Bulk Document Ingestion with indexer.py 173
RAG Pipeline API 174
Architecture Overview 174
Models Services and Endpoints 175
Install and Start RAG Services 175
Query Pipeline 180
Streamlit UI 184
Application UI 184
Conclusion 185

Introduction

Enterprises are increasingly adopting generative AI applications that interact with internal knowledge bases while maintaining strict governance over sensitive data. Many organizations store critical business documents in on-premises systems and cannot fully migrate these datasets to public cloud environments due to compliance, security, or operational constraints.

Vultr Cloud offers on-demand access to a variety of NVIDIA GPUs, enabling organizations to deploy high-performance AI and compute workloads on globally available infrastructure. Beyond GPU access, Vultr provides a comprehensive cloud platform with services such as bare metal, Kubernetes, block and object storage, file systems, snapshots, backups, VPC networking, load balancers, and security controls, helping customers build production-ready, scalable cloud environments around NVIDIA-powered acceleration.

This reference architecture demonstrates how to deploy a production-ready Retrieval-Augmented Generation (RAG) platform using NVIDIA GPUs, NVIDIA Dynamo, Qdrant Vector DB, and NetApp FlexCache hybrid storage on Vultr Cloud. The solution enables GPU-accelerated AI services to retrieve enterprise knowledge while maintaining control over data location and governance.

Key Advantages

This hybrid architecture delivers several advantages for enterprise AI deployments:

- **Data Sovereignty Preservation -** Enterprise datasets remain in on-premises storage while AI services access them through controlled caching mechanisms.
- **Low-Latency Enterprise Data Access** - NetApp FlexCache enables AI workloads to read documents directly from the extended storage namespace without replicating entire datasets to the cloud.
- **Efficient AI Inference Infrastructure –** NVIDIA GPUs combined with NVIDIA Dynamo provide optimized environments for large-scale model inference.
- **Independent Component Scaling** - Storage, vector indexing, inference services, and application layers can scale independently to support evolving workload demands.
- **Accelerated Enterprise AI Deployment -** Pre-validated architecture patterns and containerized model services simplify the deployment of production-grade generative AI applications.
- **Governed AI Workloads -** Existing enterprise security, compliance, and data governance policies remain intact while enabling AI systems to interact with internal knowledge sources.

Core Platform Components

The platform brings together GPU-accelerated inference, semantic retrieval, and hybrid data access into a single deployment model. AI services are deployed on NVIDIA GPUs using NVIDIA Dynamo, which orchestrates model execution and exposes APIs for application integration.

Document content and user queries are converted into vector embeddings and stored in Qdrant, enabling efficient semantic search and retrieval. During query processing, relevant context is retrieved from Qdrant and passed through a reranking stage before being sent to the language model for response generation.

Enterprise data remains on NetApp ONTAP, with FlexCache providing low-latency access to datasets from the cloud environment.

The architecture integrates the following components:

- **NVIDIA AI Inference Platform (Dynamo-based)** - Provides the orchestration layer used to deploy and manage GPU-accelerated AI services, including model inference and application workloads.
- **NetApp FlexCache** - Extends enterprise storage into the cloud while maintaining the authoritative data source on-premises.
- **NVIDIA GPUs (A100 / B200 / GH Series)** - Provide hardware acceleration for document parsing, embedding generation, reranking and large language model execution.
- **NVIDIA Dynamo** - Deploys and manages inference workloads across GPU resources with optimized scheduling and batching.
- **Document Parser (NVIDIA Nemotron-Parse)** - Extracts structured content from enterprise documents for downstream processing.
- **Embedding Service** - Converts document content and user queries into vector representations used for semantic search.
- **Qdrant Vector Database** - Performs semantic similarity search over document embeddings.
- **Reranker Model** - Improves retrieval quality by selecting the most relevant context
- **Large Language Model (NVIDIA Nemotron / LLM)** - Generates responses grounded in retrieved enterprise knowledge
- **Retrieval-Augmented Generation (RAG) Pipeline** - Retrieves relevant document context and generates responses grounded in enterprise knowledge.

By separating storage, vector indexing, AI inference services, and application layers, the architecture enables independent scaling of infrastructure while providing a flexible and scalable foundation for enterprise AI workloads.

NVIDIA GPUs - B200 and GH Series

NVIDIA data center GPUs provide the accelerated computing foundation required for modern AI platforms, supporting workloads such as model inference, vector embedding generation, document processing, and large language model execution. These GPUs are designed to deliver high computational throughput, large memory capacity, and efficient scaling across distributed environments.

The NVIDIA Blackwell B200 GPU and the NVIDIA Grace Hopper (GH) Superchip represent NVIDIA's latest generation of accelerated computing platforms. These architectures introduce significant advancement in compute performance, memory bandwidth, and system-level efficiency, enabling organizations to run increasingly complex generative AI workloads at scale.

NVIDIA B200 GPU

The NVIDIA B200 GPU is built on the Blackwell architecture and is designed to accelerate large-scale AI training and inference workloads. It introduces enhanced tensor performance, expanded memory capabilities, and improved multi-GPU scalability, making it well suited for running large generative AI models and high-throughput inference services in enterprise environments.

NVIDIA Grace Hopper (GH) Superchip

The NVIDIA Grace Hopper Superchip combines the Grace CPU with the Hopper GPU architecture using high-bandwidth NVLink-C2C interconnect technology. This tightly coupled design enables extremely fast communication between CPU and GPU memory, allowing large AI models and data-intensive workloads to operate more efficiently with reduced data movement overhead.

Key Capabilities

- High-performance tensor cores optimized for AI inference and training
- Large GPU memory capacity for executing large language models
- High-bandwidth NVLink interconnect for efficient multi-GPU scaling
- Architecture optimized for deep learning and generative AI workloads
- Improved system-level efficiency for data center AI infrastructure

Enterprise Benefits

These GPU platforms enable organizations to build and operate high-performance AI systems that can scale with growing model complexity and data demands while maintaining efficiency and flexibility.

- Accelerated AI workloads - Enables high-throughput inference and large model execution
- Scalable infrastructure - Supports multi-GPU deployments for distributed AI workloads
- Improved efficiency - Delivers better performance with optimized resource utilization

- Future-ready platform - Supports next-generation generative AI models
- Flexible deployment - Integrates with modern AI frameworks and enterprise environments

NVIDIA Nemotron

NVIDIA Nemotron is a family of large language models developed by NVIDIA for enterprise generative AI workloads. These models are designed to deliver strong language understanding, reasoning, and instruction-following capabilities while running efficiently on NVIDIA accelerated computing platforms.

Nemotron models are well suited for enterprise use cases such as document analysis, question answering, and conversational AI, where understanding domain-specific context is critical. In this architecture, the **Llama-3 Nemotron Super 49B** model is used as the primary language model to generate responses based on the context retrieved and refined during the workflow.

Key Features

- Advanced reasoning and instruction-following capabilities for enterprise use cases
- Context-aware response generation based on retrieved enterprise data
- Efficient execution on NVIDIA data center GPU infrastructure
- Compatibility with modern inference frameworks and orchestration platforms
- Scalable deployment across multi-GPU inference environments

Enterprise Benefits

These models enable organizations to build reliable and scalable AI systems that can generate meaningful responses grounded in enterprise data.

- **Knowledge-grounded responses** - Generates answers using retrieved enterprise information, improving reliability and contextual relevance.
- **Scalable AI deployment** - Supports high-throughput inference across GPU infrastructure for enterprise-scale workloads.
- **Improved productivity** - Enables intelligent assistants, document analysis systems, and knowledge discovery tools.
- **Flexible integration** - Can be incorporated into existing AI platforms, APIs, and enterprise applications.
- **Operational efficiency** - Accelerated inference reduces response latency while supporting demanding generative AI workloads.

NVIDIA Dynamo

NVIDIA Dynamo is an inference orchestration framework designed to manage large language model execution across GPU infrastructure. It provides a scalable control layer that coordinates request routing, model execution, and GPU resource utilization for high-performance AI inference workloads.

In this architecture, NVIDIA Dynamo functions as the inference control plane for AI services running on an **8× NVIDIA GPU node in Vultr Cloud,** handling requests from the Retrieval-Augmented Generation (RAG) application and efficiently distributing them across GPU workers.

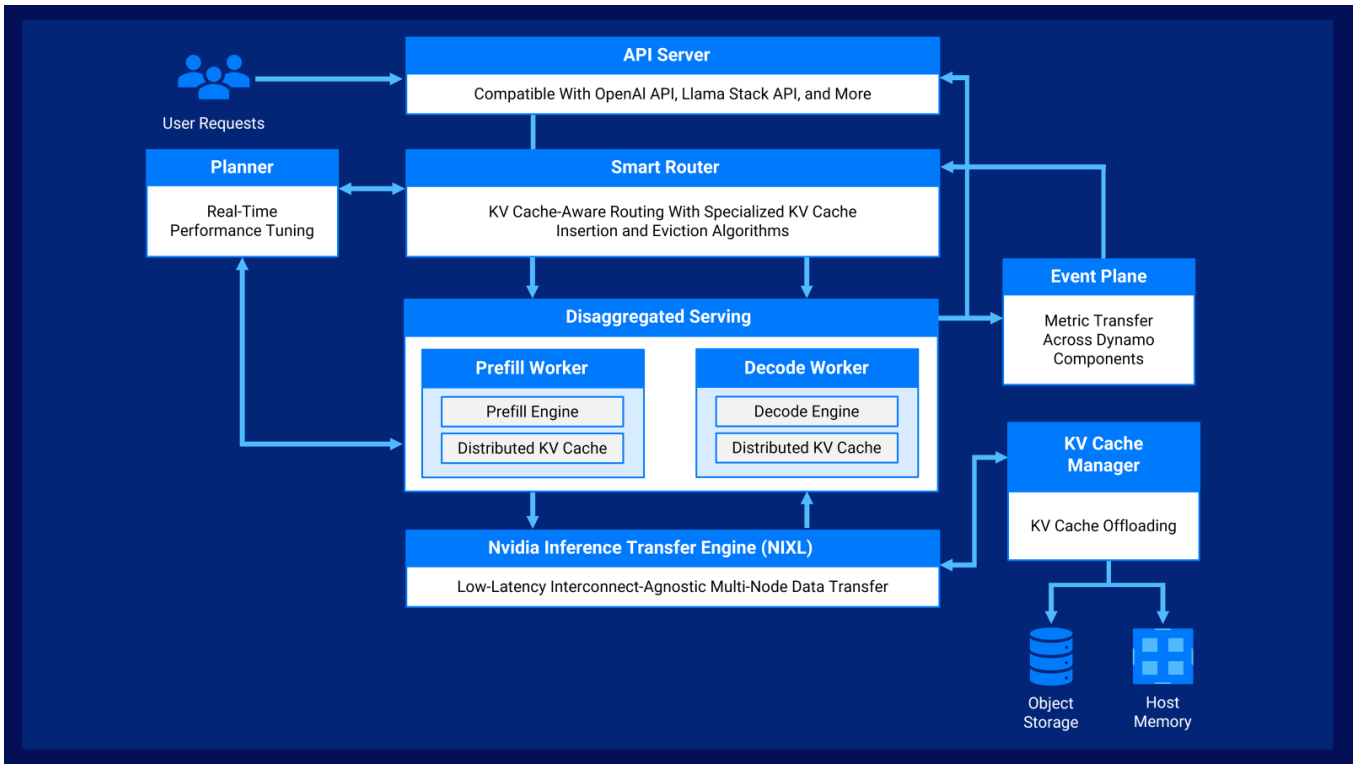


Figure 1 - NVIDIA Dynamo Disaggregated Inference Architecture

KV Cache Aware Router

The KV Cache Aware Router is a core component of the NVIDIA Dynamo architecture that improves inference efficiency by reusing previously computed attention states. During transformer-based inference, models generate key value (KV) attention pairs while processing prompts. These KV states are stored in GPU memory and reused during token generation, eliminating the need to recompute attention for previously processed tokens.

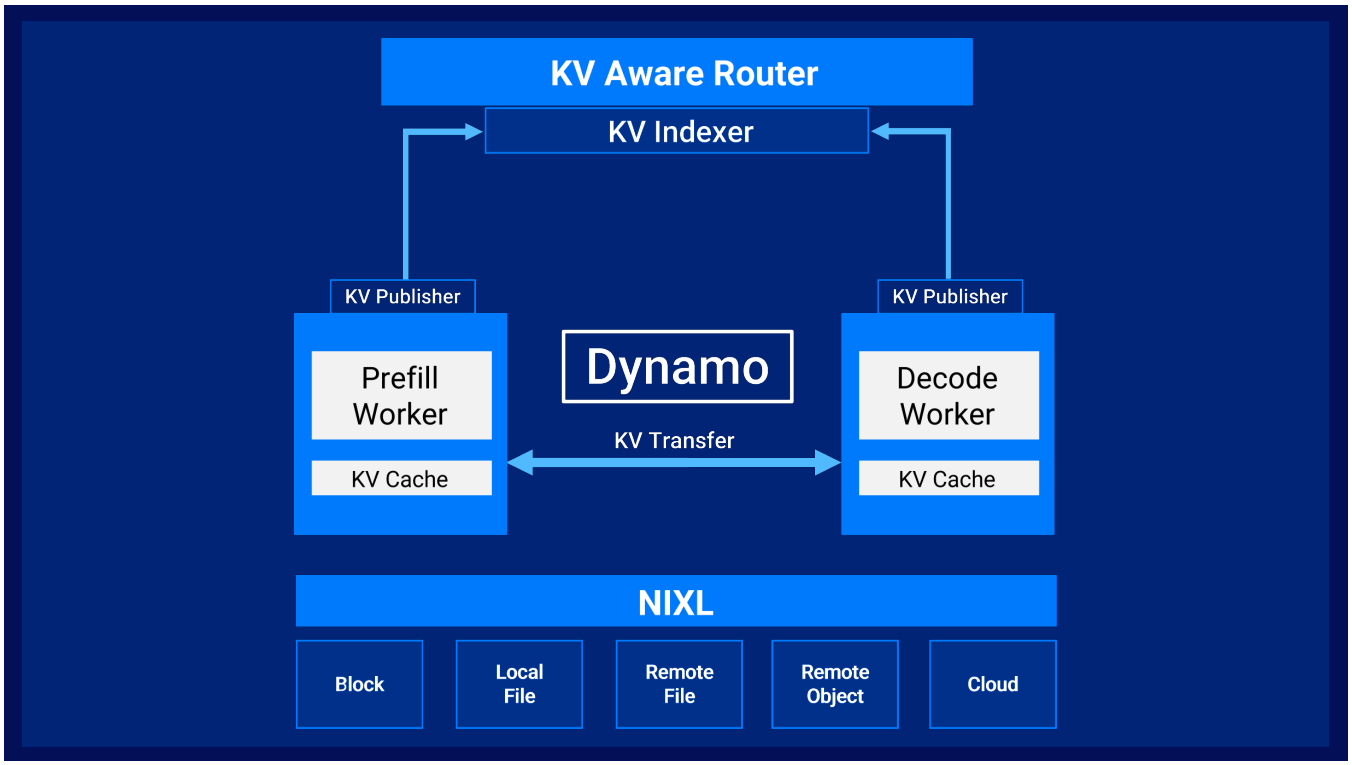


Figure 2 - KV Cache-Aware Routing and NIXL Data Flow

Within this architecture, **prefill workers** process the initial prompt and generate the KV cache during the first stage of inference. These cache entries are tracked by a **KV Indexer,** which maintains a mapping of KV states across GPU workers. When decode requests arrive, the router consults this index and directs the request to the worker that already holds the relevant KV cache, allowing token generation to continue without reprocessing the prompt context.

KV cache aware routing operates in both **aggregated** and **disaggregated** inference models. In aggregated deployments, where prefill and decode occur on the same worker, requests are consistently routed to the worker containing the cache. In disaggregated deployments, where these stages may run on different workers, Dynamo enables KV cache transfer using high-speed communication layers (such as NIXL), ensuring efficient cache reuse across distributed inference stages while maintaining low latency and high throughput.

Key Features

- OpenAI compatible API interface for seamless application integration
- Separation of API layer and model execution workers
- Support for high-performance inference backends such as vLLM and TensorRT-LLM
- GPU aware request routing and dynamic batching for efficient utilization
- Multi-model serving under a unified endpoint
- KV cache aware routing to reduce redundant computation and latency

- Support for both aggregated and disaggregated inference architectures
- Extensible framework for monitoring, high availability, and scale-out deployments

Enterprise Benefits

NVIDIA Dynamo enables organizations to operationalize AI inference at scale with improved efficiency, flexibility, and manageability.

- **Centralized inference control** - Manages multiple models and workloads through a unified platform
- **Optimized GPU utilization** - Reduces idle time and redundant computation across workloads
- **Scalable architecture** - Supports expansion across additional GPU nodes and clusters
- **Operational simplicity** - Eliminates the need for custom-built inference orchestration layers
- **Seamless integration** - Simplifies connectivity with enterprise AI applications and APIs
- **Future-ready deployment** - Supports evolving multi-node and distributed AI architectures

Hybrid Storage Architecture Using NetApp FlexCache

AI workloads require fast access to enterprise datasets, but migrating large volumes of data to the cloud is often impractical. NetApp FlexCache addresses this challenge by extending on-premises ONTAP volumes into cloud environments without requiring full dataset replication.

In this architecture:

- The on-premises NetApp ONTAP system remains the authoritative source of enterprise data.
- A FlexCache volume is deployed in Vultr Cloud and mounted by the AI application host through NFS.

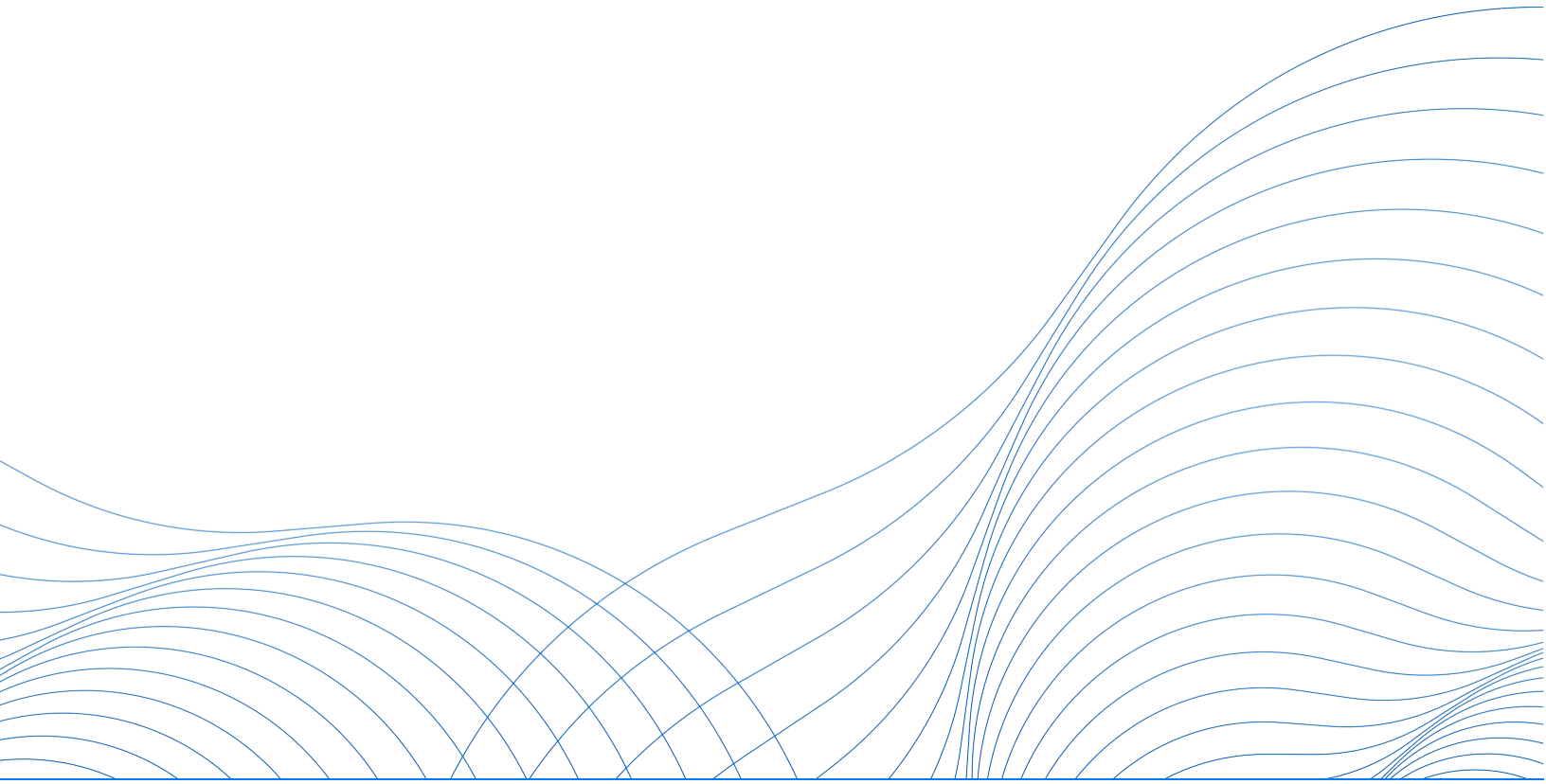
FlexCache dynamically retrieves frequently accessed files from the origin ONTAP system and caches them near the cloud compute infrastructure. This allows GPU-based AI workloads to access enterprise datasets with minimal latency while maintaining centralized storage governance.

Key Characteristics

- Enterprise data remains stored on-premises in NetApp ONTAP
- FlexCache extends the storage namespace into the cloud environment
- AI workloads access enterprise files through standard NFS mounts
- Frequently accessed files are cached near GPU/compute resources
- No bulk data migration or duplicate storage estates are required

Enterprise Benefits

- Low-latency access to enterprise datasets for AI workloads
- Preserves existing governance and compliance policies
- Reduces storage costs by avoiding full dataset replication
- Simplifies hybrid-cloud data access for AI pipelines
- Enables scalable multi-region AI deployments for enterprise AI platforms



Architecture Overview

The architecture combines hybrid storage, application services, vector search, and GPU-accelerated AI inference into a unified enterprise AI platform.

The system is composed of three primary layers:

- Hybrid Data Architecture
- Compute and AI Services Architecture
- Retrieval-Augmented Generation (RAG) Processing Pipeline

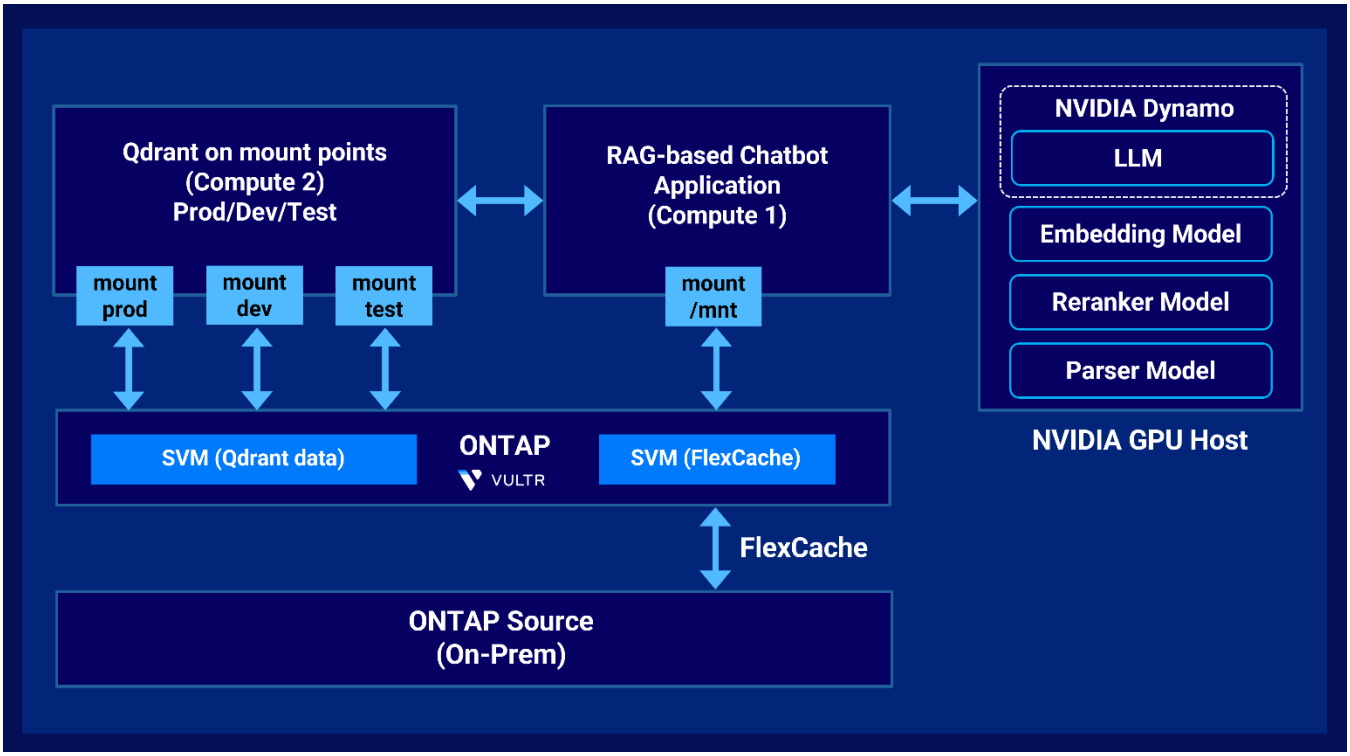


Figure 3 - High-Level RAG Architecture with NetApp Hybrid Storage and NVIDIA GPU

Hybrid Data Architecture

The hybrid data architecture enables AI services running in Vultr Cloud to access enterprise datasets stored in the on-premises NetApp ONTAP environment. A FlexCache volume deployed in Vultr Cloud provides a cached view of the ONTAP storage volumes and is mounted to the AI application host using NFS.

Through this mounted file system path, AI applications and GPU-accelerated services can access enterprise documents directly from the FlexCache volume, while the authoritative data remains stored on-premises.

This approach enables low-latency access to frequently used data without requiring separate data replication workflows, while preserving a single authoritative data source.

Compute and AI Services Architecture

This layer hosts the application, vector database, and GPU-based inference services required to process documents and serve user queries.

AI Application Host

The AI application host runs the RAG application responsible for orchestrating document ingestion and query processing.

Configuration

- 2 vCPUs
- 4 GB RAM

Key Functions

- Access enterprise files via the mounted FlexCache volume
- Process documents for ingestion
- Interact with embedding and reranking models
- Query the Qdrant vector database
- Construct prompts for LLM inference

Qdrant Vector Database

A dedicated compute instance hosts the Qdrant vector database used for semantic indexing and retrieval. Independent collections are maintained for Production, Development and Test environments to support controlled upgrades and safe experimentation.

Configuration

- 2 vCPUs
- 8 GB RAM

Key Functions

- Store vector embeddings generated from enterprise documents
- Perform similarity search operations
- Return the most relevant document chunks during query time

Separating the vector database from the application layer ensures scalability and predictable query performance.

AI Inference Layer

AI inference services run on a GPU server equipped with **8× NVIDIA GPUs** in Vultr Cloud. Model inference is orchestrated through **NVIDIA Dynamo**, which manages request routing, batching, and execution across GPU workers while supporting both **aggregated** and **disaggregated** inference architectures .

To ensure deterministic performance and eliminate resource contention, GPUs are allocated to individual models based on their functional roles within the RAG pipeline.

GPU allocation

- 1 GPU - Embedding model (BAAI/bge-m3)
- 1 GPU - Reranker model (BAAI/bge-reranker-v2-m3)
- 1 GPU - Document parser (nvidia/NVIDIA-Nemotron-Parse-v1.1)
- 4 GPUs - LLM served via NVIDIA Dynamo (nvidia/Llama-3_3-Nemotron-Super-49B-v1_5)

Inference Execution Modes

NVIDIA Dynamo supports two deployment models for large language model inference:

- **Aggregated Inference -** Prefill and decode stages are executed on the same GPU worker, simplifying deployment and reducing coordination overhead
- **Disaggregated Inference -** Prefill and decode stages are executed on separate GPU workers, enabling better resource utilization, higher throughput, and improved latency at scale

In this reference architecture, the LLM is deployed using a **disaggregated inference configuration,** where:

- Prefill Stage: 2 GPUs (TP=2)
- Decode Stage: 2 GPUs (TP=2)

This configuration enables efficient large-scale inference by separating prompt processing and token generation, improving throughput while maintaining low latency for RAG-based query workloads.

NVIDIA Dynamo - Inference Orchestration Framework

Within this architecture, NVIDIA Dynamo acts as the inference orchestration layer between the RAG application and the GPU-based model workers running on NVIDIA GPUs. All inference requests generated by the RAG pipeline are sent to the **Dynamo API endpoint**, which serves as the unified entry point for model execution. Dynamo routes each request to the appropriate model service and manages **request scheduling, batching, and GPU resource coordination** to ensure efficient and consistent inference.

By centralizing inference orchestration, Dynamo enables **multi-model serving,** supports scaling across GPU nodes, and provides a **single, consistent API interface** for integrating AI services into the application.

Retrieval Augmented Generation (RAG) Pipeline

The RAG pipeline generates responses grounded in enterprise documents by combining retrieval and generation.

The pipeline operates in two stages:

- **Offline ingestion stage -** builds a semantic knowledge index from enterprise documents
- **Online inference stage -** retrieves relevant context and generates responses in real time

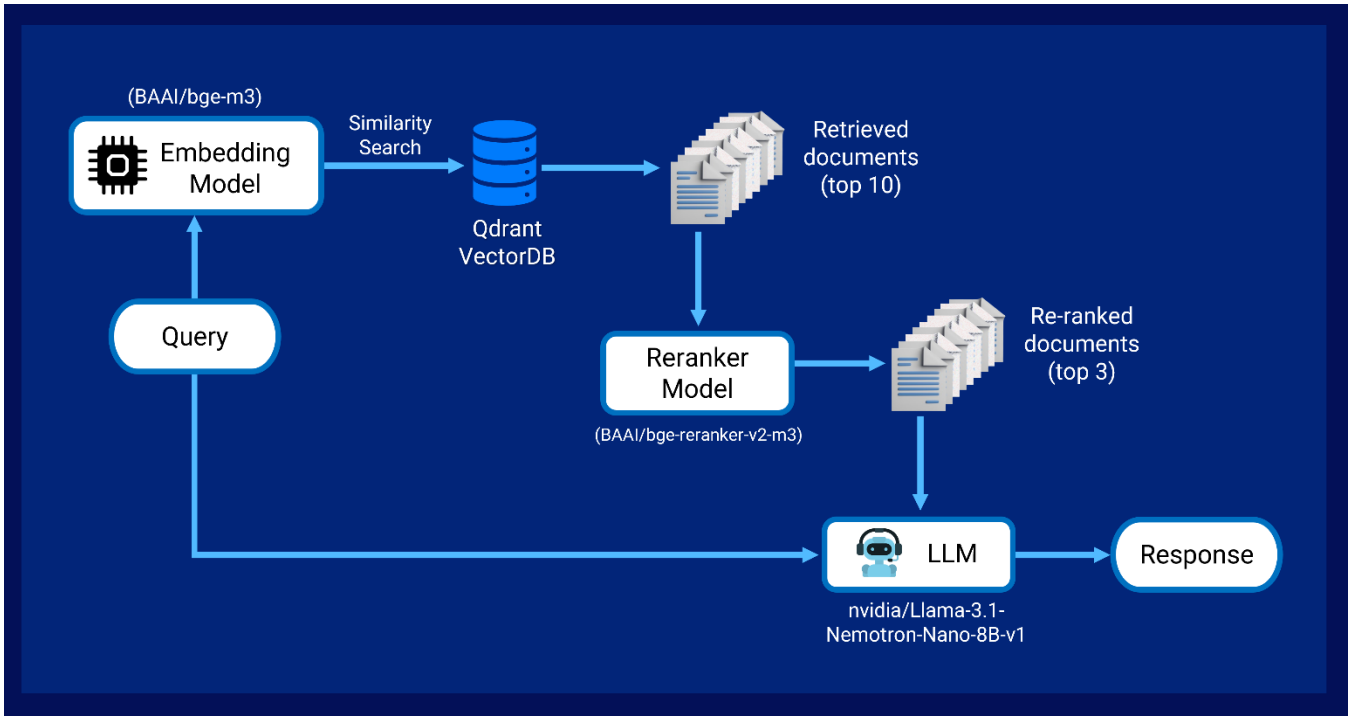


Figure 4 - Retrieval-Augmented Generation (RAG) Pipeline Workflow

Offline Ingestion - Building the Knowledge Index

Enterprise documents are processed and converted into a semantic representation stored in the Qdrant vector database.

Step 1 - Read

- Documents are accessed directly from the **NFS-mounted NetApp FlexCache volume,** without copying or staging data or secondary ingestion path. Documents are processed in place exactly as they exist on the authoritative NetApp ONTAP source.

Step 2 - Parse

- NVIDIA Nemotron-Parse (nvidia/NVIDIA-Nemotron-Parse-v1.1) extracts structured text from complex PDF layouts including tables, visually formatted PDF and multi-column content.

Step 3 - Chunk

- Documents are divided into overlapping text chunks to preserve context across sections. Overlapping ensures that meaning spanning paragraph or page boundaries is maintained, improving retrieval accuracy.

Step 4 - Embed

- Each chunk is processed by the BAAI/bge-m3 embedding model on a dedicated NVIDIA GPU to generate high-dimensional semantic vectors. Text with similar meaning produces vectors that are numerically close, enabling semantic retrieval beyond simple keyword matching.

Step 5 - Index in Qdrant

Generated embeddings are stored in **Qdrant,** deployed on a dedicated compute node within the Vultr environment. Qdrant indexes vectors for fast similarity search and maintains separate collections for Prod, Dev, and Test environments, enabling efficient retrieval during query-time inference.

Online Inference - Answering User Queries

During runtime, user queries are encoded, matched against the semantic index, filtered for precision, and answered using GPU-accelerated generation.

Step 1 - Encode Query

- The user query is embedded using the same embedding model **(BAAI/bge-m3 embedding model)** used during document ingestion. Using the same embedding space ensures that query vectors and document vectors remain aligned, enabling accurate semantic similarity matching.

Step 2 - Retrieve

- **Qdrant** performs a similarity search using the query embedding and returns the top candidate chunks with the closest semantic match. This step maximizes recall by identifying the most relevant document context for the query.

Step 3 – Re-Rank

- The retrieved candidates are passed to the **BAAI/bge-reranker-v2-m3 model** running on a dedicated NVIDIA GPU. The reranker evaluates relevance to the query and selects the most relevant chunks, ensuring only high-quality context is sent to the LLM.

Step 4 - Generate Response

- The top re-ranked chunks are assembled into a prompt and sent to the LLM **nvidia/Llama-3_3-Nemotron-Super-49B-v1_5,** served in a disaggregated architecture using 4 GPUs (Prefill: 2 GPUs, Decode: 2 GPUs). NVIDIA Dynamo orchestrates execution and routing, generating a response grounded in the retrieved enterprise document context.

Deployment Overview

This build guide describes how to deploy a production-ready Retrieval-Augmented Generation (RAG) application running on NVIDIA GPU infrastructure with NetApp ONTAP storage integration. The deployment provisions GPU-accelerated inference services using NVIDIA GPUs, integrates semantic retrieval through the Qdrant vector database, and connects enterprise documents stored in **NetApp ONTAP FlexCache.**

Application services are deployed using Docker containers on bare metal infrastructure**,** enabling organizations to operationalize a high-performance document intelligence solution on NVIDIA hardware with enterprise-grade storage.

The guide follows a sequential workflow, with each step building upon the previous configuration to ensure a reliable and production-ready RAG application deployment.

Deployment Workflow

The deployment process consists of the following stages:

- Configure networking and **NetApp FlexCache storage access** for enterprise documents.
- Install NVIDIA drivers, Docker, and container runtime with GPU support.
- Deploy NVIDIA Dynamo with vLLM backend for LLM inference (aggregated or disaggregated mode).
- Deploy embedding, reranker, and document parser models using standalone vLLM containers.
- Deploy the Qdrant vector database for storing and retrieving document embeddings.
- Validate deployment by confirming model endpoints, document ingestion, semantic retrieval, and response generation.

Prerequisites

Ensure the following infrastructure and tools are available.

Infrastructure Requirements

- A bare metal server with 8× NVIDIA GPUs (NVLink interconnect recommended)
- SSH access to the GPU server with root privileges
- Network connectivity to Hugging Face (model downloads) and NVIDIA NGC (container images)

Required Tools

The following tools must be installed on the **GPU server:**

- **Docker** – Container runtime for model deployment
- **NVIDIA Container Toolkit** – GPU access for Docker containers
- **nvidia-smi** – GPU monitoring and validation
- **curl** – API endpoint testing
- **git** – Cloning NVIDIA Dynamo repository
- **jq** – JSON parsing for API responses

Network and Security Configuration

- Allow SSH access (port 22) for remote administration
- Allow HTTPS egress for Hugging Face and NGC container registry access
- Ensure the following ports are accessible for application services:
    - Port 8000: LLM inference endpoint
    - Port 8001: Embedding model endpoint
    - Port 8002: Reranker model endpoint
    - Port 8003: Document parser endpoint
    - Port 6333: Qdrant vector database API
    - Port 6334: Qdrant gRPC interface

Baseline Environment Requirements

This guide assumes the following baseline environment:

- **ONTAP Storage:** Source SVMs and volumes available on the ONTAP cluster (Source & Destination)
- **FlexCache Configuration:** FlexCache storage configured in the Vultr Cloud environment
- **GPU Infrastructure:** A GPU server with 8× NVIDIA GPUs and compatible NVIDIA drivers installed

- **Compute Resources:** Sufficient CPU, memory, and disk space for running AI models and vector database
- **Network Connectivity:** A secure IPSec tunnel (or equivalent) established between the on-premises ONTAP environment and the Vultr cloud region
- **Administrative Access:** Root access to the GPU server and administrative access to ONTAP clusters and Vultr console or API

Preparing these baseline components ensures that the enterprise data storage, AI compute infrastructure, and networking connectivity are ready before deploying the RAG application stack.

> ⓘ Note on Example IP Addresses
>
> - This guide uses RFC 5737 documentation-reserved IP addresses (e.g., 192.0.2.XX) throughout all examples and command output. These are non-routable addresses designated by IANA specifically for use in documentation.
> - Replace them with your actual infrastructure IPs before deployment.

Deployment Scenario

For this deployment, the following components are used:

- **GPU Server:** Single bare metal server with 8× NVIDIA GPUs running Ubuntu 24.04
- **LLM Deployment:** NVIDIA Dynamo with vLLM backend, supporting three deployment modes:
  - **Traditional (Aggregated) serving with TP=4:** Simplest architecture, 4 GPUs with tensor parallelism
  - **Traditional (Aggregated) serving with TP=2:** Horizontal scaling with 2 workers, automatic load balancing
  - **Disaggregated:** Separate prefill/decode workers for latency-optimized RAG workloads
- **Standalone Models:** Embedding (GPU 4), Reranker (GPU 5), Document Parser (GPU 6) deployed as independent vLLM containers
- **Vector Database:** Qdrant deployed on a separate CPU-based compute node for semantic search and retrieval
- **FlexCache Integration:** Enterprise documents synchronized from NetApp FlexCache to local storage for ingestion
- **Container Orchestration:** Docker with NVIDIA Container Toolkit for GPU-accelerated inference

GPU Allocation Strategy

GPU ID	Service	Model	Purpose
0-3	LLM Worker	nvidia/Llama-3_3-Nemotron-Super-49B-v1_5	Chat completions, text generation
4	Embedding	BAAI/bge-m3	Document and query vectorization
5	Reranker	BAAI/bge-reranker-v2-m3	Relevance scoring for retrieved documents
6	Document Parser	nvidia/NVIDIA-Nemotron-Parse-v1.1	Document image to structured text conversion
7	Available	-	Reserved for Qdrant or additional services

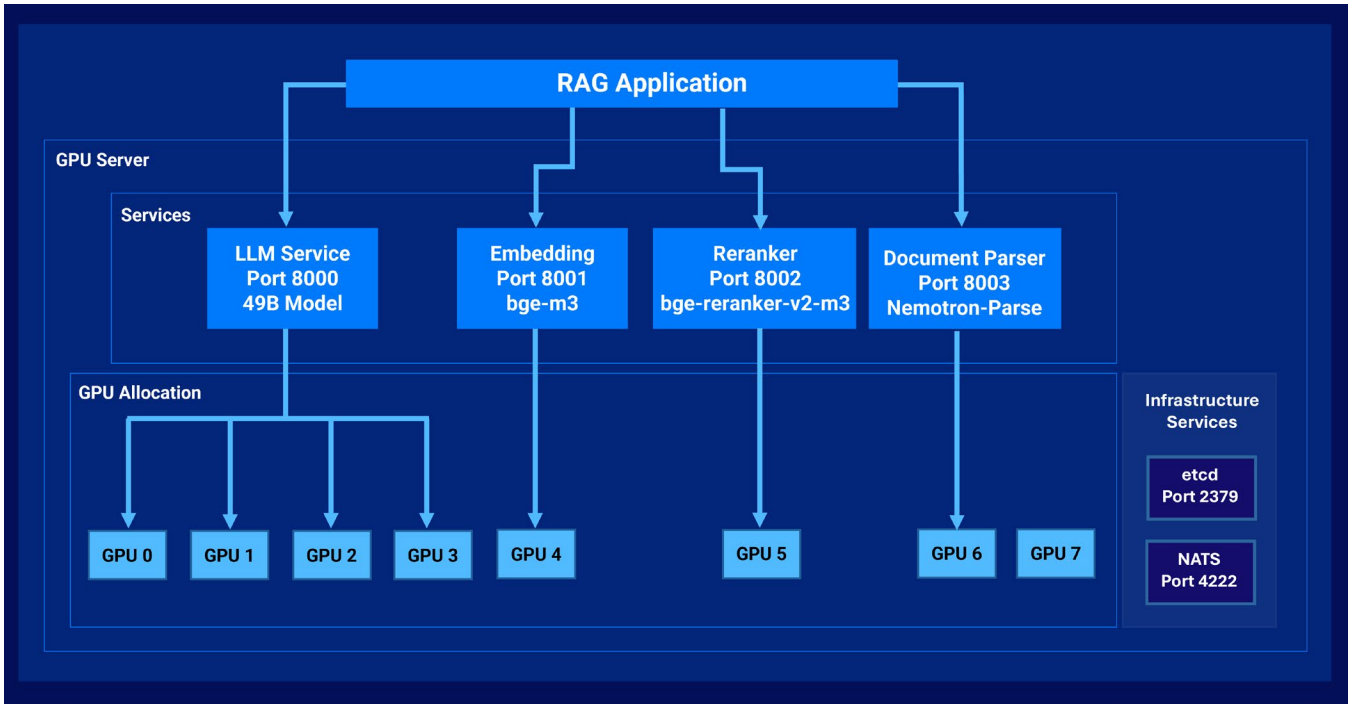


Figure 5 - GPU Allocation and AI Service Deployment Architecture

Deployment Modes

This guide outlines three LLM deployment modes to accommodate different performance and operational requirements. It provides detailed instructions for all three modes, allowing organizations to select the architecture that best matches their performance requirements and operational constraints.

Mode 1 – Traditional (Aggregated) serving with TP=4

- **Architecture:** Single worker process using 4 GPUs with tensor parallelism
- **Use Case:** Development, testing, moderate concurrency workloads
- **Advantages:** Simplest architecture, lowest single-request latency, easy debugging
- **GPU Allocation:** GPUs 0-3 (TP=4)

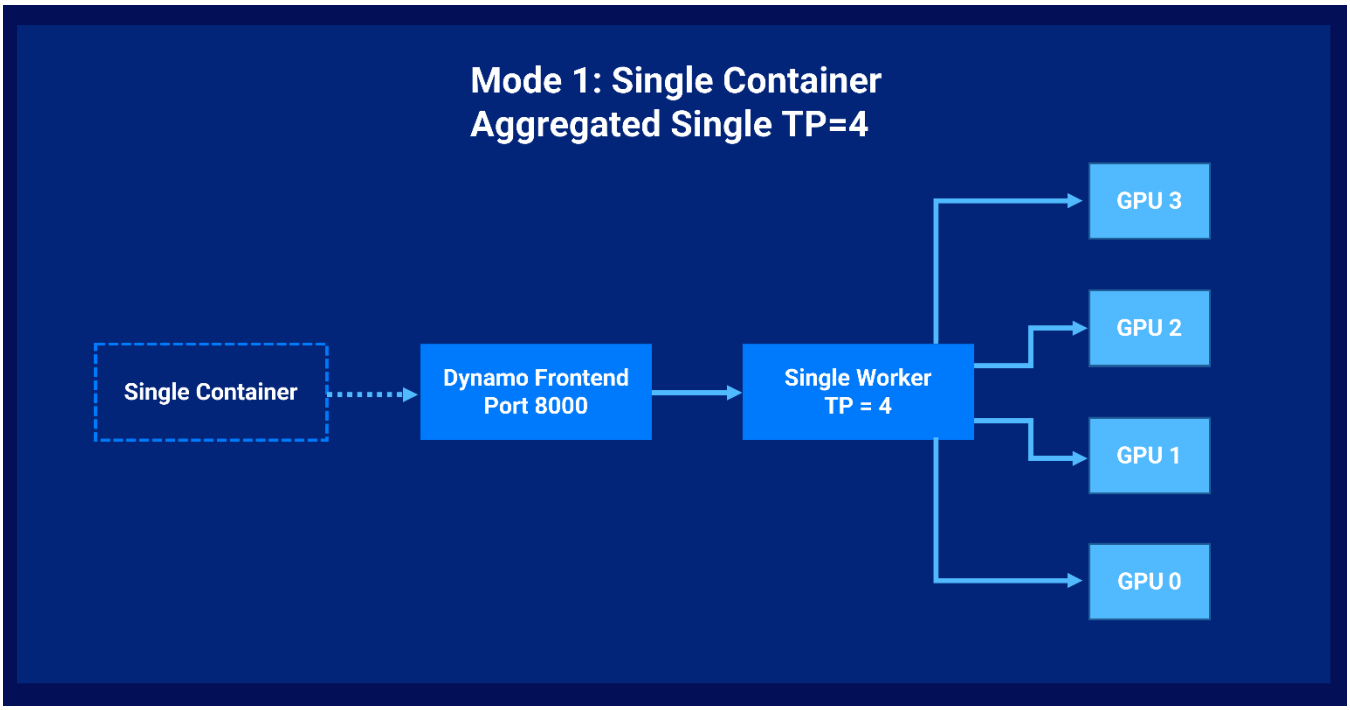


Figure 6 – Traditional (Aggregated) Serving (Single Worker, TP=4) on NVIDIA Dynamo

Mode 2 – Traditional (Aggregated) serving with TP=2

- **Architecture:** Two worker processes (TP=2 each) coordinated by single Dynamo frontend
- **Use Case:** Production workloads with high concurrent request volume
- **Advantages:** Higher throughput via parallelism, automatic load balancing, single endpoint
- **GPU Allocation:** Worker 1 (GPUs 0-1), Worker 2 (GPUs 2-3)

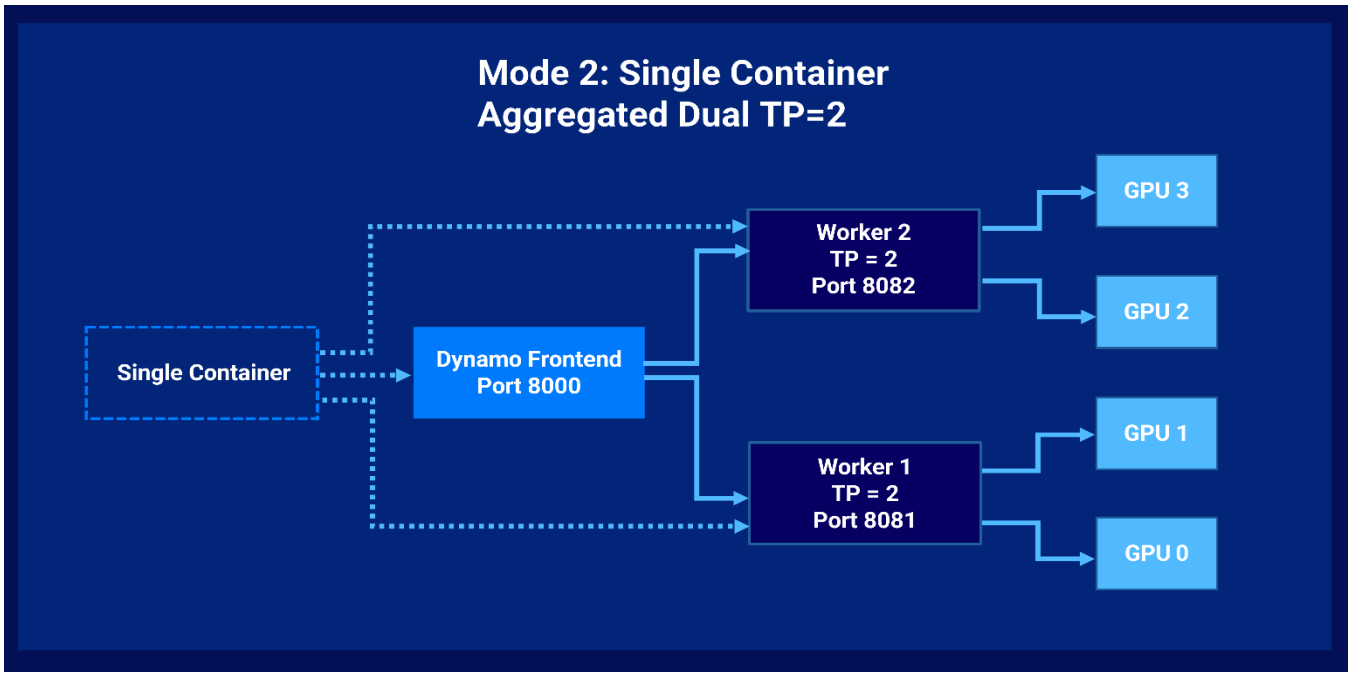


Figure 7 – Traditional (Aggregated) Serving (Dual Workers, TP=2) on NVIDIA Dynamo

Mode 3 – Disaggregated (Latency-Optimized)

- **Architecture:** Separate prefill and decode workers with NIXL connector for KV cache transfer
- **Use Case:** RAG workloads with strict latency SLAs, long prompts with short answers
- **Advantages:** Independent TTFT/ITL control eliminates prefill-decode interference
- **GPU Allocation:** Decode workers (GPUs 0-1), Prefill workers (GPUs 2-3)

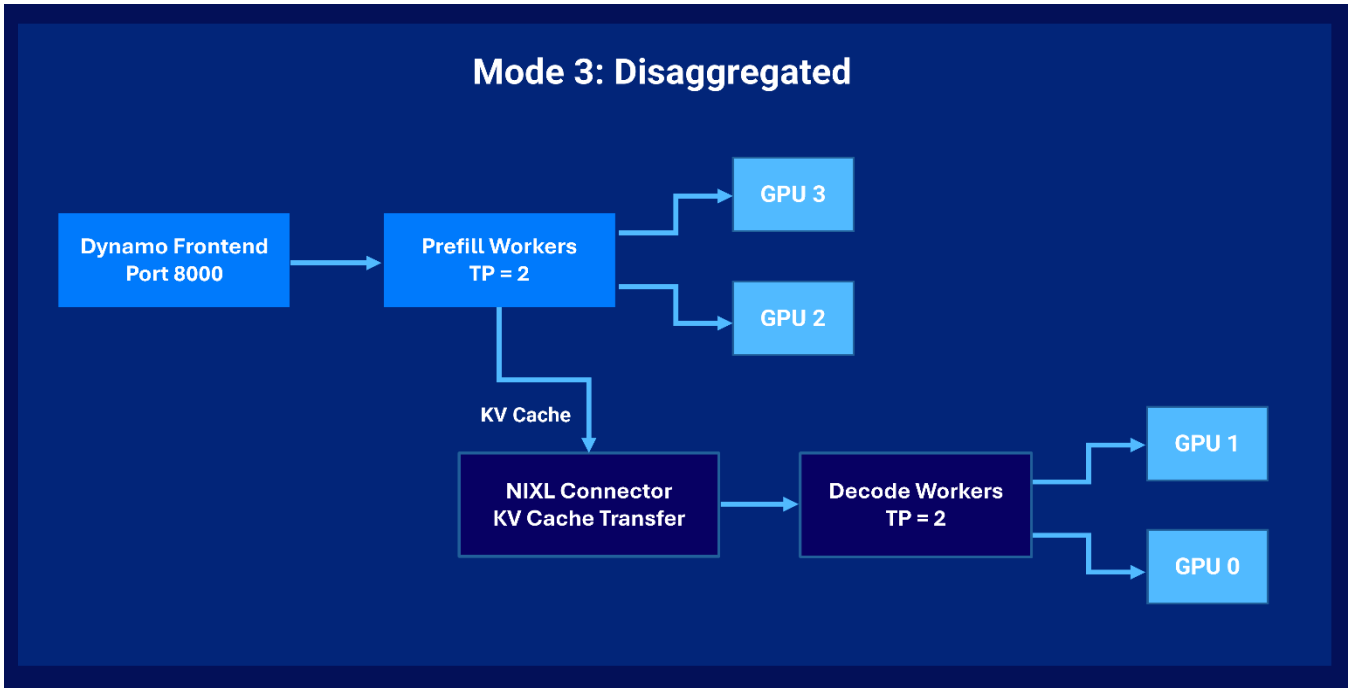


Figure 8 - Disaggregated Serving with Prefill and Decode Workers using NIXL

Prepare FlexCache Volume access for Data Ingestion

Mount the FlexCache export on the compute node before performing document ingestion. Use a stable absolute path and verify that the NFS client tools are installed. The mount must be accessible so that document synchronization can complete during deployment.

Step 1 | Configure volume export/security settings (NFS/SMB)

Export policies define which compute nodes can mount the FlexCache volume via NFS or SMB. Vultr compute instances must be explicitly granted both read-write access to the destination volumes. We will be configuring the following:

- **export-policy create** - Creates an export policy that defines the overall access framework for NFS clients on the SVM.
- **export-policy rule create** - Adds specific access rules (read-only/read-write, security type, client conditions) to the policy so ONTAP knows what permissions to enforce.
- **volume modify -policy <policy-name>** - Applies the export policy to a specific volume, enabling ONTAP to enforce those access rules when the volume is mounted.
- **volume mount -junction-path <path>** - Mounts the volume into the SVM's namespace, making it visible and accessible for NFS clients to mount.

```
vserver export-policy create -vserver dst_svm -policyname <policy-name>
```

```
ONTAPSelectCluster::*> export-policy create -policyname Test_dest_flexpolicy -vserver SVM_Dataflex
```

Validate:

```
export-policy show
```

```
ONTAPSelectCluster::*> export-policy show -policyname Test_dest_flexpolicy
Vserver        Policy Name
-------------- --------------------
SVM_Dataflex   Test_dest_flexpolicy
```

Expected Output:

- Export policy **Test_dest_flexpolicy** created successfully
- Associated with the **Vserver SVM_Dataflex**
- Status indicates the policy is **active** and ready to be configured with rules
- Confirms policy is ready to control NFS/CIFS access for volumes under the Vserver

Add rule:

- After creating the policy, associate the required hosts with the policy and assign the appropriate permission levels.

```
vserver export-policy rule create -vserver dst_svm -policyname <policy-name>-clientmatch
<nfs_node_on_vultr> -rorule any -rwrule any -protocol nfs3 –superuser any
```

```
ONTAPSelectCluster::*> vserver export-policy rule create -policyname Test_dest_flexpolicy -vserver
SVM_Dataflex -clientmatch 192.0.2.8 -rorule any -rwrule any -superuser any
```

Validate:

```
export-policy rule show -policyname Test_dest_flexpolicy -clientmatch 192.0.2.8
```

```
ONTAPSelectCluster::*> export-policy rule show -policyname Test_dest_flexpolicy -clientmatch
192.0.2.8

Vserver        Policy Name           Rule Index  Access Protocol  Client Match     RO Rule

-------------- -------------------- ---------- -------------- ---------------- ----------

SVM_Dataflex   Test_dest_flexpolicy      2          any           192.0.2.8        any
```

Expected Output:

- Rule added successfully to **export policy Test_dest_flexpolicy**
- Shows **client match:** 192.0.2.8
- Read-only (**rorule**) and read-write (**rwrule**) permissions set (**any**)
- Protocol applied: **NFSv3**
- Superuser access: **any**
- Confirms the rule is **active** and ready to allow NFS access for the client

Step 2 | Create Destination NetApp FlexCache Volume

A NetApp FlexCache volume stores hot (frequently accessed) data close to users or applications, while the origin volume remains the authoritative source. FlexCache automatically manages cache coherency, metadata synchronization, and write-forward behavior, ensuring consistent access across sites without administrative overhead.

FlexCache volumes do not need to match the size of the origin volume. Because they store only hot data, metadata, and recently accessed blocks, they can be provisioned significantly smaller - typically 5 - 20% of the origin volume's capacity - making them highly storage-efficient.

CLI

Create a NetApp FlexCache volume

The FlexCache volume is created on the destination cluster/SVM and linked to the source volume. Only metadata and frequently accessed data are cached, reducing latency and WAN traffic.

```
volume flexcache create  -vserver <cache_svm>  -volume <cache_volume_name>  -aggr-list <aggregate>
-size 1TB  -origin-volume <origin_vol>  -origin-vserver <origin_svm>
```

```
ONTAPSelectCluster::*> flexcache create -vserver SVM_Dataflex -volume Flex_vol_Test01 -size 7GB -
aggr-list aggr_data -origin-volume  Flex_Vol_S20 -origin-vserver SVM_Flexdata

(volume flexcache create)

[Job 128] Job succeeded: Successful.
```

- volume flexcache create

  Initiates the creation of a FlexCache volume on the destination (cache) cluster.
- -vserver <cache_svm>

  Specifies the Storage Virtual Machine (SVM) where the FlexCache volume will be created.
- -volume <cache_volume>

  Defines the name of the FlexCache volume on the cache SVM.
- -origin-vserver <source_svm>

  Identifies the source SVM that hosts the original (origin) volume.
- -origin-volume <source_volume>

  Specifies the source volume whose data will be cached in the FlexCache volume.
- -size <size>

  Sets the size of the FlexCache volume (used for metadata and cached data blocks).

Validate from Destination cluster

- Confirms the cache is correctly associated with the origin volume.

```
volume flexcache show
```

```
ONTAPSelectCluster::*> volume flexcache show
  (volume flexcache show)
Vserver Volume      Size       Origin-Vserver Origin-Volume Origin-Cluster
------- ----------- ---------- -------------- ------------- --------------
SVM_Dataflex Flex_vol_Test01 7GB SVM_Flexdata Flex_Vol_S20  sxId0edb1927eae795ba7
SVM_data Flex_Test_DES01 6GB   sx             Flex_Test_S10 sxId0edb1927eae795ba7
2 entries were displayed.
```

Validate from Source origin NetApp ONTAP cluster

- Validate origin and cache relationship health.

```
volume flexcache origin show
```

```
sxId0edb1927eae795ba7::*>  volume flexcache origin show
Origin-Vserver Origin-Volume  Cache-Vserver  Cache-Volume  Cache-Cluster
-------------- -------------- -------------- ------------- --------------
SVM_Flexdata   Flex_Vol_S20   SVM_Dataflex   Flex_vol_Test01 ONTAPSelectCluster
sx             Flex_Test_S10  SVM_data       Flex_Test_DES01 ONTAPSelectCluster
2 entries were displayed.
```

Expected Output:

- FlexCache volume **Flex_vol_Test01** created successfully
- Associated with **Vserver SVM_Dataflex**
- Linked to **aggregate aggr_data** and sized **7GB**
- Origin volume: **Flex_Vol_S20** on Vserver **SVM_Flexdata**
- Caching relationship with origin volume established and ready to serve cached data to clients

Step 3 | Mount the NetApp FlexCache Volume with Junction Path

The volume mount command attaches the volume to the SVM's namespace at a junction path, making it visible and mountable by NFS clients. Without a junction path, the volume exists internally but cannot be accessed over NFS.

```
volume mount -vserver <dest_svm>  -volume <cache_vol1>  -junction-path /Junction-path
```

```
ONTAPSelectCluster::*> vol mount -volume Flex_vol_Test01 -vserver SVM_Dataflex -junction-path
/Flex_vol_Test01
```

Validation

```
Volume show -volume <volumename> -fields junction-path
```

```
ONTAPSelectCluster::*> vol show -volume Flex_vol_Test01 -fields junction-path
vserver      volume          junction-path
------------ --------------- ----------------
SVM_Dataflex Flex_vol_Test01 /Flex_vol_Test01
```

Expected Output:

- Volume **Flex_vol_Test01** mounted successfully
- Mounted on **Vserver SVM_dataflex**
- Junction path set to **/Flex_vol_Test01**
- Volume status is **online** and accessible to clients
- Confirms the volume is ready for read/write operations

Step 4 | Modify the volume with export-policy

The **volume modify** command associates an export policy with the volume, allowing ONTAP to enforce the appropriate access rules when clients attempt to mount it. Without an export policy assigned, NFS access to the volume is not permitted.

Validate policy before modifying

Validate the existing export policy configuration to ensure the correct policy and rules are identified before making any modifications to the NetApp storage settings.

```
ONTAPSelectCluster::*> vol show -volume Flex_vol_Test01 -fields policy
vserver      volume          policy
------------ --------------- -------
SVM_Dataflex Flex_vol_Test01 default
```

Modify the policy

Modify the export policy to update access rules and permissions, ensuring the required clients are granted appropriate access to the storage resources.

```
volume modify -vserver dst_svm –volume <vol> -policy <policy-name>
```

```
ONTAPSelectCluster::*> vol modify -volume Flex_vol_Test01 -policy Test_dest_flexpolicy -vserver
SVM_Dataflex

[Job 133] Job succeeded: volume modify succeeded
```

Validate policy after modifying

Validate the export policy after modification to ensure the updated rules and permissions are correctly applied and functioning as intended.

```
volume show –volume <volume> -fields policy
```

```
ONTAPSelectCluster::*> vol show -volume Flex_vol_Test01 -vserver SVM_Dataflex -fields policy
vserver      volume          policy
------------ --------------- --------------------
SVM_Dataflex Flex_vol_Test01 Test_dest_flexpolicy
```

Expected Output:

- Volume **Flex_vol_Test01** updated successfully
- Associated with **Vserver SVM_dataflex**
- **Export policy** changed/applied to **Test_dest_flexpolicy**
- Confirms the volume is **online** and the new policy is active
- Volume is ready for client access under the updated export policy

Check Export Policy access

The **export-policy check-access** command in NetApp ONTAP is used to verify whether a specific client is allowed to access an NFS export and what level of access it will receive. It simulates an NFS access request and evaluates the export policy rules without mounting the volume.

The command checks the client IP or hostname against the export policy rules, determines whether access is permitted or denied, and reports the effective permissions such as read-only, read-write, superuser access, and protocol version. This is used for troubleshooting NFS permission issues, helping administrators quickly confirm if export policies are correctly configured for a given client.

```
export-policy check-access -vserver SVM_Dataflex -volume Flex_vol_Test01 -client-ip 192.0.2.8    -
authentication-method sys -protocol nfs3 -access-type read-write
```

If access is denied, add **clientmatch** into the root volume export policy as below:

```
ONTAPSelectCluster::*> export-policy rule create -policyname default -vserver SVM_Dataflex -
clientmatch 192.0.2.8 -rorule any -rwrule any -superuser any
```

Validate:

```
ONTAPSelectCluster::*> export-policy check-access -vserver SVM_Dataflex -volume Flex_vol_Test01 -
client-ip 192.0.2.8 -authentication-method sys -protocol nfs3 -access-type read-write
                                Policy    Policy     Rule             Security
Path                Policy      Owner     Owner Type Index Access     Style
------------------- ----------- --------- ---------- ---- ---------- ----------
/                   default     SVM_Dataflex_root volume 2 read      mixed
/Flex_vol_Test01    Test_dest_flexpolicy Flex_vol_Test01 volume 2 read-write mixed
2 entries were displayed.
```

Expected Output:

- Checks access for client IP 192.0.2.8 on volume **Flex_vol_Test01**
- Shows Vserver: **SVM_Dataflex**
- Authentication method: **sys**
- Protocol: **NFSv3**
- Access type tested: **read-write**
- Output indicates whether access is **allowed** or **denied**
- Confirms which **export policy rule** permits or blocks the access

Step 5 | Mount on Vultr Compute

Verification that Vultr compute nodes can access the replicated/cached dataset over NFS. Ensures the end-to-end path from **on-prem → NetApp FlexCache → NetApp ONTAP (Destination) → Vultr compute** is fully operational.

When we mount an NFS export from NetApp ONTAP(Destination) in Vultr:

- We are mounting the **active filesystem** of the destination NetApp FlexCache Volume
- NetApp FlexCache maintains a local cache of hot (frequently accessed) data from the source volume. Clients can perform read and write operations on the NetApp FlexCache volume - reads are served locally from the cache, while all writes are sent directly to the origin (source) volume, which remains the authoritative data source.

CLI (from compute instance on Vultr Cloud)

```
mount -t nfs <ontap_select_ip>:/<junction-path> <compute node mount point>
```

```
root@nvidia-gpu-compute:~# sudo mount -t nfs 192.0.2.53:/Flex_vol_Test01 /mnt/Flexcache_Site2
```

Testing Read Access

Once the FlexCache volume is mounted, we can verify that the end user is able to successfully access and read the cached data from the cache volume.

```
ls -l /mnt/ Flexcache_Site2/
```

```
root@nvidia-gpu-compute:~# cd /mnt/Flexcache_Site2
root@nvidia-gpu-compute:/mnt/Flexcache_Site2# ls -l
total 19560
-rw-rw-r-- 1 linuxuser linuxuser 4144691 Mar  2 07:18 ai-generated-images-with-stable-cascade-and-vultr-cloud-gpu.pdf
-rw-rw-r-- 1 linuxuser linuxuser 1602580 Mar  2 07:18 extract-tables-from-images-on-vultr-cloud-gpu.pdf
-rw-rw-r-- 1 linuxuser linuxuser  103115 Mar  2 07:18 how-to-establish-a-private-connection-between-vultr-and-gcp-using-direct-connect-via-console-connect.pdf
drwxr-xr-x 2 nobody    nogroup      4096 Mar  7 05:50 output
-rw-rw-r-- 1 linuxuser linuxuser  569706 Mar  2 07:18 platform-billing.pdf
-rw-rw-r-- 1 linuxuser linuxuser  274552 Mar  2 07:18 platform-security-best-practices.pdf
-rw-rw-r-- 1 linuxuser linuxuser 2053065 Mar  2 07:18 products-cloud-storage.pdf
-rw-rw-r-- 1 linuxuser linuxuser 3019509 Mar  2 07:18 products-compute.pdf
-rw-rw-r-- 1 linuxuser linuxuser  554014 Mar  2 07:18 products-kubernetes.pdf
-rw-rw-r-- 1 linuxuser linuxuser  474028 Mar  2 07:18 products-managed-database-kafka.pdf
-rw-rw-r-- 1 linuxuser linuxuser  424846 Mar  2 07:18 products-managed-database-mysql.pdf
-rw-rw-r-- 1 linuxuser linuxuser 1722812 Mar  2 07:18 products-managed-database.pdf
-rw-rw-r-- 1 linuxuser linuxuser  535387 Mar  2 07:18 products-managed-database-postgresql.pdf
-rw-rw-r-- 1 linuxuser linuxuser  357337 Mar  2 07:18 products-managed-database-valkey.pdf
-rw-rw-r-- 1 linuxuser linuxuser 1341174 Mar  2 07:18 products-network.pdf
-rw-rw-r-- 1 linuxuser linuxuser  427824 Mar  2 07:18 products-orchestration.pdf
-rw-rw-r-- 1 linuxuser linuxuser  388628 Mar  2 07:18 products-serverless-inference.pdf
```

```
-rw-rw-r-- 1 linuxuser linuxuser 1883586 Mar  2 07:18 solution-brief-qdrant.pdf
root@nvidia-gpu-compute:/mnt/Flexcache_Site2#
```

Testing Write Access (Destination NetApp ONTAP)

The following commands demonstrate the testing of FlexCache write functionality. A file named **output.txt** was successfully created on the cache volume, and it is now visible on the source NFS host, confirming that write operations are correctly redirected to the source volume.

```
root@nvidia-gpu-compute:~# cd /mnt/Flexcache_Site2
root@nvidia-gpu-compute:/mnt/Flexcache_Site2# ls -l
total 19560
-rw-rw-r-- 1 linuxuser linuxuser 4144691 Mar  2 07:18 ai-generated-images-with-stable-cascade-and-vultr-cloud-gpu.pdf
-rw-rw-r-- 1 linuxuser linuxuser 1602580 Mar  2 07:18 extract-tables-from-images-on-vultr-cloud-gpu.pdf
-rw-rw-r-- 1 linuxuser linuxuser  103115 Mar  2 07:18 how-to-establish-a-private-connection-between-vultr-and-gcp-using-direct-connect-via-console-connect.pdf
drwxr-xr-x 2 nobody    nogroup      4096 Mar  7 05:50 output
-rw-rw-r-- 1 linuxuser linuxuser  569706 Mar  2 07:18 platform-billing.pdf
-rw-rw-r-- 1 linuxuser linuxuser  274552 Mar  2 07:18 platform-security-best-practices.pdf
-rw-rw-r-- 1 linuxuser linuxuser 2053065 Mar  2 07:18 products-cloud-storage.pdf
-rw-rw-r-- 1 linuxuser linuxuser 3019509 Mar  2 07:18 products-compute.pdf
-rw-rw-r-- 1 linuxuser linuxuser  554014 Mar  2 07:18 products-kubernetes.pdf
-rw-rw-r-- 1 linuxuser linuxuser  474028 Mar  2 07:18 products-managed-database-kafka.pdf
-rw-rw-r-- 1 linuxuser linuxuser  424846 Mar  2 07:18 products-managed-database-mysql.pdf
-rw-rw-r-- 1 linuxuser linuxuser 1722812 Mar  2 07:18 products-managed-database.pdf
-rw-rw-r-- 1 linuxuser linuxuser  535387 Mar  2 07:18 products-managed-database-postgresql.pdf
-rw-rw-r-- 1 linuxuser linuxuser  357337 Mar  2 07:18 products-managed-database-valkey.pdf
-rw-rw-r-- 1 linuxuser linuxuser 1341174 Mar  2 07:18 products-network.pdf
-rw-rw-r-- 1 linuxuser linuxuser  427824 Mar  2 07:18 products-orchestration.pdf
-rw-rw-r-- 1 linuxuser linuxuser  388628 Mar  2 07:18 products-serverless-inference.pdf
-rw-rw-r-- 1 linuxuser linuxuser 1883586 Mar  2 07:18 solution-brief-qdrant.pdf
root@nvidia-gpu-compute:/mnt/Flexcache_Site2#
```

```
root@nvidia-gpu-compute:/mnt/Flexcache_Site2# cd output/
root@nvidia-gpu-compute:/mnt/Flexcache_Site2/output# date > output.txt
root@nvidia-gpu-compute:/mnt/Flexcache_Site2/output# cat output.txt
Sat Mar  7 06:39:50 AM UTC 2026
root@nvidia-gpu-compute:/mnt/Flexcache_Site2/output#
```

Testing Write Through to Origin (Source NetApp ONTAP)

ONTAP FlexCache uses **write-through semantics,** where all writes are forwarded directly to the origin volume, preserving a single authoritative source of truth. A write or update issued at the **FlexCache mount** is **forwarded to the origin volume.**

```
ubuntu@ip-192.0.2.99:/mnt/Flex_Vol_S20$ ls -l
total 19560
-rw-rw-r-- 1 ubuntu ubuntu 4144691 Mar  2 07:18 ai-generated-images-with-stable-cascade-and-vultr-
cloud-gpu.pdf
-rw-rw-r-- 1 ubuntu ubuntu 1602580 Mar  2 07:18 extract-tables-from-images-on-vultr-cloud-gpu.pdf
-rw-rw-r-- 1 ubuntu ubuntu  103115 Mar  2 07:18 how-to-establish-a-private-connection-between-
vultr-and-gcp-using-direct-connect-via-console-connect.pdf
drwxr-xr-x 2 root   root      4096 Mar  7 05:50 output
-rw-rw-r-- 1 ubuntu ubuntu  569706 Mar  2 07:18 platform-billing.pdf
-rw-rw-r-- 1 ubuntu ubuntu  274552 Mar  2 07:18 platform-security-best-practices.pdf
-rw-rw-r-- 1 ubuntu ubuntu 2053065 Mar  2 07:18 products-cloud-storage.pdf
-rw-rw-r-- 1 ubuntu ubuntu 3019509 Mar  2 07:18 products-compute.pdf
-rw-rw-r-- 1 ubuntu ubuntu  554014 Mar  2 07:18 products-kubernetes.pdf
-rw-rw-r-- 1 ubuntu ubuntu  474028 Mar  2 07:18 products-managed-database-kafka.pdf
-rw-rw-r-- 1 ubuntu ubuntu  424846 Mar  2 07:18 products-managed-database-mysql.pdf
-rw-rw-r-- 1 ubuntu ubuntu  535387 Mar  2 07:18 products-managed-database-postgresql.pdf
-rw-rw-r-- 1 ubuntu ubuntu  357337 Mar  2 07:18 products-managed-database-valkey.pdf
-rw-rw-r-- 1 ubuntu ubuntu 1722812 Mar  2 07:18 products-managed-database.pdf
-rw-rw-r-- 1 ubuntu ubuntu 1341174 Mar  2 07:18 products-network.pdf
-rw-rw-r-- 1 ubuntu ubuntu  427824 Mar  2 07:18 products-orchestration.pdf
-rw-rw-r-- 1 ubuntu ubuntu  388628 Mar  2 07:18 products-serverless-inference.pdf
-rw-rw-r-- 1 ubuntu ubuntu 1883586 Mar  2 07:18 solution-brief-qdrant.pdf
ubuntu@ip-192.0.2.99:/mnt/Flex_Vol_S20$ cd output/
ubuntu@ip-192.0.2.99:/mnt/Flex_Vol_S20/output$ ls -l
total 0
-rw-r--r-- 1 root root 32 Mar  7 06:11 output.txt
ubuntu@ip-192.0.2.99:/mnt/Flex_Vol_S20/output$ cat output.txt
Sat Mar  7 06:11:16 AM UTC 2026
ubuntu@ip-192.0.2.99:/mnt/Flex_Vol_S20/output$
```

Expected Output:

- Mount is successful
- Read and Write operations success

Create SVM, Volume and Mount point For Qdrant VectorDB

This section prepares persistent storage for the Qdrant vector database using NetApp ONTAP. A dedicated Storage Virtual Machine (SVM) is configured along with network access, an NFS-enabled volume, and appropriate export policies. The resulting storage path is then mounted on the Vultr compute instance where the Qdrant service runs, providing reliable and persistent storage for vector indexes and collections.

Step 1 | Create SVM on NetApp ONTAP for Qdrant

A Storage Virtual Machine (SVM) in ONTAP is a logical storage container that provides secure and isolated data access to clients using protocols such as NFS, SMB, and iSCSI. Each SVM independently manages its own volumes, network interfaces (LIFs), export policies, and security configurations, enabling multiple workloads to operate securely on the same storage cluster.

Create the SVM

```
vserver create -vserver vservername -subtype default -rootvolume rootvolumename -rootvolume-
security-style unix -language C.UTF-8 -snapshot-policy default -aggregate aggrname -data-services
data-iscsi,data-nfs,data-cifs,data-flexcache,data-nvme-tcp
```

```
ONTAPSelectCluster::> vserver create -vserver SVM_qdrant -subtype default -rootvolume
SVM_qdrant_root -rootvolume-security-style unix -language C.UTF-8 -snapshot-policy default -
aggregate aggr_data -data-services data-iscsi,data-nfs,data-cifs,data-flexcache,data-nvme-tcp

[Job 436] Job succeeded:

Vserver creation completed.
```

CLI Validation

```
vserver show -vserver *SVM_qdrant*
```

```
ONTAPSelectCluster::> vserver show -vserver *SVM_qdrant*
                                Admin      Operational Root
Vserver     Type    Subtype     State      State       Volume              Aggregate
----------- ------- ---------- ---------- ----------- ---------------     ----------
SVM_qdrant  data    default     running    running     SVM_ qdrant_root    aggr_data
```

Step 2 | Create data LIF on NetApp ONTAP

A Data LIF (Logical Interface) in NetApp ONTAP is a logical network interface configured with an IP address on the data network and associated with a specific node and physical port. It acts as the client-facing endpoint for storage access, enabling protocols such as NFS, SMB, and iSCSI to serve data from volumes within an SVM. Unlike cluster or inter-cluster LIFs, Data LIFs handle application and user traffic and must be explicitly created and assigned the appropriate data service policy. Without a properly configured Data LIF, client systems cannot mount or access storage volumes, even if basic network connectivity to the cluster is available. For availability and load distribution, it is recommended to configure at least one Data LIF per node involved in serving client data.

Check the available ports

Checking the available port before creating a LIF ensures that the selected network interface is active, properly configured, and free for assignment, enabling successful and reliable LIF deployment.

```
net port show
```

```
ONTAPSelectCluster::> net port show
  (network port show)

Node: ONTAPSelectCluster-01

Port    IPspace        Broadcast Domain  Link  MTU   Speed(Mbps)  Health
                                                     Admin/Oper   Status
------- ------------ ----------------- ----- ----- ------------ -------
e0a     Default        Default           up    1500  auto/auto    healthy
e0b     Default        Default           up    1500  auto/auto    healthy
e0c     Default        Default           up    1500  auto/auto    healthy
3 entries were displayed.

ONTAPSelectCluster::>
```

Use **e0c** which is the **default, stable, and recommended data-capable port** available on ONTAP for data traffic in most deployments.

```
ONTAPSelectCluster::> net int create -vserver SVM_qdrant -lif SVM_qdrant_nfs-lif -service-policy
default-data-files -address 192.0.2.54 -netmask 255.255.240.0 -home-node ONTAPSelectCluster-01 -
home-port e0c
(network interface create)
```

CLI Validation | Check LIF state:

Check the LIF state to verify that the logical interface is active and operational, ensuring proper network connectivity for storage access.

```
network interface show
```

```
ONTAPSelectCluster::> network interface show

Vserver           Logical           Status     Network            Current            Current Is
                  Interface         Admin/Oper Address/Mask       Node               Port    Home
---------------- ---------------- ---------- ------------------ ------------------ ------- ----
ONTAPSelectCluster
           ONTAPSelectCluster-01_mgmt1 up/up    192.0.2.61/20 ONTAPSelectCluster-01 e0a     true
           cluster_mgmt                up/up    192.0.2.60/20 ONTAPSelectCluster-01 e0a     true
           ic1                         up/up    192.0.2.51/20 ONTAPSelectCluster-01 e0a     true
SVM_Dataflex
           SVM_Dataflex_data_01        up/up    192.0.2.53/20 ONTAPSelectCluster-01 e0b     true
SVM_data
           SVM_data_cifs               up/up    192.0.2.52/20 ONTAPSelectCluster-01 e0c     true
SVM_qdrant
           SVM_qdrant_nfs-lif          up/up    192.0.2.54/20 ONTAPSelectCluster-01 e0c     true

6 entries were displayed.
```

```
net int show -role data -fields role,address,status-oper,status-admin,lif,home-port,home-node
```

```
ONTAPSelectCluster::> net int show -role data -fields role,address,status-oper,status-
admin,lif,home-port,home-node
  (network interface show)

vserver      lif                   role  address   home-node  home-port  status-oper status-admin
------------ --------------------  ----- --------  ---------  ---------- ----------  ------------
SVM_Dataflex SVM_Dataflex_data_01  data  192.0.2.53  ONTAPSelectCluster-01  e0b       up     up
SVM_data     SVM_data_cifs         data  192.0.2.52  ONTAPSelectCluster-01  e0c       up     up
SVM_qdrant   SVM_qdrant_nfs-lif    data  192.0.2.54  ONTAPSelectCluster-01  e0c       up     up

3 entries were displayed.
```

Expected Output:

- LIF: SVM_qdrant_nfs-lif
- Address: 192.0.2.54
- Home Node: ONTAPSelectCluster-01
- Current Node: ONTAPSelectCluster-01 (unless it has failed over)
- Home Port: e0c
- Operational Status: up
- Administrative Status: up

Enable NFS on ONTAP

Enable the NFS service so that external clients can access volumes hosted within the SVM.

```
vserver nfs create -vserver <vservername> -v3 enabled -v4.0 enabled
```

CLI

```
vserver nfs create -vserver SVM_qdrant -v3 enabled -v4.0 enabled
```

```
ONTAPSelectCluster::> vserver nfs create -vserver SVM_qdrant -v3 enabled -v4.0 enabled
ONTAPSelectCluster::> nfs show -vserver *SVM_qdrant*
Vserver: SVM_qdrant
        General Access:  true
                    v3:  enabled
                  v4.0:  enabled
                   4.1:  enabled
                   UDP:  enabled
                   TCP:  enabled
                  RDMA:  enabled
  Default Windows User:  -
 Default Windows Group:  -
```

Verify that the NFS server is configured and enabled on the SVM

```
nfs show
```

```
ONTAPSelectCluster::> nfs show -vserver *SVM_qdrant*

Vserver: SVM_qdrant

         General Access:  true
                     v3:  enabled
                   v4.0:  enabled
                    4.1:  enabled
                    UDP:  enabled
                    TCP:  enabled
                   RDMA:  enabled
   Default Windows User:  -
  Default Windows Group:  -
```

Step 3 | Create a volume for Qdrant database

A NetApp volume in ONTAP is a logical storage container created within a Storage Virtual Machine (SVM) that serves as the primary location where application and user data is stored. It is provisioned from an aggregate and presented to clients through supported protocols such as NFS, SMB, or iSCSI. The volume defines the namespace (junction path), capacity, security style, and export policy that control how data is accessed. Without creating and properly configuring a volume, no client system can store or retrieve data from the SVM, even if the network and Data LIF are correctly configured.

```
vol create -vserver <vservername> -volume <volumename> -state online -policy default -aggregate
<aggrname> -size <volumesize>
```

```
ONTAPSelectCluster::> vol create -vserver SVM_qdrant -volume qdrant_prod -state online -policy
default -aggregate aggr_data -size 100GB
[Job 437] Job succeeded: Successful
```

- volume create

  Initiates the creation of a Qdrant DB volume on the destination cluster.

- -vserver <SVM_Name>

  Specifies the Storage Virtual Machine (SVM) where the Qdrant DB volume will be created.

- -volume <volume_Name>

  Defines the name of the Qdrant DB volume on the SVM.

- -aggr <Aggr_Name>

  Defines the name of the aggr.

- -size <size>

  Sets the size of the Qdrant DB volume.

CLI Validation

```
Vol show -volume volumename -vserver vservername
```

```
ONTAPSelectCluster::> vol show -volume *qdrant_prod* -vserver SVM_qdrant
Vserver     Volume       Aggregate  State   Type  Size   Available  Used%
----------- ------------ ---------- ------- ----  ------ ---------- -----
SVM_qdrant  qdrant_prod  aggr_data  online  RW    100GB  95.00GB    0%
```

Expected Output:

- The volume **qdrant_prod** has been successfully created with a size of 100GB on the aggregate **aggr_data.**
- It is mounted on the Vserver **SVM_qdrant.**
- The volume status is **online** and accessible to clients.
- This confirms that the volume is ready for read/write operations.

Step 4 | Set Junction Path for Qdrant Volume

The volume mount command attaches the volume to the SVM's namespace at a junction path, making it visible and mountable by NFS clients. Without a junction path, the volume exists internally but cannot be accessed over NFS.

Check the Current Status

Check the current status to verify whether the volume or junction path is already mounted or in use before proceeding with the mount operation.

```
Vol show -vserver vserver name -volume volumename -fields junction-path
```

```
ONTAPSelectCluster::> vol show -vserver  SVM_qdrant -volume qdrant_prod -fields junction-path
vserver    volume      junction-path
---------- ----------- -------------
SVM_qdrant qdrant_prod -
```

Mount the volume

Mount the volume to make the storage accessible on the host system, allowing applications and users to read from and write data to the mounted directory.

```
volume mount -vserver <dest_svm>  -volume <cache_vol1>  -junction-path /Junction-path
```

```
ONTAPSelectCluster::> vol mount -volume qdrant_prod -junction-path /qdrant_prod -vserver
SVM_qdrant
```

Validation

```
Volume show -volume <volumename> -fields junction-path
```

```
ONTAPSelectCluster::> vol show -vserver  SVM_qdrant -volume qdrant_prod -fields junction-path
vserver    volume      junction-path
---------- ----------- -------------
SVM_qdrant qdrant_prod /qdrant_prod
```

Expected Output:

- Volume **qdrant_prod** mounted successfully
- Mounted on **Vserver SVM_qdrant**
- Junction path set to **/qdrant_prod**
- Volume status is **online** and accessible to clients
- Confirms the volume is ready for read/write operations

Configure volume export/security settings (NFS/SMB)

Export policies define which compute nodes can mount the FlexCache volume via NFS or SMB. Vultr compute instances must be explicitly granted both read-write access to the destination volumes. We will be configuring the following:

- **export-policy create -** Creates an export policy that defines the overall access framework for NFS clients on the SVM.
- **export-policy rule create -** Adds specific access rules (read-only/read-write, security type, client conditions) to the policy so ONTAP knows what permissions to enforce.
- **volume modify -policy <policy-name> -** Applies the export policy to a specific volume, enabling ONTAP to enforce those access rules when the volume is mounted.
- **volume mount -junction-path <path> -** Mounts the volume into the SVM's namespace, making it visible and accessible for NFS clients to mount.

```
vserver export-policy create -vserver dst_svm -policyname <policy-name>
```

```
ONTAPSelectCluster::> export-policy create -policyname Policy_qdrant -vserver SVM_qdrant
```

Validate

```
export-policy show -policyname Policy_qdrant
```

```
ONTAPSelectCluster::> export-policy show -policyname Policy_qdrant
Vserver          Policy Name
---------------  -------------------
SVM_qdrant       Policy_qdrant
```

Expected Output:

- Export policy **Policy_qdrant** created successfully
- Associated with the **Vserver SVM_qdrant**
- Status indicates the policy is **active** and ready to be configured with rules
- Confirms policy is ready to control NFS/CIFS access for volumes under the Vserver

Add rule

After creating the policy, associate the required hosts with the policy and assign the appropriate permission levels.

```
vserver export-policy rule create -vserver dst_svm -policyname <policy-name>-clientmatch
<nfs_node_on_vultr> -rorule any -rwrule any -protocol nfs3 –superuser any
```

```
ONTAPSelectCluster::> export-policy rule create -policyname Policy_qdrant -clientmatch 192.0.2.11
-rorule any -rwrule any -vserver SVM_qdrant -protocol nfs3 -superuser any
```

Validate

```
export-policy rule show -policyname Policy_qdrant -clientmatch 192.0.2.11
```

```
ONTAPSelectCluster::> export-policy rule show -policyname Policy_qdrant -clientmatch 192.0.2.11

Vserver     Policy Name    Rule Index  Access Protocol  Client Match  RO Rule
----------- -------------- ---------- --------------- ------------ -------
SVM_qdrant  Policy_qdrant 1           nfs3            192.0.2.11    any
```

```
export-policy rule show -policyname Policy_qdrant -fields rorule,rwrule
```

```
ONTAPSelectCluster::> export-policy rule show -policyname Policy_qdrant -fields rorule,rwrule

vserver     policyname    ruleindex  rorule  rwrule
----------- ------------ ---------- ------- -------
SVM_qdrant  Policy_qdrant 1          any     any
```

Expected Output:

- Rule added successfully to export policy **Policy_qdrant**
- Shows **client match:** 192.0.2.11
- Read-only (**rorule**) and read-write (**rwrule**) permissions set (**any)**
- Protocol applied: **NFSv3**
- Superuser access: **any**
- Confirms the rule is **active** and ready to allow NFS access for the client

Modify the volume with export-policy

The volume modify command assigns an export policy to the volume, enabling ONTAP to apply the correct access rules when clients attempt to mount it. Without an export policy bound to the volume, ONTAP will not allow any NFS access.

Validate policy before modifying

Validate the existing export policy configuration to ensure the correct policy and rules are identified before making any modifications to the NetApp storage settings.

```
ONTAPSelectCluster::> vol show qdrant_prod -fields policy
vserver    volume      policy
---------- ----------- -------
SVM_qdrant qdrant_prod default
```

Modify the policy

Modify the export policy to update access rules and permissions, ensuring the required clients are granted appropriate access to the storage resources.

```
volume modify -vserver dst_svm –volume <vol> -policy <policy-name>
```

```
ONTAPSelectCluster::> vol modify -volume qdrant_prod -vserver SVM_qdrant -policy Policy_qdrant
Volume modify successful on volume qdrant_prod of Vserver SVM_qdrant.
```

Validate policy after modifying

Validate the export policy after modification to ensure the updated rules and permissions are correctly applied and functioning as intended.

```
volume show –volume <volume> -fields policy
```

```
ONTAPSelectCluster::> vol show -volume qdrant_prod -fields policy
vserver    volume      policy
---------- ----------- -------------
SVM_qdrant qdrant_prod Policy_qdrant
```

Expected Output:

- Volume **qdrant_prod** updated successfully
- Associated with **Vserver SVM_qdrant**
- **Export policy** changed/applied to **Policy_qdrant**
- Confirms the volume is **online** and the new policy is active
- Volume is ready for client access under the updated export policy

Check Export Policy access

The **export-policy check-access** command in NetApp ONTAP is used to **verify whether a specific client is allowed to access an NFS export and what level of access it will receive**. It simulates an NFS access request and evaluates the export policy rules without mounting the volume. The command checks the client IP or hostname against the export policy rules, determines whether access is permitted or denied, and reports the effective permissions such as read-only, read-write, superuser access, and protocol version. This is mainly used for **troubleshooting NFS permission issues,** helping administrators quickly confirm if export policies are correctly configured for a given client.

```
export-policy check-access -vserver SVM_qdrant -volume qdrant_prod -client-ip 192.0.2.11 -
authentication-method sys -protocol nfs3 -access-type read-write
```

Validate

```
ONTAPSelectCluster::> export-policy check-access -vserver SVM_qdrant -volume qdrant_prod -client-
ip 192.0.2.11 -authentication-method sys -protocol nfs3 -access-type read-write

                                Policy    Policy     Rule            Security
Path                Policy      Owner     Owner Type Index Access    Style
------------------- ---------- --------- ---------- ---- ---------- ----------
/                   default    SVM_qdrant_root volume  0 denied     unix
```

If access is denied, add **clientmatch** into the root volume export policy as below:

```
ONTAPSelectCluster::> export-policy rule create -policyname default -vserver SVM_qdrant -
clientmatch 192.0.2.11 -rorule any -rwrule any -superuser any
```

Validate

```
ONTAPSelectCluster::> export-policy check-access -vserver SVM_qdrant -volume qdrant_prod -client-
ip 192.0.2.11 -authentication-method sys -protocol nfs3 -access-type read-write

                                Policy    Policy     Rule            Security
Path                Policy      Owner     Owner Type Index Access    Style
------------------- ---------- --------- ---------- ---- ---------- ----------
/                   default    SVM_qdrant_root volume  1 read       unix
/qdrant_prod        Policy_qdrant qdrant_prod volume   1 read-write mixed
2 entries were displayed.
```

Expected Output:

- Checks access for client IP 192.0.2.11 on volume **qdrant_prod**
- Shows **Vserver: SVM_qdrant**
- Authentication method: **sys**
- Protocol: **NFSv3**
- Access type tested: **read-write**
- Output indicates whether access is **allowed** or **denied**
- Confirms which **export policy rule** permits or blocks the access

Step 5 | Mount on Vultr Compute

Verify that Vultr compute nodes can access the provisioned NetApp storage FlexVolume using NFS ensures the end-to-end path - from **NetApp ONTAP(Vultr) → Vultr compute** - is fully operational.

When a volume is provisioned and mounted via FlexVolume:

- The volume is dynamically attached and mounted to the compute node using the NFS
- The volume is mounted over NFS from NetApp ONTAP, providing persistent storage that is directly accessible to applications running on the compute node
- Applications can perform read and write operations on the mounted volume, with data managed directly by the underlying ONTAP storage system

CLI (from compute instance on Vultr Cloud)

```
mount -t nfs <ontap_select_ip>:/<junction-path> <compute node mount point>
```

```
root@qdrant-compute:~# mount -t nfs 192.0.2.54:/qdrant_prod /mnt/qdrant_prod
```

```
ls /mnt/qdrant_prod
```

```
root@qdrant-compute:~# cd /mnt/qdrant_prod
root@qdrant-compute:/mnt/qdrant_prod# ls
root@qdrant-compute:/mnt/qdrant_prod#
```

Deploy Qdrant Vector DB on prod mount

The **Qdrant** is deployed using **Docker** by installing Docker on the host system, validating network and storage readiness, and running the official Qdrant container image. This enables a consistent and portable deployment model suitable for production environments.

The deployment includes exposing required service ports and mounting the production filesystem (qdrant_prod) to ensure persistent storage for vector data and collections. This approach simplifies management while maintaining an isolated and scalable runtime environment for vector search workloads.

Step 1 | Pre validate docker and qdrant volume mount

Verify that Docker is installed and properly configured on the host system. This validation step ensures that the Docker engine is available, the service (daemon) is running, and the system is ready to support containerized workloads.

Pre-validation typically includes confirming:

- Installed Docker version
- Ensuring that required Qdrant storage is mounted.

Check whether docker is available on the host system

```
root@qdrant-compute:~# docker ps
Command 'docker' not found, but can be installed with:
snap install docker         # version 28.4.0, or
apt  install docker.io      # version 28.2.2-0ubuntu1~24.04.1
apt  install podman-docker  # version 4.9.3+ds1-1ubuntu0.2
See 'snap info docker' for additional versions.
```

If Docker is not already installed or properly configured, install and validate it before proceeding. Otherwise, this step can be skipped.

Installing the docker

```
root@qdrant-compute:~# apt install docker.io -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base pigz runc ubuntu-fan
Suggested packages:
  ifupdown aufs-tools cgroupfs-mount | cgroup-lite debootstrap docker-buildx docker-compose-v2
docker-doc rinse zfs-fuse | zfsutils
The following NEW packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base docker.io pigz runc ubuntu-fan
0 upgraded, 8 newly installed, 0 to remove and 10 not upgraded.
```

CLI Validation

Validate the Docker version and status after installation.

```
root@qdrant-compute:~# docker --version
Docker version 28.2.2, build 28.2.2-0ubuntu1~24.04.1
```

Expected Output:

- Docker version <version>, build <build-id>

Step 2 | Verify storage Mount points

```
root@qdrant-compute:~# df -h
Filesystem                Size  Used Avail Use% Mounted on
tmpfs                     794M  1.3M  793M   1% /run
efivarfs                  256K   25K  227K  10% /sys/firmware/efi/efivars
/dev/vda2                  47G  8.9G   36G  20% /
tmpfs                     3.9G     0  3.9G   0% /dev/shm
tmpfs                     5.0M     0  5.0M   0% /run/lock
/dev/vda1                 511M  6.2M  505M   2% /boot/efi
tmpfs                     794M   12K  794M   1% /run/user/0
192.0.2.54:/qdrant_prod   95G  2.1M   95G   1% /mnt/qdrant_prod
```

Expected Output:

- Filesystem 192.0.2.54:**/qdrant_prod** is mounted on **/mnt/qdrant_prod**

Step 3 | Create a storage directory

Create a storage sub directory inside the **qdrant_prod** mount point.

```
root@qdrant-compute:~# mkdir -p /mnt/qdrant_prod/storage
```

CLI Validation

```
root@qdrant-compute:~# ls /mnt/qdrant_prod
storage
```

Step 4 | Deploy Qdrant Container

Run Qdrant using the official Docker image without specifying a tag. By default, Docker pulls the latest available version.

```
root@qdrant-compute:~# docker run -d   --name qdrant   -p 6333:6333   -p 6334:6334   -e
QDRANT__STORAGE__ON_DISK_PAYLOAD=true   -v
/mnt/qdrant_prod/storage:/qdrant/storage  qdrant/qdrant
98f84d8bfbdcd9659c49b6a3f074e327d7bad20ae61a9aaeadcfa6277494a05b
```

Container Runtime Parameters

- **-d** - Runs the container in Detached Mode
- **--name** - Assigns the container a readable name
- **-p** - Map a port from the host machine to a port inside the container
- **-e** - Sets an environment variable inside the container
- **-v** - Creates a volume mount that maps the host directory

CLI Validation

```
root@qdrant-compute:~# docker ps
CONTAINER ID   IMAGE          COMMAND           CREATED        STATUS        PORTS       NAMES
98f84d8bfbdc   qdrant/qdrant   "./entrypoint.sh"  7 seconds ago   Up 7 seconds   0.0.0.0:6333-
6334->6333-6334/tcp, [::]:6333-6334->6333-6334/tcp   qdrant
```

Expected Output:

- Image Name = Qdrant
- STATUS = up
- PORTS= 6333-6334->6333-6334/tcp

Deploy a DEV/TEST FlexClone volume for Qdrant database

Once data ingestion to /mnt/qdrant_prod is complete, FlexClone volumes can be created for the DEV and TEST environments. These provide space-efficient, writable copies of the production dataset, allowing development and testing activities to proceed without affecting production.

A FlexClone volume in NetApp ONTAP is created almost instantly from an existing volume or snapshot without duplicating the underlying data, which minimizes additional storage usage, because a FlexClone is a point-in-time copy and does not automatically re-sync with the source, this step should be performed only after the production data load is finalized. This ensures the DEV and TEST environments are based on the intended production baseline for validation and testing.

Create a DEV FlexClone volume for Qdrant database

Step 1 | Create a DEV flexclone volume

Create a development FlexClone to provision a writable copy of the dataset for testing or development activities without impacting the source data.

```
ONTAPSelectCluster::> vol clone create -vserver SVM_qdrant -flexclone qdrant_dev -type RW -parent-vserver SVM_qdrant -parent-volume  qdrant_prod

[Job 494] Job succeeded: Successful

ONTAPSelectCluster::>
```

- Flexclone volume create

  Initiates the creation of a qdrant DB dev flexclone volume on the destination cluster.

- vserver <SVM_Name>

  Specifies the Storage Virtual Machine (SVM) where the qdrant DB dev volume will be created.

- parent-vserver <SVM_Name>

  Indicates the Storage Virtual Machine (SVM) that serves as the source for creating the Qdrant database

- parent-volume <volume_Name>

  Specifies the name of the parent volume on the SVM from which the Qdrant database volume will be derived.

Step 2 | Check the volume clone status

Check the clone status to verify that the FlexClone volume has been successfully created and is available for use.

```
ONTAPSelectCluster::> volume clone show

                      Parent  Parent        Parent
Vserver FlexClone     Vserver Volume        Snapshot            State     Type

------- ------------ ------- ------------ -------------------- --------- ----

SVM_qdrant qdrant_dev SVM_qdrant qdrant_prod clone_qdrant_dev.2026-03-09_044951.0 online RW

ONTAPSelectCluster::>
```

Expected Output:

- Flex clone volume **qdrant_dev** created successfully
- Mounted on **Vserver SVM_qdrant**
- Volume status is **online** and accessible to clients
- Confirm the snapshot **clone_qdrant_dev.2026-03-09_044951.0** created successfully
- Confirms the volume is ready for read/write operations

Step 3 | Check volume clone snapshot status

Check the cloned snapshot status to confirm that the snapshot-based clone has been successfully created and is available for use.

```
ONTAPSelectCluster::> snap list qdrant_prod
                                                                  ---Blocks---
Vserver  Volume   Snapshot                                  Size Total% Used%
-------- -------- ------------------------------------ -------- ------ -----
SVM_qdrant qdrant_prod
                  weekly.2026-03-01_0015                 1.40MB     0%    2%
                  daily.2026-03-08_0010                   160KB     0%    0%
                  weekly.2026-03-08_0015                41.17MB     0%   41%
                  hourly.2026-03-08_2305                  164KB     0%    0%
                  hourly.2026-03-09_0005                  160KB     0%    0%
                  daily.2026-03-09_0010                   164KB     0%    0%
                  hourly.2026-03-09_0105                  164KB     0%    0%
                  hourly.2026-03-09_0205                  156KB     0%    0%
                  hourly.2026-03-09_0305                  164KB     0%    0%
                  hourly.2026-03-09_0405                  164KB     0%    0%
                  clone_qdrant_dev.2026-03-09_044951.0    144KB     0%    0%
11 entries were displayed.
```

Step 4 | Split the cloned volume

Split the clone volume to convert the FlexClone into a fully independent volume by separating it from its parent dataset.

```
ONTAPSelectCluster::> volume clone split start -vserver SVM_qdrant -flexclone qdrant_dev

Warning: Are you sure you want to split clone volume qdrant_dev in Vserver SVM_qdrant ?
{y|n}: y
[Job 495] Job is queued: Split qdrant_dev.

ONTAPSelectCluster::>
```

Step 5 | Check the clone status post-split

Check the clone status after the split to confirm that the FlexClone volume has successfully completed the split process and is now fully independent from the parent volume.

```
ONTAPSelectCluster::> vol clone split show

This table is currently empty.

ONTAPSelectCluster::>
```

```
ONTAPSelectCluster::> vol clone show

This table is currently empty.

ONTAPSelectCluster::>
```

Step 6 | Mount the NetApp Qdrant database Volume with Junction Path

The volume mount command attaches the volume to the SVM's namespace at a junction path, making it visible and mountable by NFS clients. Without a junction path, the volume exists internally but cannot be accessed over NFS.

```
Vol show -vserver <vservername> -volume <volumename> -fields junction-path
```

```
vol show -vserver  SVM_qdrant -volume qdrant_dev -fields junction-path
```

Check the Current Status

Check the current status to verify whether the volume or junction path is already mounted or in use before proceeding with the mount operation.

```
ONTAPSelectCluster::> vol show -vserver  SVM_qdrant -volume qdrant_dev -fields junction-path

vserver    volume     junction-path

---------- ---------- -------------

SVM_qdrant qdrant_dev -

ONTAPSelectCluster::>
```

Mount the volume

Mount the volume to make the storage accessible on the host system, allowing applications and users to read from and write data to the mounted directory.

```
volume mount -vserver <dest_svm>  -volume <cache_vol1>  -junction-path /Junction-path
```

```
ONTAPSelectCluster::> vol mount -volume qdrant_dev -junction-path /qdrant_dev -vserver SVM_qdrant
```

Validation

```
Volume show -volume <volumename> -fields junction-path
```

```
ONTAPSelectCluster::> vol show -vserver  SVM_qdrant -volume qdrant_dev -fields junction-path
vserver    volume     junction-path
---------- ---------- -------------
SVM_qdrant qdrant_dev /qdrant_dev
ONTAPSelectCluster::>
```

Expected Output:

- Volume **qdrant_dev** mounted successfully
- Mounted on **Vserver SVM_qdrant**
- Junction path set to **/qdrant_dev**
- Volume status is **online** and accessible to clients
- Confirms the volume is ready for read/write operations

Step 7 | Check the status of the export-policy

Check the status of the export policy to verify that the appropriate access rules are applied.
Note: cloned volume inherits the export policy from its parent by default.

Check the Export-policy status

```
ONTAPSelectCluster::> vol show qdrant_dev -fields policy
vserver    volume     policy
---------- ---------- -------------
SVM_qdrant qdrant_dev Policy_qdrant
ONTAPSelectCluster::>
```

Check the Export-policy rule status

```
ONTAPSelectCluster::> export-policy rule show -vserver SVM_qdrant -policyname Policy_qdrant
             Policy          Rule    Access   Client                RO
Vserver      Name            Index   Protocol Match                 Rule
------------ --------------- ------  -------- --------------------- ---------
SVM_qdrant   Policy_qdrant   1       nfs3     192.0.2.11          any

ONTAPSelectCluster::>
```

Step 8 | Mount on Vultr Compute

Verification that Vultr compute nodes can access the cloned dataset over NFS ensures the end-to-end path - from **NetApp ONTAP(Vultr) → Vultr compute** - is fully operational. When we mount an NFS export from NetApp ONTAP in Vultr:

- We are mounting the active filesystem of the NetApp FlexClone volume.
- NetApp FlexClone creates a writable clone of the parent volume or snapshot, enabling independent read and write operations without affecting the source data. The clone shares underlying storage blocks with the parent volume, providing space efficiency while maintaining data consistency.

CLI (from compute instance on Vultr Cloud)

```
mount -t nfs <ontap_select_ip>:/<junction-path> <compute node mount point>
```

```
root@qdrant-compute:~# mount -t nfs 192.0.2.54:/qdrant_dev /mnt/qdrant_dev
```

```
ls /mnt/qdrant_dev
```

```
root@qdrant-compute:/mnt/qdrant_dev# cd /mnt/qdrant_dev
root@qdrant-compute:/mnt/qdrant_dev#
root@qdrant-compute:/mnt/qdrant_dev#
root@qdrant-compute:/mnt/qdrant_dev# ls -l
total 8
drwxr-xr-x 3 root root 4096 Mar  2 08:57 demo
drwxr-xr-x 6 root root 4096 Mar  5 06:55 storage
root@qdrant-compute:/mnt/qdrant_dev#
```

Step 9 | Deploy Qdrant Container as a Dev VectorDB on `/mnt/qdrant_dev`

Run Qdrant using the official Docker image without specifying a tag, which pulls the latest version by default. The **/mnt/qdrant_dev** mount is based on the FlexClone created from the production Qdrant volume and is used for Development purposes.

```
root@qdrant-compute:/mnt/qdrant_dev# docker run -d --name qdrant-dev -p 6335:6333   -p
6336:6334   -e QDRANT__STORAGE__ON_DISK_PAYLOAD=true   -v
/mnt/qdrant_dev/storage:/qdrant/storage  qdrant/qdrant

aa119fec70252564bb884c855fb0c32148ac1bf16d541a1b869d15dca8d749fe
```

```
root@qdrant-compute:/# docker ps
CONTAINER ID   IMAGE          COMMAND             CREATED          STATUS          PORTS
                                                  NAMES
aa119fec7025   qdrant/qdrant   "./entrypoint.sh"   10 minutes ago   Up 10 minutes   0.0.0.0:6335->6333/tcp, [::]:6335
->6333/tcp, 0.0.0.0:6336->6334/tcp, [::]:6336->6334/tcp   qdrant-dev
```

Container Runtime Parameters

- **-d** - Runs the container in Detached Mode
- **--name** - Assigns the container a readable name
- **-p** - Map a port from the host machine to a port inside the container
- **-e** - Sets an environment variable inside the container
- **-v** - Creates a volume mount that maps the host directory

Create a TEST FlexClone volume for Qdrant database

The same procedure used for creating the **DEV FlexClone volume** is followed to provision the **TEST FlexClone volume** and mount it on the Qdrant host.

While executing the steps, replace the DEV-specific identifiers with TEST-specific values to ensure correct configuration:

- Replace **qdrant_dev** with **qdrant_test** (volume name)
- Replace the junction path **/qdrant_dev** with **/qdrant_test**
- Replace the Vultr compute mount point **/mnt/qdrant_dev** with **/mnt/qdrant_test**

Ensure these substitutions are applied consistently across all commands and configurations during the FlexClone creation and mount process, and the TEST mount point can then be used to run a Qdrant instance for testing purposes using the dataset available at the time the FlexClone was created.

Setting up NVIDIA Dynamo for serving LLMs

This section describes how to deploy Large Language Models (LLMs) with NVIDIA Dynamo using the vLLM backend. Dynamo supports multiple inference engines, including **vLLM, TensorRT-LLM,** and **SGLang**. In this guide, vLLM is used because Dynamo provides direct vLLM integration for disaggregated serving, KV-aware routing, and OpenAI-compatible serving workflows.

Inference Serving Modes

There are two deployment patterns for Nvidia Dynamo serving:

- **Traditional (Aggregated) Serving :** a single worker handles both **prefill** and **decode**
- **Disaggregated serving:** prefill and decode run on separate worker types, with KV-cache transfer between them through NIXL-based mechanisms in Dynamo's vLLM integration

Traditional (Aggregated) serving

In aggregated mode, one worker performs both prompt processing and token generation. This is the simpler deployment model and is generally preferred for development, functional validation, and moderate-load environments. In NVIDIA's vLLM deployment examples, aggregated deployment is implemented with a frontend and a single VLLMDecodeWorker that handles both phases.

Use aggregated mode when:

- Simplicity and faster setup matter most
- The environment is intended for development or testing
- Workloads are moderate and do not require separate tuning of prefill and decode behaviour
- Operational overhead should be minimized

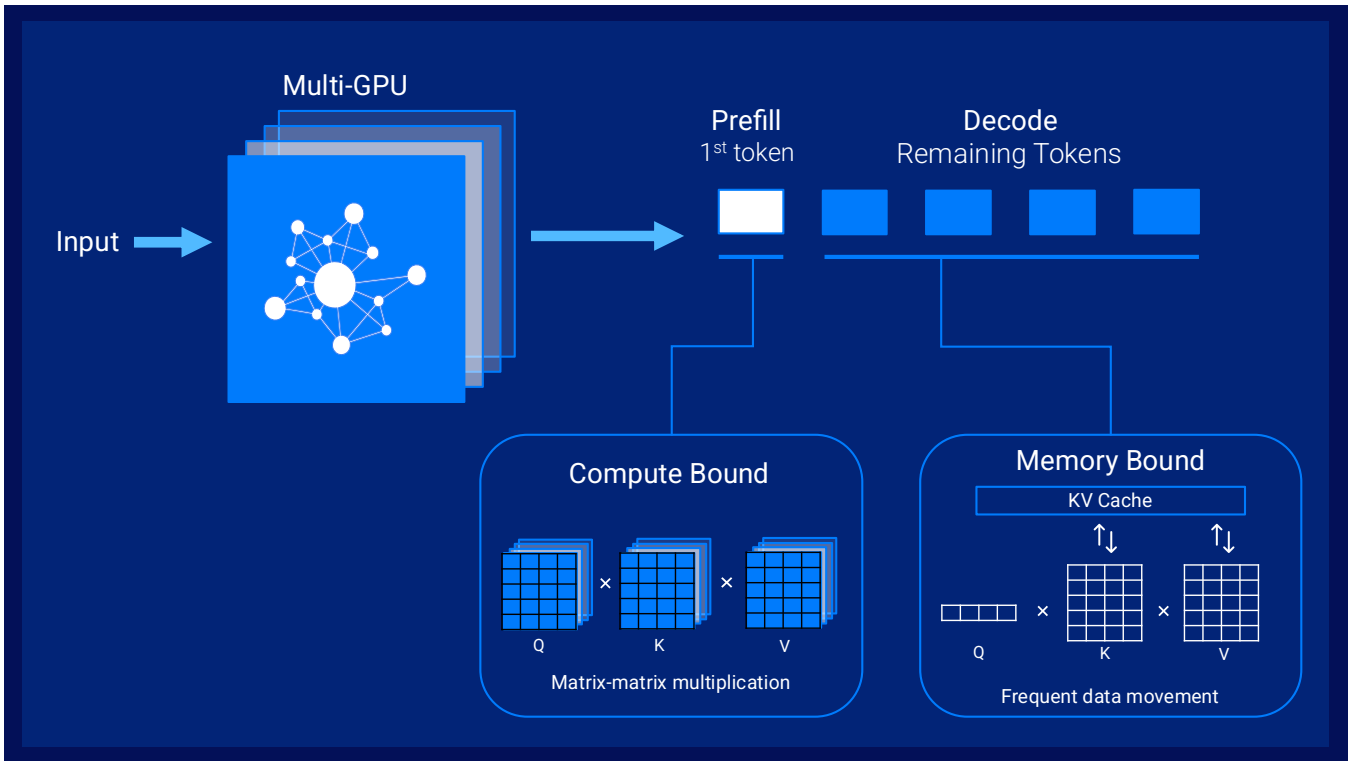


Figure 9 - Traditional (Aggregated) Inference Workflow (Prefill and Decode Coupled)

Disaggregated serving

In disaggregated mode, the prefill phase and decode phase are separated into different workers. NVIDIA documents this architecture as enabling independent scaling, better resource utilization, and KV-cache transfer between workers. This is useful because prefill is typically more compute-bound, while decode is more memory-bound.

Use disaggregated mode when:

- The workload is prompt-heavy, such as RAG or document Q&A
- Tighter control of TTFT and inter-token latency is required
- Higher concurrency causes prefill activity to interfere with steady decode performance
- Prefill and decode phases need to be tuned or scaled independently

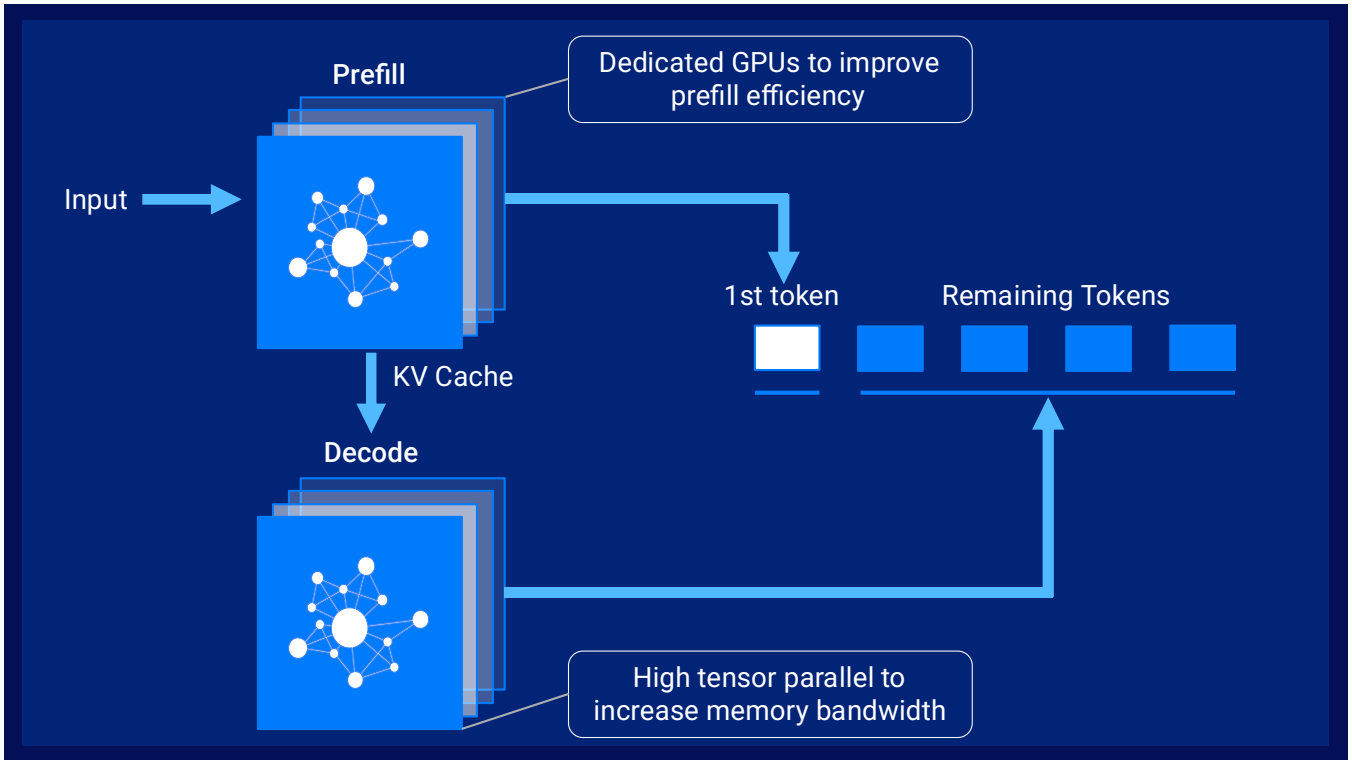


Figure 10 - Disaggregated Inference Workflow with KV Cache Transfer

Considerations for disaggregated mode:

- Deployment is more complex than aggregated mode
- Separate worker roles must be configured and managed
- KV-cache transfer introduces an additional dependency path between worker types
- Model memory planning must account for separate serving components

Deployment Overview

This guide deploys NVIDIA Dynamo with the **vLLM runtime** for LLM inference.

The deployment workflow includes:

- Validating access to the GPU server
- Confirming GPU, driver, CUDA, and storage readiness
- Selecting a compatible Dynamo container/runtime
- Deploying the model-serving stack
- Validating endpoint availability and inference behaviour

System Access

Before performing deployment steps, verify command-line access to the GPU server where the deployment will run.

SSH Connection to GPU Server

Connect to the GPU server using SSH:

```
ssh root@<NVIDIA_GPU_SERVER_IP>
```

```
PS C:\Users\xxxxxxxx> ssh root@192.0.2.101
root@192.0.2.101's password:
Welcome to Ubuntu 24.04.4 LTS (GNU/Linux 6.8.0-101-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

System information as of Tue Mar 10 09:40:36 AM UTC 2026

  System load:  0.05
  Usage of /:   9.5% of 837.71GB
  Memory usage: 1%
  Swap usage:   0%
  Temperature:  61.0 C
  Processes:    2243
  Users logged in: 1
  IPv4 address for eno12399np0: 192.0.2.101
  IPv6 address for eno12399np0: 2001:db8::10

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
   just raised the bar for easy, resilient and secure K8s cluster deployment.

   https://ubuntu.com/engage/secure-kubernetes-at-the-edge
```

```
  Expanded Security Maintenance for Applications is not enabled.

  6 updates can be applied immediately.
  To see these additional updates run: apt list --upgradable

  Enable ESM Apps to receive additional future security updates.
  See https://ubuntu.com/esm or run: sudo pro status

  *** System restart required ***
  Pending processor microcode upgrade!
  Diagnostics:
    The currently running processor microcode revision is 0x2b000643 which is not the expected
  microcode
    revision 0x2b000661.
  Last login: Tue Mar 10 09:28:58 2026 from 192.0.2.166
  root@phase3-nvidia:~#
```

Note: Replace <NVIDIA_GPU_SERVER_IP> with the IP address of the GPU host.

Expected Output:

- successful SSH authentication
- root shell access
- ability to execute administrative commands

All remaining steps in this guide are expected to be run on this GPU server unless stated otherwise.

Infrastructure and GPU Validation

Before deploying Dynamo, validate that the host is ready for GPU-based AI inference. This helps prevent failures caused by GPU visibility issues, driver or CUDA mismatches, missing container support, insufficient storage, or model-download/authentication problems.

This section verifies:

- GPU detection
- GPU topology
- CUDA compatibility
- Disk capacity

Verify GPU Detection

Confirm that the system detects all installed GPUs:

```
nvidia-smi
```

```
root@phase3-nvidia:~# nvidia-smi
Tue Mar 10 09:41:57 2026
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 580.95.05              Driver Version: 580.95.05      CUDA Version: 13.0     |
+-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA A100-SXM4-80GB          On  |   00000000:1C:00.0 Off |                  Off |
| N/A   39C    P0             67W /  500W |       0MiB /  81920MiB |      0%      Default |
|                                         |                        |             Disabled |
+-----------------------------------------+------------------------+----------------------+
|   1  NVIDIA A100-SXM4-80GB          On  |   00000000:3E:00.0 Off |                  Off |
| N/A   38C    P0             68W /  500W |       0MiB /  81920MiB |      0%      Default |
|                                         |                        |             Disabled |
+-----------------------------------------+------------------------+----------------------+
|   2  NVIDIA A100-SXM4-80GB          On  |   00000000:4F:00.0 Off |                  Off |
| N/A   37C    P0             60W /  500W |       0MiB /  81920MiB |      0%      Default |
|                                         |                        |             Disabled |
+-----------------------------------------+------------------------+----------------------+
|   3  NVIDIA A100-SXM4-80GB          On  |   00000000:60:00.0 Off |                    0 |
| N/A   38C    P0             64W /  500W |       0MiB /  81920MiB |      0%      Default |
|                                         |                        |             Disabled |
+-----------------------------------------+------------------------+----------------------+
+-----------------------------------------+------------------------+----------------------+
|   4  NVIDIA A100-SXM4-80GB          On  |   00000000:9E:00.0 Off |                    0 |
| N/A   40C    P0             68W /  500W |       0MiB /  81920MiB |      0%      Default |
|                                         |                        |             Disabled |
+-----------------------------------------+------------------------+----------------------+
|   5  NVIDIA A100-SXM4-80GB          On  |   00000000:BE:00.0 Off |                    0 |
| N/A   38C    P0             70W /  500W |       0MiB /  81920MiB |      0%      Default |
|                                         |                        |             Disabled |
+-----------------------------------------+------------------------+----------------------+
|   6  NVIDIA A100-SXM4-80GB          On  |   00000000:CE:00.0 Off |                    0 |
| N/A   40C    P0             67W /  500W |       0MiB /  81920MiB |      0%      Default |
|                                         |                        |             Disabled |
+-----------------------------------------+------------------------+----------------------+
|   7  NVIDIA A100-SXM4-80GB          On  |   00000000:DE:00.0 Off |                    0 |
| N/A   40C    P0             66W /  500W |       0MiB /  81920MiB |      0%      Default |
|                                         |                        |             Disabled |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI              PID   Type   Process name                        GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|  No running processes found                                                             |
+-----------------------------------------------------------------------------------------+
root@phase3-nvidia:~#
```

Expected Output:

- All 8 GPUs are visible
- Driver version is displayed (e.g., 580.95.05)
- CUDA version is shown (e.g., 13.0)
- No GPU errors are reported.

If nvidia-smi fails, NVIDIA drivers must be installed:

Check available driver versions:

```
apt search nvidia-driver | grep "^nvidia-driver-[0-9]"
```

```
root@phase3-nvidia:~# apt search nvidia-driver | grep "^nvidia-driver-[0-9]"

WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

nvidia-driver-460/noble-updates,noble-security 470.256.02-0ubuntu0.24.04.1 amd64
nvidia-driver-460-server/noble-updates,noble-security 470.256.02-0ubuntu0.24.04.1 amd64
nvidia-driver-465/noble-updates,noble-security 470.256.02-0ubuntu0.24.04.1 amd64
nvidia-driver-470/noble-updates,noble-security 470.256.02-0ubuntu0.24.04.1 amd64
nvidia-driver-470-server/noble-updates,noble-security 470.256.02-0ubuntu0.24.04.1 amd64
nvidia-driver-510/noble 525.147.05-0ubuntu2 amd64
nvidia-driver-515/noble 525.147.05-0ubuntu2 amd64
nvidia-driver-515-open/noble 525.147.05-0ubuntu2 amd64
nvidia-driver-515-server/noble 525.147.05-0ubuntu2 amd64
nvidia-driver-520/noble 525.147.05-0ubuntu2 amd64
nvidia-driver-520-open/noble 525.147.05-0ubuntu2 amd64
nvidia-driver-525/noble 525.147.05-0ubuntu2 amd64
nvidia-driver-525-open/noble 525.147.05-0ubuntu2 amd64
nvidia-driver-525-server/noble 525.147.05-0ubuntu2 amd64
nvidia-driver-530/unknown 560.35.03-0ubuntu1 amd64
nvidia-driver-530-open/unknown 560.35.03-0ubuntu1 amd64
nvidia-driver-535/noble-updates,noble-security 535.288.01-0ubuntu0.24.04.2 amd64
nvidia-driver-535-open/noble-updates,noble-security 535.288.01-0ubuntu0.24.04.2 amd64
nvidia-driver-535-server/noble-updates,noble-security 535.288.01-0ubuntu0.24.04.3 amd64
nvidia-driver-535-server-open/noble-updates,noble-security 535.288.01-0ubuntu0.24.04.3 amd64
nvidia-driver-550/unknown 550.163.01-0ubuntu1 amd64
nvidia-driver-550-open/unknown 550.163.01-0ubuntu1 amd64
nvidia-driver-550-server/noble-updates 550.163.01-0ubuntu0.24.04.2 amd64
nvidia-driver-550-server-open/noble-updates 550.163.01-0ubuntu0.24.04.2 amd64
nvidia-driver-555/unknown 555.42.06-0ubuntu1 amd64
nvidia-driver-555-open/unknown 555.42.06-0ubuntu1 amd64
nvidia-driver-560/unknown 560.35.05-0ubuntu1 amd64
nvidia-driver-560-open/unknown 560.35.05-0ubuntu1 amd64
nvidia-driver-565/unknown 565.57.01-0ubuntu1 amd64
nvidia-driver-565-open/unknown 565.57.01-0ubuntu1 amd64
nvidia-driver-565-server/noble-updates 565.57.01-0ubuntu0.24.04.3 amd64
nvidia-driver-565-server-open/noble-updates 565.57.01-0ubuntu0.24.04.3 amd64
nvidia-driver-570/unknown 570.211.01-0ubuntu1 amd64
nvidia-driver-570-open/unknown 570.211.01-0ubuntu1 amd64
nvidia-driver-570-server/noble-updates,noble-security 570.211.01-0ubuntu0.24.04.2 amd64
nvidia-driver-570-server-open/noble-updates,noble-security 570.211.01-0ubuntu0.24.04.2 amd64
nvidia-driver-575/unknown 575.57.08-0ubuntu1 amd64
nvidia-driver-575-open/unknown 575.57.08-0ubuntu1 amd64
nvidia-driver-575-server/noble-updates,noble-security 575.57.08-0ubuntu0.24.04.3 amd64
nvidia-driver-575-server-open/noble-updates,noble-security 575.57.08-0ubuntu0.24.04.3 amd64
nvidia-driver-580/unknown 580.95.05-0ubuntu1 amd64
nvidia-driver-580-open/unknown 580.95.05-0ubuntu1 amd64
nvidia-driver-580-server/noble-updates,noble-security 580.126.09-0ubuntu0.24.04.2 amd64
nvidia-driver-580-server-open/noble-updates,noble-security 580.126.09-0ubuntu0.24.04.2 amd64
nvidia-driver-590/noble-updates,noble-security 590.48.01-0ubuntu0.24.04.4 amd64
nvidia-driver-590-open/noble-updates,noble-security 590.48.01-0ubuntu0.24.04.4 amd64
nvidia-driver-590-server-open/noble-updates,noble-security 590.48.01-0ubuntu0.24.04.4 amd64
root@phase3-nvidia:~#
```

Install the latest available driver (replace 580 with the latest version from search):

```
sudo apt update

sudo apt install -y nvidia-driver-580

sudo reboot
```

After reboot, verify GPU detection:

```
nvidia-smi
```

Verify GPU Topology

Check GPU interconnect topology to understand communication pathways between GPUs:

```
nvidia-smi topo -m
```

```
root@phase3-nvidia:~# nvidia-smi topo -m
        GPU0    GPU1    GPU2    GPU3    GPU4    GPU5    GPU6    GPU7    NIC0    NIC1    NIC2    NIC3    NIC4    NIC5    NIC6    NIC7    NIC8    NIC9    CPU Affinity    NUMA Affinity   GPU NUMA
_ID
GPU0     X      NV12    NV12    NV12    NV12    NV12    NV12    NV12    PXB     PXB     NODE    NODE    NODE    SYS     SYS     SYS     SYS     SYS     0,2,4,6,8,10    0               N/A
GPU1    NV12     X      NV12    NV12    NV12    NV12    NV12    NV12    NODE    NODE    PXB     NODE    NODE    SYS     SYS     SYS     SYS     SYS     0,2,4,6,8,10    0               N/A
GPU2    NV12    NV12     X      NV12    NV12    NV12    NV12    NV12    NODE    NODE    NODE    PXB     NODE    SYS     SYS     SYS     SYS     SYS     0,2,4,6,8,10    0               N/A
GPU3    NV12    NV12    NV12     X      NV12    NV12    NV12    NV12    NODE    NODE    NODE    NODE    PXB     SYS     SYS     SYS     SYS     SYS     0,2,4,6,8,10    0               N/A
GPU4    NV12    NV12    NV12    NV12     X      NV12    NV12    NV12    SYS     SYS     SYS     SYS     SYS     PXB     PXB     NODE    NODE    NODE    1,3,5,7,9,11    1               N/A
GPU5    NV12    NV12    NV12    NV12    NV12     X      NV12    NV12    SYS     SYS     SYS     SYS     SYS     NODE    NODE    PXB     NODE    NODE    1,3,5,7,9,11    1               N/A
GPU6    NV12    NV12    NV12    NV12    NV12    NV12     X      NV12    SYS     SYS     SYS     SYS     SYS     NODE    NODE    NODE    PXB     NODE    1,3,5,7,9,11    1               N/A
GPU7    NV12    NV12    NV12    NV12    NV12    NV12    NV12     X      SYS     SYS     SYS     SYS     SYS     NODE    NODE    NODE    NODE    PXB     1,3,5,7,9,11    1               N/A
NIC0    PXB     NODE    NODE    NODE    SYS     SYS     SYS     SYS      X      PIX     NODE    NODE    NODE    SYS     SYS     SYS     SYS     SYS
NIC1    PXB     NODE    NODE    NODE    SYS     SYS     SYS     SYS     PIX      X      NODE    NODE    NODE    SYS     SYS     SYS     SYS     SYS
NIC2    NODE    PXB     NODE    NODE    SYS     SYS     SYS     SYS     NODE    NODE     X      NODE    NODE    SYS     SYS     SYS     SYS     SYS
NIC3    NODE    NODE    PXB     NODE    SYS     SYS     SYS     SYS     NODE    NODE    NODE     X      NODE    SYS     SYS     SYS     SYS     SYS
NIC4    NODE    NODE    NODE    PXB     SYS     SYS     SYS     SYS     NODE    NODE    NODE    NODE     X      SYS     SYS     SYS     SYS     SYS
NIC5    SYS     SYS     SYS     SYS     PXB     NODE    NODE    NODE    SYS     SYS     SYS     SYS     SYS      X      PIX     NODE    NODE    NODE
NIC6    SYS     SYS     SYS     SYS     PXB     NODE    NODE    NODE    SYS     SYS     SYS     SYS     SYS     PIX      X      NODE    NODE    NODE
NIC7    SYS     SYS     SYS     SYS     NODE    PXB     NODE    NODE    SYS     SYS     SYS     SYS     SYS     NODE    NODE     X      NODE    NODE
NIC8    SYS     SYS     SYS     SYS     NODE    NODE    PXB     NODE    SYS     SYS     SYS     SYS     SYS     NODE    NODE    NODE     X      NODE
NIC9    SYS     SYS     SYS     SYS     NODE    NODE    NODE    PXB     SYS     SYS     SYS     SYS     SYS     NODE    NODE    NODE    NODE     X

Legend:

  X    = Self
  SYS  = Connection traversing PCIe as well as the SMP interconnect between NUMA nodes (e.g., QPI/UPI)
  NODE = Connection traversing PCIe as well as the interconnect between PCIe Host Bridges within a NUMA node
  PHB  = Connection traversing PCIe as well as a PCIe Host Bridge (typically the CPU)
  PXB  = Connection traversing multiple PCIe bridges (without traversing the PCIe Host Bridge)
  PIX  = Connection traversing at most a single PCIe bridge
  NV#  = Connection traversing a bonded set of # NVLinks

NIC Legend:

  NIC0: mlx5_0
  NIC1: mlx5_1
  NIC2: mlx5_2
  NIC3: mlx5_3
  NIC4: mlx5_4
  NIC5: mlx5_5
  NIC6: mlx5_6
  NIC7: mlx5_7
  NIC8: mlx5_8
  NIC9: mlx5_9

root@phase3-nvidia:~#
```

This command displays how GPUs communicate with each other.

Important indicators:

- NV# → NVLink connection (high-speed GPU-to-GPU interconnect)
- PHB → PCIe host bridge (CPU-to-GPU connection)
- SYS → System interconnect (cross-socket communication)

For multi-GPU inference, faster GPU-to-GPU connectivity is beneficial, especially when workloads or serving components exchange data across devices

Verify CUDA Compatibility

Confirm the CUDA version supported by the installed Nvidia driver:

```
nvidia-smi | grep "CUDA Version"
```

```
root@phase3-nvidia:~# nvidia-smi | grep "CUDA Version"
| NVIDIA-SMI 580.95.05    Driver Version: 580.95.05    CUDA Version: 13.0 |
root@phase3-nvidia:~#
```

Expected result:

- A line showing the available CUDA version, such as CUDA Version: 12.x or 13.x

Use this value when selecting the container image or runtime version so that the container stack remains compatible with the host GPU driver environment.

Check Disk Capacity

Verify that the server has enough free space for model weights, cache, logs, and temporary data:

```
df -h
```

```
root@phase3-nvidia:~# df -h
Filesystem      Size  Used Avail Use% Mounted on
tmpfs           202G  5.0M  202G   1% /run
efivarfs        304K  235K   65K  79% /sys/firmware/efi/efivars
/dev/nvme0n1p2  838G   87G  715G  11% /
tmpfs           1008G 170G 1008G   1% /dev/shm
tmpfs           5.0M     0  5.0M   0% /run/lock
/dev/nvme1n1p1  3.5T   28K  3.3T   1% /data
/dev/nvme3n1p1  3.5T   28K  3.3T   1% /data2
/dev/nvme2n1p1  3.5T   28K  3.3T   1% /data3
/dev/nvme6n1p1  3.5T   28K  3.3T   1% /data5
/dev/nvme8n1p1  3.5T   28K  3.3T   1% /data6
/dev/nvme7n1p1  3.5T   28K  3.3T   1% /data4
/dev/nvme4n1p1  3.5T   28K  3.3T   1% /data8
/dev/nvme5n1p1  3.5T   28K  3.3T   1% /data7
/dev/nvme0n1p1  511M  7.2M  504M   2% /boot/efi
tmpfs           202G   16K  202G   1% /run/user/0
root@phase3-nvidia:~#
```

Expected Output:

- Sufficient available space on the target *root filesystem /*

Note: Model weights for the 49B parameter model require approximately 100GB, with additional space needed for cache, logs, and temporary files.

Model Repository Access and Hugging Face Authentication

Model weights for NVIDIA and open-source models are distributed through Hugging Face, a centralized repository for machine learning models. Access to these models requires authentication via a Hugging Face account and API token. This token enables automated model downloads during container initialization and runtime operations.

The token must be persisted in the shell environment to ensure containers can authenticate when downloading models, both during initial deployment and after system restarts.

- Create account at https://huggingface.co/join
- Generate access token at https://huggingface.co/settings/tokens
    - Token type: "Read" access is sufficient
    - Name it something like "vultr-rag-deployment"
- Request access to models (if required - wait for approval, usually instant):
    - https://huggingface.co/nvidia/Llama-3_3-Nemotron-Super-49B-v1_5
    - https://huggingface.co/BAAI/bge-m3
    - https://huggingface.co/BAAI/bge-reranker-v2-m3
    - https://huggingface.co/nvidia/NVIDIA-Nemotron-Parse-v1.1

Export token for use in deployment:

```
export HF_TOKEN=hf_xxxxxxxxxxxxxxxxx
```

Persist token in shell profile (*Keeps this token active*):

```
echo "export HF_TOKEN=hf_ xxxxxxxxxxxxxxxxx " >> ~/.bashrc
```

Reload bashrc:

```
source ~/.bashrc
```

```
root@phase3-nvidia:~# export HF_TOKEN=hf_ xxxxxxxxxxxxxxxxx
root@phase3-nvidia:~# echo "export HF_TOKEN=hf_ xxxxxxxxxxxxxxxxx " >> ~/.bashrc
root@phase3-nvidia:~# source ~/.bashrc
root@phase3-nvidia:~#
```

Deployment Considerations

- Model Source: All model weights are downloaded from Hugging Face
- Container Source: Container images are pulled from NGC (nvcr.io)
- Token Persistence: Token must remain active - containers require it for runtime operations and restarts
- Token Security: Treat the token as a credential - it grants access to the Hugging Face account
- Automatic Downloads: Models are downloaded automatically by containers on first run

Host Preparation

The host preparation phase ensures that the system is fully configured to support GPU-accelerated container workloads. This includes installing required system dependencies, validating the container runtime, confirming NVIDIA driver availability, and configuring authentication for both Hugging Face and NVIDIA GPU Cloud (NGC) repositories.

These components form the foundation for running GPU-enabled containers and accessing model and container artifacts required for deployment.

A properly prepared host helps prevent common deployment issues such as GPU detection failures, driver incompatibility, missing container dependencies, or authentication failures when pulling models and container images.

Update system packages

Update the system package index and upgrade existing packages to ensure compatibility with required tools and drivers.

```
sudo apt update && sudo apt upgrade -y
```

```
root@phase3-nvidia:~# sudo apt update && sudo apt upgrade -y
Get:1 https://nvidia.github.io/libnvidia-container/stable/deb/amd64  InRelease [1,477 B]
Hit:2 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2404/x86_64  InRelease
Hit:3 http://ubuntu.mirror.constant.com noble InRelease
Hit:4 http://ubuntu.mirror.constant.com noble-updates InRelease
Hit:5 https://download.docker.com/linux/ubuntu noble InRelease
Hit:6 http://ubuntu.mirror.constant.com noble-backports InRelease
Ign:7 http://linux.mellanox.com/public/repo/mlnx_ofed/24.10-1.1.4.0/ubuntu24.04/amd64 ./ InRelease
Hit:8 http://security.ubuntu.com/ubuntu noble-security InRelease
Hit:9 http://linux.mellanox.com/public/repo/mlnx_ofed/24.10-1.1.4.0/ubuntu24.04/amd64 ./ Release
Hit:11 https://deb.frrouting.org/frr noble InRelease
Fetched 1,477 B in 1s (2,511 B/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
All packages are up to date.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done
The following packages were automatically installed and are no longer required:
 cuda-cccl-13-1 cuda-command-line-tools-13-1 cuda-compiler-13-1 cuda-crt-13-1 cuda-cudart-13-1
 cuda-cudart-dev-13-1 cuda-culibos-dev-13-1 cuda-cuobjdump-13-1 cuda-cupti-13-1 cuda-cupti-dev-13-
1
 cuda-cuxxfilt-13-1 cuda-documentation-13-1 cuda-driver-dev-13-1 cuda-gdb-13-1 cuda-libraries-13-1
 cuda-libraries-dev-13-1 cuda-nsight-13-1 cuda-nsight-compute-13-1 cuda-nsight-systems-13-1
 cuda-nvcc-13-1 cuda-nvdisasm-13-1 cuda-nvml-dev-13-1 cuda-nvprune-13-1 cuda-nvrtc-13-1
 cuda-nvrtc-dev-13-1 cuda-nvtx-13-1 cuda-opencl-13-1 cuda-profiler-api-13-1 cuda-sandbox-dev-13-1
 cuda-sanitizer-13-1 cuda-toolkit-13-1 cuda-toolkit-13-1-config-common cuda-tools-13-1
 cuda-visual-tools-13-1 gds-tools-13-1 libcublas-13-1 libcublas-dev-13-1 libcufft-13-1
```

```
 libcufft-dev-13-1 libcuFile-13-1 libcuFile-dev-13-1 libcuobjclient-13-1 libcuobjclient-dev-13-1
 libcurand-13-1 libcurand-dev-13-1 libcusolver-13-1 libcusolver-dev-13-1 libcusparse-13-1
 libcusparse-dev-13-1 libnpp-13-1 libnpp-dev-13-1 libnvfatbin-13-1 libnvfatbin-dev-13-1
 libnvjitlink-13-1 libnvjitlink-dev-13-1 libnvjpeg-13-1 libnvjpeg-dev-13-1
 libnvptxcompiler-13-1 libnvvm-13-1 nsight-compute-2025.4.1 nsight-systems-2025.5.2
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
root@phase3-nvidia:~#
```

Install prerequisites

Install essential packages required for building, running containers, and managing dependencies. These packages provide compiler tools, kernel headers for driver modules, Python support, and utilities required for automation and downloads.

```
sudo apt install -y build-essential linux-headers-$(uname -r) dkms python3-dev python3-pip
python3-venv git curl wget jq
```

```
root@phase3-nvidia:~# sudo apt install -y build-essential linux-headers-$(uname -r) dkms python3-
dev python3-pip python3-venv git curl wget jq
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
build-essential is already the newest version (12.10ubuntu1).
linux-headers-6.8.0-101-generic is already the newest version (6.8.0-101.101).
dkms is already the newest version (1:3.3.0-1ubuntu1).
python3-dev is already the newest version (3.12.3-0ubuntu2.1).
python3-pip is already the newest version (24.0+dfsg-1ubuntu1.3).
python3-venv is already the newest version (3.12.3-0ubuntu2.1).
git is already the newest version (1:2.43.0-1ubuntu7.3).
curl is already the newest version (8.5.0-2ubuntu10.7).
wget is already the newest version (1.21.4-1ubuntu4.1).
jq is already the newest version (1.7.1-3ubuntu0.24.04.1).
The following packages were automatically installed and are no longer required:
 cuda-cccl-13-1 cuda-command-line-tools-13-1 cuda-compiler-13-1 cuda-crt-13-1 cuda-cudart-13-1
 cuda-cudart-dev-13-1 cuda-culibos-dev-13-1 cuda-cuobjdump-13-1 cuda-cupti-13-1 cuda-cupti-dev-13-
1
 cuda-cuxxfilt-13-1 cuda-documentation-13-1 cuda-driver-dev-13-1 cuda-gdb-13-1 cuda-libraries-13-1
 cuda-libraries-dev-13-1 cuda-nsight-13-1 cuda-nsight-compute-13-1 cuda-nsight-systems-13-1
 cuda-nvcc-13-1 cuda-nvdisasm-13-1 cuda-nvml-dev-13-1 cuda-nvprune-13-1 cuda-nvrtc-13-1
 cuda-nvrtc-dev-13-1 cuda-nvtx-13-1 cuda-opencl-13-1 cuda-profiler-api-13-1 cuda-sandbox-dev-13-1
 cuda-sanitizer-13-1 cuda-tileiras-13-1 cuda-toolkit-13-1 cuda-toolkit-13-1-config-common
 cuda-tools-13-1 cuda-visual-tools-13-1 gds-tools-13-1 libcublas-13-1 libcublas-dev-13-1
 libcufft-13-1 libcufft-dev-13-1 libcufile-13-1 libcufile-dev-13-1 libcuobjclient-13-1
 libcuobjclient-dev-13-1 libcurand-13-1 libcurand-dev-13-1 libcusolver-13-1 libcusolver-dev-13-1
 libcusparse-13-1 libcusparse-dev-13-1 libnpp-13-1 libnpp-dev-13-1 libnvfatbin-13-1
 libnvfatbin-dev-13-1 libnvjitlink-13-1 libnvjitlink-dev-13-1 libnvjpeg-13-1 libnvjpeg-dev-13-1
 libnvptxcompiler-13-1 libnvvm-13-1 nsight-compute-2025.4.1 nsight-systems-2025.5.2
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
root@phase3-nvidia:~#
```

Verify Docker is installed

```
docker --version
```

```
root@phase3-nvidia:~# docker --version
Docker version 29.3.0, build 5927d80
root@phase3-nvidia:~#
```

Expected Output:

- Docker version 29.x.x or similar

Verify Docker Compose (v2 or higher)

```
docker compose version
```

```
root@phase3-nvidia:~# docker compose version
Docker Compose version v5.1.0
root@phase3-nvidia:~#
```

Expected Output:

- Docker Compose version v2.x.x or higher

Enable Docker service and add user to docker group

This ensures Docker starts automatically and allows the current user to run Docker commands without requiring *sudo*.

```
sudo systemctl enable --now docker
sudo usermod -aG docker $USER
newgrp docker
```

```
root@phase3-nvidia:~# sudo systemctl enable --now docker
Synchronizing state of docker.service with SysV service script with /usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable docker
root@phase3-nvidia:~# sudo usermod -aG docker $USER
root@phase3-nvidia:~# newgrp docker
root@phase3-nvidia:~#
```

Install NVIDIA Container Toolkit

The NVIDIA Container Toolkit enables Docker containers to access GPU resources. This toolkit provides the runtime components necessary for GPU-accelerated containers to communicate with the host GPU drivers. Installation involves adding the NVIDIA repository, installing the toolkit package, configuring the Docker runtime, and verifying GPU access from within containers.

Add NVIDIA Container Toolkit repository and GPG key

Add the NVIDIA Container Toolkit GPG key

```
curl -fsSL https://nvidia.github.io/libnvidia-container/gpgkey | sudo gpg --dearmor -o
/usr/share/keyrings/nvidia-container-toolkit-keyring.gpg
```

```
root@phase3-nvidia:~# curl -fsSL https://nvidia.github.io/libnvidia-container/gpgkey | sudo gpg --
dearmor -o /usr/share/keyrings/nvidia-container-toolkit-keyring.gpg

root@phase3-nvidia:~#
```

Add the NVIDIA Container Toolkit repository

```
curl -s -L https://nvidia.github.io/libnvidia-container/stable/deb/nvidia-container-toolkit.list |
sed 's#deb https://#deb [signed-by=/usr/share/keyrings/nvidia-container-toolkit-keyring.gpg]
https://#g' | sudo tee /etc/apt/sources.list.d/nvidia-container-toolkit.list
```

```
root@phase3-nvidia:~# curl -s -L https://nvidia.github.io/libnvidia-container/stable/deb/nvidia-
container-toolkit.list | sed 's#deb https://#deb [signed-by=/usr/share/keyrings/nvidia-container-
toolkit-keyring.gpg] https://#g' | sudo tee /etc/apt/sources.list.d/nvidia-container-toolkit.list

deb [signed-by=/usr/share/keyrings/nvidia-container-toolkit-keyring.gpg]
https://nvidia.github.io/libnvidia-container/stable/deb/$(ARCH) /

#deb [signed-by=/usr/share/keyrings/nvidia-container-toolkit-keyring.gpg]
https://nvidia.github.io/libnvidia-container/experimental/deb/$(ARCH) /

root@phase3-nvidia:~#
```

Update package list (necessary after adding new repository)

```
sudo apt update
```

```
root@phase3-nvidia:~# sudo apt update
Get:1 https://nvidia.github.io/libnvidia-container/stable/deb/amd64 InRelease [1,477 B]
Hit:2 http://ubuntu.mirror.constant.com noble InRelease
Hit:3 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2404/x86_64 InRelease
Hit:4 http://ubuntu.mirror.constant.com noble-updates InRelease
Hit:5 http://ubuntu.mirror.constant.com noble-backports InRelease
Hit:6 https://download.docker.com/linux/ubuntu noble InRelease
Ign:7 http://linux.mellanox.com/public/repo/mlnx_ofed/24.10-1.1.4.0/ubuntu24.04/amd64 ./ InRelease
```

```
Hit:8 http://security.ubuntu.com/ubuntu noble-security InRelease
Hit:9 http://linux.mellanox.com/public/repo/mlnx_ofed/24.10-1.1.4.0/ubuntu24.04/amd64 ./ Release
Hit:11 https://deb.frrouting.org/frr noble InRelease
Fetched 1,477 B in 11s (138 B/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
All packages are up to date.
root@phase3-nvidia:~#
```

Install Toolkit

Install the NVIDIA Container Toolkit package

```
sudo apt install -y nvidia-container-toolkit
```

```
root@phase3-nvidia:~# sudo apt install -y nvidia-container-toolkit
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
nvidia-container-toolkit is already the newest version (1.18.2-1).
The following packages were automatically installed and are no longer required:
 cuda-cccl-13-1 cuda-command-line-tools-13-1 cuda-compiler-13-1 cuda-crt-13-1 cuda-cudart-13-1
 cuda-cudart-dev-13-1 cuda-culibos-dev-13-1 cuda-cuobjdump-13-1 cuda-cupti-13-1 cuda-cupti-dev-13-
1
 cuda-cuxxfilt-13-1 cuda-documentation-13-1 cuda-driver-dev-13-1 cuda-gdb-13-1 cuda-libraries-13-1
 cuda-libraries-dev-13-1 cuda-nsight-13-1 cuda-nsight-compute-13-1 cuda-nsight-systems-13-1
 cuda-nvcc-13-1 cuda-nvdisasm-13-1 cuda-nvml-dev-13-1 cuda-nvprune-13-1 cuda-nvrtc-13-1
 cuda-nvrtc-dev-13-1 cuda-nvtx-13-1 cuda-opencl-13-1 cuda-profiler-api-13-1 cuda-sandbox-dev-13-1
 cuda-sanitizer-13-1 cuda-tileiras-13-1 cuda-toolkit-13-1 cuda-toolkit-13-1-config-common
 cuda-tools-13-1 cuda-visual-tools-13-1 gds-tools-13-1 libcublas-13-1 libcublas-dev-13-1
 libcufft-13-1 libcufft-dev-13-1 libcufile-13-1 libcufile-dev-13-1 libcuobjclient-13-1
 libcuobjclient-dev-13-1 libcurand-13-1 libcurand-dev-13-1 libcusolver-13-1 libcusolver-dev-13-1
 libcusparse-13-1 libcusparse-dev-13-1 libnpp-13-1 libnpp-dev-13-1 libnvfatbin-13-1
 libnvfatbin-dev-13-1 libnvjitlink-13-1 libnvjitlink-dev-13-1 libnvjpeg-13-1 libnvjpeg-dev-13-1
 libnvptxcompiler-13-1 libnvvm-13-1 nsight-compute-2025.4.1 nsight-systems-2025.5.2
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
root@phase3-nvidia:~#
```

Configure Docker to use the NVIDIA runtime

```
sudo nvidia-ctk runtime configure --runtime=docker
```

```
root@phase3-nvidia:~# sudo nvidia-ctk runtime configure --runtime=docker
INFO[0000] Loading config from /etc/docker/daemon.json
INFO[0000] Wrote updated config to /etc/docker/daemon.json
INFO[0000] It is recommended that docker daemon be restarted.
root@phase3-nvidia:~#
```

Restart Docker to apply changes

```
sudo systemctl restart docker
```

```
root@phase3-nvidia:~# sudo systemctl restart docker
root@phase3-nvidia:~#
```

Run a test container to verify GPU access:

```
docker run --rm --gpus all nvidia/cuda:12.9.0-base-ubuntu24.04 nvidia-smi
```

```
root@phase3-nvidia:~ # docker run --rm --gpus all nvidia/cuda:12.9.0-base-ubuntu24.04 nvidia-smi
Unable to find image 'nvidia/cuda:12.9.0-base-ubuntu24.04' locally
12.9.0-base-ubuntu24.04: Pulling from nvidia/cuda
1bba15468fcc: Pull complete
2b2da1c48640: Pull complete
0622fac788ed: Pull complete
b2276ea4fcfd: Pull complete
2311d82dd6d8: Pull complete
Digest: sha256:48e21b10467354655f5073c05eebdeaac9818c6b40d70f334f7ad2df000463d8
Status: Downloaded newer image for nvidia/cuda:12.9.0-base-ubuntu24.04
Tue Mar 10 10:31:50 2026
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 580.95.05    Driver Version: 580.95.05    CUDA Version: 13.0     |
+-----------------------------------------+------------------------+----------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
+=========================================+========================+======================+
|   0  NVIDIA A100-SXM4-80GB  On | 00000000:1C:00.0 Off | 0 |
| N/A   39C   P0             67W / 500W | 0MiB / 81920MiB | 0%      Default |
|                                         |                        | Disabled |
+-----------------------------------------+------------------------+----------+
|   1  NVIDIA A100-SXM4-80GB  On | 00000000:3E:00.0 Off | 0 |
| N/A   38C   P0             68W / 500W | 0MiB / 81920MiB | 0%      Default |
|                                         |                        | Disabled |
+-----------------------------------------+------------------------+----------+
|   2  NVIDIA A100-SXM4-80GB  On | 00000000:4F:00.0 Off | 0 |
| N/A   37C   P0             60W / 500W | 0MiB / 81920MiB | 0%      Default |
|                                         |                        | Disabled |
+-----------------------------------------+------------------------+----------+
|   3  NVIDIA A100-SXM4-80GB  On | 00000000:60:00.0 Off | 0 |
| N/A   38C   P0             64W / 500W | 0MiB / 81920MiB | 0%      Default |
|                                         |                        | Disabled |
+-----------------------------------------+------------------------+----------+
|   4  NVIDIA A100-SXM4-80GB  On | 00000000:9E:00.0 Off | 0 |
| N/A   40C   P0             68W / 500W | 0MiB / 81920MiB | 0%      Default |
|                                         |                        | Disabled |
+-----------------------------------------+------------------------+----------+
|   5  NVIDIA A100-SXM4-80GB  On | 00000000:BE:00.0 Off | 0 |
| N/A   38C   P0             71W / 500W | 0MiB / 81920MiB | 0%      Default |
|                                         |                        | Disabled |
+-----------------------------------------+------------------------+----------+
```

```
|   6  NVIDIA A100-SXM4-80GB  On | 00000000:CE:00.0 Off | 0 |
| N/A   40C   P0             67W / 500W | 0MiB / 81920MiB | 0%      Default |
|                               |                      | Disabled |
+-----------------------------------------------------------------------------+
|   7  NVIDIA A100-SXM4-80GB  On | 00000000:DE:00.0 Off | 0 |
| N/A   40C   P0             66W / 500W | 0MiB / 81920MiB | 0%      Default |
|                               |                      | Disabled |
+-----------------------------------------------------------------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
| GPU   GI   CI        PID   Type   Process name                 GPU Memory Usage |
|=============================================================================|
| No running processes found                                                  |
+-----------------------------------------------------------------------------+

root@phase3-nvidia:~#
```

Expected Output:

- CUDA container image is downloaded (first run only)
- *nvidia-smi* output shows all available GPUs

Verify GPU Device Selection Syntax:

Test GPU allocation by selecting specific devices

```
docker run --rm --gpus '"device=0,1,2,3"' nvidia/cuda:12.9.0-base-ubuntu24.04 nvidia-smi
```

```
root@phase3-nvidia:~# docker run --rm --gpus "device=0,1,2,3" nvidia/cuda:12.9.0-base-ubuntu24.04
nvidia-smi
Tue Mar 10 10:39:20 2026
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 580.95.05    Driver Version: 580.95.05    CUDA Version: 13.0     |
+-----------------------------------------------------------------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf          Pwr:Usage/Cap | Memory-Usage  | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
+-----------------------------------------------------------------------------+
|   0  NVIDIA A100-SXM4-80GB  On | 00000000:1C:00.0 Off | 0 |
| N/A   39C   P0             67W / 500W | 0MiB / 81920MiB | 0%      Default |
|                               |                      | Disabled |
+-----------------------------------------------------------------------------+
|   1  NVIDIA A100-SXM4-80GB  On | 00000000:3E:00.0 Off | 0 |
| N/A   38C   P0             68W / 500W | 0MiB / 81920MiB | 0%      Default |
|                               |                      | Disabled |
+-----------------------------------------------------------------------------+
|   2  NVIDIA A100-SXM4-80GB  On | 00000000:4F:00.0 Off | 0 |
| N/A   37C   P0             60W / 500W | 0MiB / 81920MiB | 0%      Default |
|                               |                      | Disabled |
+-----------------------------------------------------------------------------+
|   3  NVIDIA A100-SXM4-80GB  On | 00000000:60:00.0 Off | 0 |
| N/A   38C   P0             64W / 500W | 0MiB / 81920MiB | 0%      Default |
|                               |                      | Disabled |
```

```
+-----------------------------------------------------------------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
| GPU   GI   CI        PID   Type   Process name                 GPU Memory Usage |
|=============================================================================|
| No running processes found                                                  |
+-----------------------------------------------------------------------------+

root@phase3-nvidia:~#
```

Expected Output:

- Only GPUs 0-3 are visible inside the container

This confirms that GPU device selection is functioning correctly, which is required for assigning specific GPUs to different services or workloads.

NVIDIA GPU Cloud (NGC) Authentication

NVIDIA GPU Cloud (NGC) is the distribution platform for NVIDIA's containerized AI software, including the Dynamo and associated runtime images. Access to these container images requires authentication via an NGC API key. This key enables automated container image pulls during deployment.

Note: NGC is used exclusively for pulling container images. Model weights are downloaded from Hugging Face using the previously configured HF_TOKEN.

- Obtain NGC API Key from https://ngc.nvidia.com/setup/api-key
- Obtain NGC API Key by navigating to https://ngc.nvidia.com/setup/api-key
    - Sign in with NVIDIA account
    - Navigate to "Setup" → "Generate API Key"
    - Copy the generated key (format varies, typically 40-80 characters)

Configure Environment Variable for authenticating to NGC

Export the API key

```
export NGC_API_KEY=<your_ngc_api_key>
```

```
root@phase3-nvidia:~# export
NGC_API_KEY=ZWxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

root@phase3-nvidia:~#
```

Authenticate Docker to NGC

Login to the NGC container registry:

```
docker login nvcr.io -u '$oauthtoken' -p $NGC_API_KEY
```

```
root@phase3-nvidia:~# docker login nvcr.io -u '$oauthtoken' -p $NGC_API_KEY
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
Login Succeeded
root@phase3-nvidia:~#
```

Expected Output:

- "Login Succeeded" message (ignore password warning)

Persist NGC key for future sessions

```
echo "export NGC_API_KEY=<your_ngc_api_key>" >> ~/.bashrc
```

```
root@phase3-nvidia:~#  echo "export NGC_API_KEY=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" >> ~/.bashrc
root@phase3-nvidia:~#
```

- Replace '<your_ngc_api_key>' with your actual NGC API Key

Reload bashrc

```
source ~/.bashrc
```

```
root@phase3-nvidia:~# source ~/.bashrc
root@phase3-nvidia:~#
```

Validate NGC authentication by pulling a test image

```
docker pull nvcr.io/nvidia/cuda:12.9.0-base-ubuntu24.04
```

```
root@phase3-nvidia:~# docker pull nvcr.io/nvidia/cuda:12.9.0-base-ubuntu24.04
12.9.0-base-ubuntu24.04: Pulling from nvidia/cuda
Digest: sha256:48e21b10467354655f5073c05eebdeaac9818c6b40d70f334f7ad2df000463d8
Status: Downloaded newer image for nvcr.io/nvidia/cuda:12.9.0-base-ubuntu24.04
nvcr.io/nvidia/cuda:12.9.0-base-ubuntu24.04
root@phase3-nvidia:~#
```

Expected Output:

- Image download progress is displayed
- Final message shows: Status: Downloaded newer image

Clone NVIDIA Dynamo repository

The Nvidia Dynamo repository contains deployment scripts, Docker Compose configurations, and orchestration modules required to run distributed inference workloads. To ensure compatibility and stability, the repository should be cloned and checked out to a stable release version.

Clone Repository

Clone the repository from GitHub

```
git clone https://github.com/ai-dynamo/dynamo.git
```

```
root@phase3-nvidia:~# git clone https://github.com/ai-dynamo/dynamo.git
Cloning into 'dynamo'...
remote: Enumerating objects: 159300, done.
remote: Counting objects: 100% (670/670), done.
remote: Compressing objects: 100% (315/315), done.
remote: Total 159300 (delta 535), reused 355 (delta 355), pack-reused 158630 (from 3)
Receiving objects: 100% (159300/159300), 178.28 MiB | 42.74 MiB/s, done.
Resolving deltas: 100% (109137/109137), done.
root@phase3-nvidia:~#
```

Expected Output:

- Repository download progress displayed

Navigate to the repository

```
cd dynamo
```

```
root@phase3-nvidia:~# cd dynamo
root@phase3-nvidia:~/dynamo#
```

Fetch Release Tags

Retrieve all available release tags

```
git fetch --all –tags
```

```
root@phase3-nvidia:~/dynamo# git fetch --all --tags
root@phase3-nvidia:~/dynamo#
```

Identify Latest Stable Version

Find the latest stable (non-RC) release:

```
git tag | grep "^v[0-9]" | grep -v "rc" | sort -V | tail -1
```

```
root@phase3-nvidia:~/dynamo# git tag | grep "^v[0-9]" | grep -v "rc" | sort -V | tail -1
v0.9.1
root@phase3-nvidia:~/dynamo#
```

Expected Output:

- Latest version tag (e.g., v0.9.1)

Alternatively, releases can be reviewed at:
https://github.com/ai-dynamo/dynamo/releases

Switch to the stable release

In this guide we are switching to the latest stable release v0.9.1

```
git checkout v0.9.1
```

```
root@phase3-nvidia:~/dynamo# git checkout v0.9.1
Note: switching to 'v0.9.1'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at ebcbd617a chore: update the attributions files for the changes in 0.9.1 (#6798)
root@phase3-nvidia:~/dynamo#
```

Expected Output:

- HEAD is now at ... (confirming successful checkout)

Verify repository structure

Confirm required deployment files are present:

```
ls -la deploy/
```

```
root@phase3-nvidia:~/dynamo# ls -la deploy/
total 192
drwxr-xr-x  9 root docker  4096 Mar 10 11:43 .
drwxr-xr-x 18 root docker  4096 Mar 10 11:43 ..
-rw-r--r--  1 root docker  3409 Mar 10 10:48 CONTRIBUTING.md
drwxr-xr-x  2 root docker  4096 Mar 10 11:43 discovery
-rw-r--r--  1 root docker  1100 Mar 10 10:48 docker-compose.yml
-rw-r--r--  1 root docker  4850 Mar 10 10:48 docker-observability.yml
drwxr-xr-x  3 root docker  4096 Mar 10 11:43 helm
drwxr-xr-x  5 root docker  4096 Mar 10 11:43 inference-gateway
-rw-r--r--  1 root docker   723 Mar 10 10:48 __init__.py
-rw-r--r--  1 root docker   293 Mar 10 10:48 nats-server.conf
drwxr-xr-x  4 root docker  4096 Mar 10 10:48 observability
drwxr-xr-x  9 root docker  4096 Mar 10 11:43 operator
drwxr-xr-x  3 root docker  4096 Mar 10 10:48 pre-deployment
lrwxrwxrwx  1 root docker    28 Mar 10 11:43 README.md -> ../docs/kubernetes/README.md
-rwxr-xr-x  1 root docker 131679 Mar 10 11:43 sanity_check.py
drwxr-xr-x  3 root docker  4096 Mar 10 11:43 utils
root@phase3-nvidia:~/dynamo#
```

Expected Output:

- Directory listing includes docker-compose.yml and related configuration files

Pull vLLM Container Image (CUDA Version Match)

The vLLM runtime container image must be compatible with the CUDA version supported by the host GPU driver. This compatibility is critical, as a mismatch between the container and host CUDA versions can lead to container startup failures or runtime errors.

To ensure proper operation, the container image tag should be selected based on the CUDA version identified during the infrastructure validation phase.

Check Host CUDA Version

```
nvidia-smi | grep "CUDA Version"
```

```
root@phase3-nvidia:~/dynamo# nvidia-smi | grep "CUDA Version"
| NVIDIA-SMI 580.95.05              Driver Version: 580.95.05      CUDA Version: 13.0 |
```

Expected Output:

- CUDA version displayed (e.g., 12.x or 13.x)

Pull the Matching Container Image

For CUDA 12.x:

```
docker pull nvcr.io/nvidia/ai-dynamo/vllm-runtime:0.9.0
```

For CUDA 13.x:

```
docker pull nvcr.io/nvidia/ai-dynamo/vllm-runtime:0.9.0-cuda13
```

Expected Output:

- Image download progress is displayed
- Final message shows:
    - Status: Downloaded newer image

```
root@phase3-nvidia:~/dynamo# docker pull nvcr.io/nvidia/ai-dynamo/vllm-runtime:0.9.0-cuda13
0.9.0-cuda13: Pulling from nvidia/ai-dynamo/vllm-runtime
c567a87f21d2: Pulling fs layer
0b3b5bd92824: Pulling fs layer
cf57d2112d89: Pulling fs layer
aead5daed6c7: Pulling fs layer
b04c4b3cf9a5: Pulling fs layer
12bbbb535d5b: Pulling fs layer
b15c30b5470f: Pulling fs layer
0e7901fd1abf: Pulling fs layer
7c01d6db7fac: Pulling fs layer
4d56bf5082ea: Pulling fs layer
52ed47cdb2da: Pulling fs layer
9ff7205d44a6: Pulling fs layer
4022c8c09fdb: Pulling fs layer
1d47c4b7bf22: Pulling fs layer
a97438ae3544: Pulling fs layer
e26f1230890e: Pulling fs layer
1ed82f7ab4a3: Pulling fs layer
6ae8aa18fac9: Pulling fs layer
99daae752634: Pulling fs layer
ebbd7d5d157a: Pull complete
e299c59d78a3: Pull complete
30a6a8476000: Pull complete
70f87b6ed43e: Pull complete
818e754c4508: Pull complete
ae3026bfe14e: Pull complete
8e5ca99591ed: Pull complete
Digest: sha256:24d164b9550dabb256954fb3acc3f526e84163a0afbb345d0524020aac57da6e
Status: Downloaded newer image for nvcr.io/nvidia/ai-dynamo/vllm-runtime:0.9.0-cuda13
nvcr.io/nvidia/ai-dynamo/vllm-runtime:0.9.0-cuda13
root@phase3-nvidia:~/dynamo#
```

Verify Image Availability

```
docker images | grep vllm-runtime
```

```
root@phase3-nvidia:~/dynamo# docker images | grep vllm-runtime
WARNING: This output is designed for human readability. For machine-readable output, please use --
format.
nvcr.io/nvidia/ai-dynamo/vllm-runtime:0.9.0-cuda13   24d164b9550d   30.6GB   8.82GB
root@phase3-nvidia:~/dynamo#
```

Expected Output:

- Image is listed with tag, image ID, and size
- **Note:**
  To view all available image tags and versions, refer to the NVIDIA NGC catalog:
  https://catalog.ngc.nvidia.com/orgs/nvidia/teams/ai-dynamo/containers/vllm-runtime

Configure Model Cache Directories

The Dynamo container runs as UID 1000 and requires write access to cache directories for storing Hugging Face model weights, vLLM kernel cache, and FlashInfer compilation artifacts.

Proper directory creation, ownership, and permissions must be configured before container deployment to prevent permission errors during model download and execution.

Create cache directory structure for Dynamo LLM worker

```
mkdir -p ~/dynamo/container/.cache/huggingface
mkdir -p ~/dynamo/container/.cache/vllm
mkdir -p ~/dynamo/container/.cache/flashinfer
```

```
root@phase3-nvidia:~/dynamo# mkdir -p ~/dynamo/container/.cache/huggingface
root@phase3-nvidia:~/dynamo# mkdir -p ~/dynamo/container/.cache/vllm
root@phase3-nvidia:~/dynamo# mkdir -p ~/dynamo/container/.cache/flashinfer
root@phase3-nvidia:~/dynamo#
```

Set Ownership

Assign ownership to UID 1000, which is used by the container runtime

```
sudo chown -R 1000:1000 ~/dynamo/container/.cache
```

```
root@phase3-nvidia:~/dynamo# sudo chown -R 1000:1000 ~/dynamo/container/.cache
root@phase3-nvidia:~/dynamo#
```

Set Permissions

Grant read, write, and execute permissions required for runtime operations

```
sudo chmod -R 775 ~/dynamo/container/.cache
```

```
root@phase3-nvidia:~/dynamo# sudo chmod -R 775 ~/dynamo/container/.cache
root@phase3-nvidia:~/dynamo#
```

Verify Configuration

```
ls -la ~/dynamo/container/.cache/
```

```
root@phase3-nvidia:~/dynamo# ls -la ~/dynamo/container/.cache/
total 20
drwxrwxr-x 5 linuxuser linuxuser 4096 Mar 10 11:53 .
drwxr-xr-x 6 root      docker    4096 Mar 10 11:53 ..
drwxrwxr-x 2 linuxuser linuxuser 4096 Mar 10 11:53 flashinfer
drwxrwxr-x 2 linuxuser linuxuser 4096 Mar 10 11:53 huggingface
drwxrwxr-x 2 linuxuser linuxuser 4096 Mar 10 11:53 vllm
root@phase3-nvidia:~/dynamo#
```

Expected Output:

- Directories: flashinfer, huggingface, vllm
- Ownership set to UID 1000 (may appear as a username depending on system mapping)
- Permissions allow read/write access

Deploy Dynamo Infrastructure Services

NVIDIA Dynamo's distributed architecture depends on foundational infrastructure services to coordinate inference workloads across GPU workers. These services must be deployed prior to launching LLM workers to enable service discovery, worker registration, and inter-worker communication.

The deployment includes the following components:

- **etcd** - A distributed key-value store responsible for worker registration, health monitoring, and configuration management
- **NATS** - A lightweight messaging system used for communication and KV cache event propagation between prefill and decode workers in disaggregated serving mode

Together, these services provide the control-plane capabilities required for Dynamo to manage and orchestrate distributed inference effectively.

Start Infrastructure Services

Ensure the current working directory is the Dynamo repository:

```
cd ~/dynamo
```

Start the infrastructure services using Docker Compose:

```
docker compose -f deploy/docker-compose.yml up -d
```

```
root@phase3-nvidia:~/dynamo# docker compose -f deploy/docker-compose.yml up -d
[+] Up 9/9
 ✓ Image bitnamilegacy/etcd:3.6.1 Pulled                                    4.6s
 ✓ Image nats:2.11.4              Pulled                                    1.2s
 ✓ Network deploy_server          Created                                   0.0s
 ✓ Container deploy-etcd-server-1 Started                                   0.6s
 ✓ Container deploy-nats-server-1 Started                                   0.6s
root@phase3-nvidia:~/dynamo#
```

Expected Output:

- Docker network creation message
- Container pull and startup messages
- deploy-etcd-server-1 and deploy-nats-server-1 show successful startup

Verify Service Status

Check that the infrastructure containers are running:

```
docker compose -f deploy/docker-compose.yml ps
```

```
root@phase3-nvidia:~/dynamo# docker compose -f deploy/docker-compose.yml ps
NAME                   IMAGE                     COMMAND                  SERVICE      CREATED
STATUS              PORTS
deploy-etcd-server-1   bitnamilegacy/etcd:3.6.1  "/opt/bitnami/script..."  etcd-server  About a
minute ago   Up About a minute   0.0.0.0:2379-2380->2379-2380/tcp, [::]:2379-2380->2379-2380/tcp
deploy-nats-server-1   nats:2.11.4               "/nats-server -c /etc/..." nats-server  About a
minute ago   Up About a minute   0.0.0.0:4222->4222/tcp, [::]:4222->4222/tcp, 0.0.0.0:6222-
>6222/tcp, [::]:6222->6222/tcp, 0.0.0.0:8222->8222/tcp, [::]:8222->8222/tcp
root@phase3-nvidia:~/dynamo#
```

Expected Output:

- Output shows the `dynamo-etcd` and `dynamo-nats` containers in **Up** status.

Validate `etcd` Health

Test the `etcd` health endpoint:

```
curl http://localhost:2379/health
```

```
root@phase3-nvidia:~/dynamo# curl http://localhost:2379/health
{"health":"true","reason":""}root@phase3-nvidia:~/dynamo#
```

Expected Output:

- "health":"true"

Validate NATS Connectivity

Test the NATS monitoring endpoint:

```
curl http://localhost:8222/varz
```

```
root@phase3-nvidia:~ /dynamo# curl http://localhost:8222/varz
{
  "server_id": "NATCTYO2L6KC443QUVIWV5T4J4C2TX5MYPYRLBSU6PWC3ZE2P7MXU6DI",
  "server_name": "NATCTYO2L6KC443QUVIWV5T4J4C2TX5MYPYRLBSU6PWC3ZE2P7MXU6DI",
  "version": "2.11.4",
  "proto": 1,
  "git_commit": "4c2fc7f",
  "go": "go1.24.3",
  "host": "0.0.0.0",
  "port": 4222,
  "max_connections": 65536,
  "ping_interval": 120000000000,
  "ping_max": 2,
  "http_host": "0.0.0.0",
  "http_port": 8222,
  "http_base_path": "",
  "https_port": 0,
  "auth_timeout": 2,
  "max_control_line": 4096,
  "max_payload": 15728640,
  "max_pending": 67108864,
  "cluster": {},
  "gateway": {},
  "leaf": {},
  "mqtt": {},
  "websocket": {},
  "jetstream": {
    "config": {
      "max_memory": 1622958855168,
      "max_storage": 552429007872,
      "store_dir": "/tmp/nats/jetstream",
      "sync_interval": 120000000000
    },
    "stats": {
```

```
      "memory": 0,
      "storage": 0,
      "reserved_memory": 0,
      "reserved_storage": 0,
      "accounts": 1,
      "ha_assets": 0,
      "api": {
        "level": 1,
        "total": 0,
        "errors": 0
      }
    }
  },
  "limits": {},
  "tls_timeout": 2,
  "write_deadline": 10000000000,
  "start": "2026-03-10T12:00:40.181612211Z",
  "now": "2026-03-10T12:03:45.495411657Z",
  "uptime": "3m5s",
  "mem": 11010048,
  "cores": 224,
  "gomaxprocs": 224,
  "cpu": 0,
  "connections": 0,
  "total_connections": 0,
  "routes": 0,
  "remotes": 0,
  "leafnodes": 0,
  "in_msgs": 0,
  "out_msgs": 0,
  "in_bytes": 0,
  "out_bytes": 0,
  "slow_consumers": 0,
  "subscriptions": 63,
  "http_req_stats": {
    "/varz": 1
  },
  "config_load_time": "2026-03-10T12:00:40.181612211Z",
  "config_digest": "sha256:fdb567466244e7c5fdc750926c4628cec2e318231adfbef3dde0e4f910d97fff",
  "system_account": "$SYS",
  "slow_consumer_stats": {
    "clients": 0,
    "routes": 0,
    "gateways": 0,
    "leafs": 0
  }
}
root@phase3-nvidia:~/dynamo#
```


Expected Output:

- JSON response containing NATS server information such as version, ports, and connection details

Deploy LLM with Aggregated Serving on NVIDIA Dynamo

Aggregated serving architecture combines both prefill (prompt processing) and decode (token generation) phases within a single worker process. This unified approach simplifies deployment and resource management by allocating GPUs to a single worker that handles the complete inference pipeline.

Architectural Characteristics:

- Unified Worker: Single process manages both prefill and decode operations
- GPU Allocation: GPUs 0-3 assigned to worker for balanced performance
- Use Case: Moderate concurrency scenarios where simplicity and per-request latency are prioritized
- Resource Efficiency: Lower overhead compared to disaggregated mode due to single-process architecture
- Production-Ready: Background daemon with automatic restart policy survives SSH disconnections and system reboots

This mode is recommended for initial deployments, development environments, and production workloads with moderate concurrent request volumes (typically under 50 concurrent users).

Context Length Configuration: The model supports up to 128K tokens. This deployment uses 32K (32768) for balanced performance considering the test hardware being used for this case is 4 X 80GB (total 320 GB). Adjust `--max-model-len` based on VRAM availability:

- 32768 (32K): ~150GB VRAM total, balanced performance
- 65536 (64K): ~180GB VRAM total, longer context
- 131072 (128K): ~250GB+ VRAM total, maximum capability

More details are available here:
https://huggingface.co/nvidia/Llama-3_3-Nemotron-Super-49B-v1_5

Aggregated Serving Deployment Configurations

NVIDIA Dynamo supports multiple deployment configurations for aggregated serving, allowing operators to optimize inference performance based on workload characteristics and GPU topology.

This section demonstrates two deployment configurations for serving the 49B parameter model across four GPUs. Each configuration provides different trade-offs between latency, throughput, and operational simplicity.

Configuration Overview

Configuration 1 - Single Worker (TP=4)

Deploys a single worker process that utilizes all four GPUs using tensor parallelism (TP=4). This configuration prioritizes **lowest per-request latency** and operational simplicity.

Configuration 2 - Dual Workers (TP=2)

Deploys two worker processes, each using two GPUs with TP=2, coordinated by a shared Dynamo frontend. This configuration increases **aggregate throughput** by allowing requests to be processed in parallel across workers.

Both configurations expose the same **OpenAI-compatible API endpoint (port 8000),** enabling seamless integration regardless of the deployment topology.

Configuration 1: Deploy Aggregated Serving with Single Worker (TP=4)

This configuration deploys a single worker that utilizes all four GPUs using **tensor parallelism** (TP=4). Prefill and decode stages execute within the same worker process.

Implementation Details

- **Process Architecture:** Single container running two processes (1 frontend + 1 worker)
- **GPU Allocation:** GPUs 0–3 assigned to one worker with TP=4
- **Memory Utilization:** ~38 GB per GPU
- **Context Length:** Configured for 32K tokens (expandable to 128K depending on available VRAM)
- **Operational Simplicity:** Single worker simplifies monitoring and debugging

This configuration maximizes single-request performance and is best suited for workloads where latency is critical and concurrency requirements are moderate.

Create the Launch Script

Ensure current directory is ~/dynamo

```
pwd
```

```
root@phase3-nvidia: ~/dynamo# pwd
/root/dynamo
root@phase3-nvidia:~/dynamo#
```

Otherwise, navigate to the directory:

```
cd dynamo
```

```
root@phase3-nvidia:~# cd dynamo
root@phase3-nvidia:~/dynamo#
```

Create the launch script:

```
cat << 'EOF' > container/launch_llm_aggregated.sh
#!/bin/bash
set -e
export PYTHONHASHSEED=0
MODEL="nvidia/Llama-3_3-Nemotron-Super-49B-v1_5"
# Pre-download model with retry logic to handle rate limits
echo "Pre-downloading model: $MODEL"
python3 << 'PYEOF'
import time
from huggingface_hub import snapshot_download
model_id = "nvidia/Llama-3_3-Nemotron-Super-49B-v1_5"
max_retries = 6
retry_delay = 60
for attempt in range(max_retries):
    try:
        print(f"Download attempt {attempt + 1}/{max_retries}")
        snapshot_download(
            repo_id=model_id,
            cache_dir="/home/dynamo/.cache/huggingface/hub",
            resume_download=True,
            max_workers=4
        )
        print("Model downloaded successfully!")
        break
    except Exception as e:
        if "429" in str(e) or "Too Many Requests" in str(e):
            if attempt < max_retries - 1:
                wait_time = retry_delay * (2 ** attempt)
                print(f"Rate limit hit. Waiting {wait_time} seconds before retry...")
                time.sleep(wait_time)
            else:
                print("Max retries reached. Exiting.")
```

```
                raise
        else:
            raise
PYEOF
echo "Starting Dynamo Frontend..."
python -m dynamo.frontend &
echo "Starting vLLM LLM Worker in AGGREGATED mode with model: $MODEL"
CUDA_VISIBLE_DEVICES=0,1,2,3 DYN_SYSTEM_PORT=${DYN_SYSTEM_PORT:-8081} python -m dynamo.vllm --
model "$MODEL" --tensor-parallel-size 4 --max-model-len 32768 --trust-remote-code --enforce-eager
--connector none
EOF
```

```
root@phase3-nvidia:~/dynamo# cat << 'EOF' > container/launch_llm_aggregated.sh
#!/bin/bash
set -e
export PYTHONHASHSEED=0
MODEL="nvidia/Llama-3_3-Nemotron-Super-49B-v1_5"
# Pre-download model with retry logic to handle rate limits
echo "Pre-downloading model: $MODEL"
python3 << 'PYEOF'
import time
from huggingface_hub import snapshot_download
model_id = "nvidia/Llama-3_3-Nemotron-Super-49B-v1_5"
max_retries = 6
retry_delay = 60
for attempt in range(max_retries):
    try:
        print(f"Download attempt {attempt + 1}/{max_retries}")
        snapshot_download(
            repo_id=model_id,
            cache_dir="/home/dynamo/.cache/huggingface/hub",
            resume_download=True,
            max_workers=4
        )
        print("Model downloaded successfully!")
        break
    except Exception as e:
EOFel”  --tensor-parallel-size 4   --max-model-len 32768   --trust-remote-code  --enforce-eager
  --connector
root@phase3-nvidia:~/dynamo#
```

Fix line endings for cross-platform compatibility:

```
sed -i 's/\r$//' container/launch_llm_aggregated.sh
```

```
root@phase3-nvidia:~/dynamo# sed -i 's/\r$//' container/launch_llm_aggregated.sh

root@phase3-nvidia:~/dynamo#
```

Make the launch script executable:

```
chmod +x container/launch_llm_aggregated.sh
```

```
root@phase3-nvidia:~/dynamo# chmod +x container/launch_llm_aggregated.sh
root@phase3-nvidia:~/dynamo#
```

Deploy LLM Worker as a Background Service

The LLM worker is deployed as a background daemon with automatic restart policy. This configuration ensures the service survives SSH disconnections, container failures, and system reboots. Host networking is used for Dynamo service discovery, and shared memory (IPC) enables efficient multi-GPU communication.

Run the LLM Worker Container

```
docker run -d --name dynamo-llm --restart always --gpus '"device=0,1,2,3"' --network host --
ipc=host --ulimit memlock=-1 --ulimit stack=67108864 -e HF_TOKEN=$HF_TOKEN -e PYTHONHASHSEED=0 -v
$(pwd)/container:/workspace -v $(pwd)/container/.cache:/home/dynamo/.cache -w /workspace -u
1000:1000 $DYNAMO_IMAGE bash -c "/workspace/launch_llm_aggregated.sh"
```

```
root@phase3-nvidia:~/dynamo# docker run -d --name dynamo-llm --restart always --gpus
"device=0,1,2,3" ' --network host --ipc=host --ulimit memlock=-1 --ulimit stack=67108864
-e HF_TOKEN=$HF_TOKEN -e PYTHONHASHSEED=0 -v $(pwd)/container:/workspace -v
$(pwd)/container/.cache:/home/dynamo/.cache -w /workspace -u 1000:1000 $DYNAMO_IMAGE bash -c
"/workspace/launch_llm_aggregated.sh"
cdb26a2e0ef0f50d59c3cb5b5e8d7b1df9f237a352bd7f4db6230f6af8a1855e
root@phase3-nvidia:~/dynamo#
```

Container Runtime Parameters

- **-d:** Run in background (daemon mode)
- **--name dynamo-llm:** Named container for easy management
- **--restart always:** Automatically restart on failure or system reboot
- **--gpus '"device=0,1,2,3"':** Allocate GPUs 0-3 for LLM worker
- **--network host:** Use host networking (required for Dynamo discovery)
- **--ipc=host:** Shared memory for multi-GPU communication
- **--ulimit memlock=-1:** Unlimited locked memory for GPU operations
- **--ulimit stack=67108864:** Increased stack size for deep models
- **-e HF_TOKEN:** Hugging Face authentication token
- **-e PYTHONHASHSEED=0:** Deterministic Python hashing
- **-v $(pwd)/container:/workspace:** Mount workspace directory

- **-v $(pwd)/container/.cache:/home/dynamo/.cache:** Persist model cache
- **-w /workspace:** Set working directory
- **-u 1000:1000:** Run as UID 1000 (matches container user)

Expected Output:

- Container ID (64-character hex string)

Monitor Worker Initialization

Worker initialization involves model pre-downloading with retry logic (10-20 minutes for 49B model), Dynamo frontend startup, and vLLM engine initialization. Monitoring the container logs provides visibility into each phase and helps identify issues during deployment.

View Container Logs in Real Time

```
docker logs -f dynamo-llm
```

```
root@phase3-nvidia:~/dynamo# docker logs -f dynamo-llm

==========
== CUDA ==
==========

CUDA Version 13.0.2

Container image Copyright (c) 2016-2023, NVIDIA CORPORATION & AFFILIATES. All rights reserved.

This container image and its contents are governed by the NVIDIA Deep Learning Container License.
By pulling and using the container, you accept the terms and conditions of this license:
https://developer.nvidia.com/ngc/nvidia-deep-learning-container-license

A copy of this license is made available in this container at /NGC-DL-CONTAINER-LICENSE for your
convenience.

Pre-downloading model: nvidia/Llama-3_3-Nemotron-Super-49B-v1_5
/opt/dynamo/venv/lib/python3.12/site-packages/huggingface_hub/file_download.py:949: FutureWarning:
`resume_download` is deprecated and will be removed in version 1.0.0. Downloads always resume when
possible. If you want to force a new download, use `force_download=True`.
warnings.warn(
Download attempt 1/6
Fetching 47 files: 100% |████████████████████████████████████████| 47/47 [01:02<00:00, 1.33s/it]
Model downloaded successfully!
```

```
Starting Dynamo Frontend...
Starting VLLM LLM Worker in AGGREGATED mode with model: nvidia/Llama-3_3-Nemotron-Super-49B-v1_5
2026-03-10T12:19:34.870855Z  INFO dynamo_runtime::distributed: Initializing KV store discovery
backend
2026-03-10T12:19:34.870981Z  INFO dynamo_runtime::pipeline::network::manager: Initializing
NetworkManager with TCP request plane mode=tcp host=192.0.2.101 port=OS-assigned
```

```
2026-03-10T12:19:34.885791Z  INFO dynamo_llm::http::service::service_v2: Starting HTTP(S) service
protocol="HTTP" address="0.0.0.0:8000"
[2026-03-10 12:19:59] INFO __init__.py:53: dynamo.nixl_connect: Utilizing CuPy to enable GPU
acceleration.
[2026-03-10 12:20:01] INFO encode_worker_handler.py:39: Using cupy for array operations (GPU
mode).
2026-03-10T12:20:01.300187Z  INFO args.create_kv_transfer_config: Using vLLM defaults for
kv_transfer_config
2026-03-10T12:20:01.300246Z  INFO args.create_kv_events_config: Using env-var
DYN_VLLM_KV_EVENT_PORT=20080 to create kv_events_config
2026-03-10T12:20:01.300298Z  INFO args.overwrite_args: Using kv_events_config for publishing vLLM
kv events over zmq: KVEventsConfig(enable_kv_cache_events=True, publisher='zmq',
endpoint='tcp://*:20080', replay_endpoint=None, buffer_steps=10000, hwm=100000,
max_queue_size=100000, topic=' ') (use_kv_events'=True)
2026-03-10T12:20:01.344955Z  INFO dynamo_runtime::distributed: Initializing KV store discovery
backend
2026-03-10T12:20:01.345110Z  INFO dynamo_runtime::pipeline::network::manager: Initializing
NetworkManager with TCP request plane mode=tcp host=192.0.2.101 port=OS-assigned
2026-03-10T12:20:01.345956Z  INFO dynamo_runtime::system_status_server:
[spawn_system_status_server] binding to: 0.0.0.0:8081
2026-03-10T12:20:01.345997Z  INFO dynamo_runtime::system_status_server:
[spawn_system_status_server] system status server bound to: 0.0.0.0:8081
```

```
2026-03-10T12:21:12.049456Z  INFO cuda.get_attn_backend_cls: Using FLASH_ATTN attention backend
out of potential backends: ('FLASH_ATTN', 'FLASHINFER', 'TRITON_ATTN', 'FLEX_ATTENTION')
Loading safetensors checkpoint shards:   0% Completed | 0/21 [00:00<?, ?it/s]
Loading safetensors checkpoint shards:   5% Completed | 1/21 [00:00<00:09, 2.08it/s]
Loading safetensors checkpoint shards:  10% Completed | 2/21 [00:00<00:08, 2.13it/s]
Loading safetensors checkpoint shards:  14% Completed | 3/21 [00:01<00:08, 2.15it/s]
Loading safetensors checkpoint shards:  19% Completed | 4/21 [00:01<00:07, 2.29it/s]
Loading safetensors checkpoint shards:  24% Completed | 5/21 [00:02<00:06, 2.39it/s]
Loading safetensors checkpoint shards:  29% Completed | 6/21 [00:02<00:06, 2.41it/s]
Loading safetensors checkpoint shards:  33% Completed | 7/21 [00:03<00:06, 2.29it/s]
Loading safetensors checkpoint shards:  38% Completed | 8/21 [00:03<00:05, 2.21it/s]
Loading safetensors checkpoint shards:  43% Completed | 9/21 [00:04<00:05, 2.19it/s]
Loading safetensors checkpoint shards:  48% Completed | 10/21 [00:04<00:05, 2.16it/s]
Loading safetensors checkpoint shards:  52% Completed | 11/21 [00:05<00:04, 2.24it/s]
Loading safetensors checkpoint shards:  57% Completed | 12/21 [00:05<00:04, 2.22it/s]
Loading safetensors checkpoint shards:  62% Completed | 13/21 [00:05<00:03, 2.60it/s]
Loading safetensors checkpoint shards:  67% Completed | 14/21 [00:06<00:02, 2.66it/s]
Loading safetensors checkpoint shards:  71% Completed | 15/21 [00:06<00:02, 2.45it/s]
Loading safetensors checkpoint shards:  76% Completed | 16/21 [00:07<00:02, 2.39it/s]
Loading safetensors checkpoint shards:  81% Completed | 17/21 [00:07<00:01, 2.31it/s]
Loading safetensors checkpoint shards:  86% Completed | 18/21 [00:08<00:01, 2.25it/s]
Loading safetensors checkpoint shards:  90% Completed | 19/21 [00:08<00:00, 2.22it/s]
Loading safetensors checkpoint shards:  95% Completed | 20/21 [00:08<00:00, 2.21it/s]
Loading safetensors checkpoint shards: 100% Completed | 21/21 [00:09<00:00, 2.29it/s]
```



```
{ rfilename: "model-00017-of-00021.safetensors" }, Siblings { filename: "model-00018-of-
00021.safetensors" }, Siblings { rfilename: "model-00019-of-00021.safetensors" }, Siblings {
rfilename: "model-00020-of-00021.safetensors" }, Siblings { rfilename: "model-00021-of-
00021.safetensors" }, Siblings { rfilename: "model.safetensors.index.json" }, Siblings {
rfilename: "modeling_decilm.py" }, Siblings { rfilename: "special_tokens_map.json" }, Siblings {
rfilename: "tokenizer.json" }, Siblings { rfilename: "tokenizer_config.json" }, Siblings {
rfilename: "transformers_4_44_2__activations.py" }, Siblings { rfilename:
"transformers_4_44_2__cache_utils.py" }, Siblings { rfilename:
"transformers_4_44_2__configuration_llama.py" }, Siblings { rfilename:
"transformers_4_44_2__modeling_attn_mask_utils.py" }, Siblings { rfilename:
"transformers_4_44_2__modeling_flash_attention_utils_backward_compat.py" }, Siblings { rfilename:
"transformers_4_44_2__modeling_outputs.py" }, Siblings { rfilename:
"transformers_4_44_2__modeling_rope_utils.py" }, Siblings { rfilename:
"transformers_4_44_2__pytorch_utils.py" }, Siblings { rfilename: "variable_cache.py" }, sha:
"420ba7d282112bf116b8b103ab700d92619daf98"
2026-03-10T12:21:50.324034Z  INFO modelexpress_common::providers::huggingface: Downloaded model
files for nvidia/Llama-3_3-Nemotron-Super-49B-v1_5
2026-03-10T12:21:50.324141Z  INFO dynamo_llm::hub: ModelExpress download completed successfully
for model nvidia/Llama-3_3-Nemotron-Super-49B-v1_5
2026-03-10T12:21:50.821539Z  INFO dynamo_llm::http::service::service_v2: chat endpoints enabled
2026-03-10T12:21:50.821575Z  INFO dynamo_llm::http::service::service_v2: completion endpoints
enabled
2026-03-10T12:21:50.847551Z  INFO dynamo_llm::discovery::watcher: Chat completions is ready
2026-03-10T12:21:50.869813Z  INFO dynamo_llm::discovery::watcher: Completions is ready
2026-03-10T12:21:50.869829Z  INFO dynamo_llm::discovery::watcher: added model
model_name="nvidia/Llama-3_3-Nemotron-Super-49B-v1_5" namespace="dynamo"
2026-03-10T12:39:03.393238Z  INFO http_request: dynamo_runtime::system_status_server: [fallback
handler] called method=GET uri=/ version=HTTP/1.1
```

Expected Output:

- Log sequence showing download attempts, model download success, frontend startup, and engine initialization

Key Log Indicators

- "Download attempt X/6": Model download progress with retry logic
- "Model downloaded successfully!": Model weights fully downloaded
- "Starting Dynamo Frontend...": Frontend service initialization
- "Starting vLLM LLM Worker in AGGREGATED mode...": Worker startup
- "Chat completions is ready": Chat endpoint ready to serve requests
- "Completions is ready": Completions endpoint ready to serve requests
- "added model": Model registered and available for inference

Press Ctrl+C to to exit log stream. Container keeps running.

Filter Important Logs Only

Container logs can be verbose with hundreds of lines of initialization details. This filtered command extracts only the critical status messages, making it easy to verify successful deployment without scrolling through extensive debug output.

```
docker logs dynamo-llm 2>&1 | grep -iE "Model downloaded successfully|Starting Dynamo
Frontend|Starting vLLM LLM Worker|Chat completions is ready|Completions is ready|added model"
```

```
root@phase3-nvidia:~/dynamo# docker logs dynamo-llm 2>&1 | grep -iE "Model downloaded
successfully|Starting Dynamo Frontend|Starting VLLM LLM Worker|Chat completions is
ready|Completions is ready|added model"
Model downloaded successfully!
Starting Dynamo Frontend...
Starting VLLM LLM Worker in AGGREGATED mode with model: nvidia/Llama-3_3-Nemotron-Super-49B-v1_5
2026-03-13T10:19:52.915707Z  INFO dynamo_llm::discovery::watcher: Chat completions is ready
2026-03-13T10:19:52.938138Z  INFO dynamo_llm::discovery::watcher: Completions is ready
2026-03-13T10:19:52.938153Z  INFO dynamo_llm::discovery::watcher: added model
model_name="nvidia/Llama-3_3-Nemotron-Super-49B-v1_5" namespace="dynamo"
root@phase3-nvidia:~/dynamo#
```

Expected Filtered Output:

- Model downloaded successfully!
- Starting Dynamo Frontend...
- Starting vLLM LLM Worker in AGGREGATED mode with model: nvidia/Llama-3_3-Nemotron-Super-49B-v1_5
- Chat completions is ready
- Completions is ready
- added model model_name="nvidia/Llama-3_3-Nemotron-Super-49B-v1_5" namespace="dynamo"

Verify Single Worker Deployment

Verification confirms that the container is running, GPUs are allocated correctly, and the API endpoint is responding to inference requests.

Check Container Status

```
docker ps | grep dynamo-llm
```

```
root@phase3-nvidia:~/dynamo# docker ps | grep dynamo-llm
cdb26a2e0ef0   nvcr.io/nvidia/ai-dynamo/vllm-runtime:0.9.0-cuda13   "/opt/nvidia/nvidia_…"   41
minutes ago   Up 41 minutes
                                                                    dynamo-llm
root@phase3-nvidia:~/dynamo#
```

Expected Output:

- Container shows Up status

Verify GPU Utilization

```
nvidia-smi
```

```
root@phase3-nvidia:~/dynamo# nvidia-smi
Tue Mar 10 12:59:45 2026
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 580.95.05              Driver Version: 580.95.05      CUDA Version: 13.0     |
+-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA A100-SXM4-80GB          On  |   00000000:1C:00.0 Off |                  Off |
| N/A   39C    P0             74W /  500W |   74371MiB /  81920MiB |      0%      Default |
|                                         |                        |             Disabled |
+-----------------------------------------+------------------------+----------------------+
|   1  NVIDIA A100-SXM4-80GB          On  |   00000000:3E:00.0 Off |                  Off |
| N/A   39C    P0             75W /  500W |   74371MiB /  81920MiB |      0%      Default |
|                                         |                        |             Disabled |
+-----------------------------------------+------------------------+----------------------+
|   2  NVIDIA A100-SXM4-80GB          On  |   00000000:4F:00.0 Off |                  Off |
| N/A   38C    P0             66W /  500W |   74371MiB /  81920MiB |      0%      Default |
|                                         |                        |             Disabled |
+-----------------------------------------+------------------------+----------------------+
|   3  NVIDIA A100-SXM4-80GB          On  |   00000000:60:00.0 Off |                    0 |
| N/A   39C    P0             70W /  500W |   74371MiB /  81920MiB |      0%      Default |
|                                         |                        |             Disabled |
+-----------------------------------------+------------------------+----------------------+
+-----------------------------------------+------------------------+----------------------+
|   4  NVIDIA A100-SXM4-80GB          On  |   00000000:9E:00.0 Off |                    0 |
| N/A   39C    P0             68W /  500W |       0MiB /  81920MiB |      0%      Default |
|                                         |                        |             Disabled |
+-----------------------------------------+------------------------+----------------------+
|   5  NVIDIA A100-SXM4-80GB          On  |   00000000:BE:00.0 Off |                    0 |
| N/A   38C    P0             70W /  500W |       0MiB /  81920MiB |      0%      Default |
|                                         |                        |             Disabled |
+-----------------------------------------+------------------------+----------------------+
|   6  NVIDIA A100-SXM4-80GB          On  |   00000000:CE:00.0 Off |                    0 |
| N/A   40C    P0             67W /  500W |       0MiB /  81920MiB |      0%      Default |
|                                         |                        |             Disabled |
+-----------------------------------------+------------------------+----------------------+
|   7  NVIDIA A100-SXM4-80GB          On  |   00000000:DE:00.0 Off |                    0 |
| N/A   40C    P0             66W /  500W |       0MiB /  81920MiB |      0%      Default |
|                                         |                        |             Disabled |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI              PID   Type   Process name                        GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|    0   N/A  N/A         1499405      C   VLLM::Worker_TP0                      74362MiB |
|    1   N/A  N/A         1499412      C   VLLM::Worker_TP1                      74362MiB |
|    2   N/A  N/A         1499423      C   VLLM::Worker_TP2                      74362MiB |
|    3   N/A  N/A         1499439      C   VLLM::Worker_TP3                      74362MiB |
+-----------------------------------------------------------------------------------------+
root@phase3-nvidia:~/dynamo#
```

Expected Output:

- GPUs 0–3 show active processes with expected memory usage of approximately 75GB each

Test LLM endpoint

```
MODEL_NAME="nvidia/Llama-3_3-Nemotron-Super-49B-v1_5"
```

```
curl -X POST http://localhost:8000/v1/chat/completions -H "Content-Type: application/json" -d
"{\"model\": \"$MODEL_NAME\",\"messages\": [{\"role\": \"user\", \"content\": \"Say
hello\"}],\"max_tokens\": 50}" | jq
```

```
root@phase3-nvidia:~/dynamo# MODEL_NAME="nvidia/Llama-3_3-Nemotron-Super-49B-v1_5"
root@phase3-nvidia:~/dynamo# curl -X POST http://localhost:8000/v1/chat/completions -H "Content-
Type: application/json" -d "{\"model\": \"$MODEL_NAME\", \"messages\": [{\"role\": \"user\",
\"content\": \"Say hello\"}], \"max_tokens\": 50}" | jq
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   686  100   561  100   125    417     93  0:00:01  0:00:01 --:--:--   510
{  "id": "chatcmpl-2b781f21-1109-4d50-afb7-ba40f9a67906",
  "choices": [
    {
      "index": 0,
      "message": {
        "content": "<think>\nOkay, the user said \"Say hello\". Let me think about how to
respond.\n\n
First, I need to acknowledge their message. Since they asked me to say hello, I should start with
a friendly greeting. Maybe \"Hello!\" or",
        "role": "assistant",
        "reasoning_content": null
      },
      "finish_reason": "length"
    }
  ],
  "created": 1773147505,
  "model": "nvidia/Llama-3_3-Nemotron-Super-49B-v1_5",
  "object": "chat.completion",
  "usage": {
    "prompt_tokens": 17,
    "completion_tokens": 50,
    "total_tokens": 67
  }
}
root@phase3-nvidia:~/dynamo#
```

Expected Output:

- JSON response with generated output in choices[0].message.content

Service Endpoint Information

The deployed LLM service exposes an OpenAI-compatible API endpoint for chat completions. This endpoint can be integrated into applications using standard OpenAI client libraries or HTTP clients. The service runs on host networking and is accessible on port 8000.

Endpoint Details

- URL: http://< NVIDIA_GPU_SERVER_IP>:8000/v1/chat/completions
- Model: nvidia/Llama-3_3-Nemotron-Super-49B-v1_5
- Context Length: 32K tokens (configurable up to 128K)
- API Standard: OpenAI-compatible
- Authentication: None (internal network deployment)

Example Python Integration

```
import httpx
response = httpx.post(
    "http://< NVIDIA_GPU_SERVER_IP>:8000/v1/chat/completions",
    json={
        "model": "nvidia/Llama-3_3-Nemotron-Super-49B-v1_5",
        "messages": [
            {"role": "system", "content": "You are a helpful assistant."},
            {"role": "user", "content": "Explain RAG in simple terms."}
        ],
        "max_tokens": 500,
        "temperature": 0.7
    }
)
result = response.json()
answer = result["choices"][0]["message"]["content"]
print (answer)
```

Example cURL Request

```
curl -X POST http://< NVIDIA_GPU_SERVER_IP>:8000/v1/chat/completions \
  -H "Content-Type: application/json" \
  -d '{
    "model": "nvidia/Llama-3_3-Nemotron-Super-49B-v1_5",
    "messages": [{"role": "user", "content": "What is AI?"}],
    "max_tokens": 200,
    "temperature": 0.7
  }'
```

Replace `< NVIDIA_GPU_SERVER_IP>` with the actual internal IP address of the GPU server.

```
root@phase3-nvidia:~# curl -X POST http://192.0.2.101:8000/v1/chat/completions \
-H "Content-Type: application/json" \
-d '{
  "model": "nvidia/Llama-3_3-Nemotron-Super-49B-v1_5",
  "messages": [{"role": "user", "content": "What is AI?"}],
  "max_tokens": 200,
  "temperature": 0.7
```

```
}'
{"id":"chatcmpl-ed33d5e7-bce8-4f61-a229 17c42ae4cd33","choices": [{"index":0,"message":{"content"
:"<think>\nOkay, so I need to explain what AI is. Let me start by breaking down the term. AI
stands for Artificial Intelligence. But what does that really mean? I know it's a broad field, but
I should try to define it clearly.\n\n First, I should mention that AI refers to machines or
software that can perform tasks that typically require human intelligence. But wait, what are
those tasks? Maybe things like learning, reasoning, problem-solving, understanding language,
perception, and decision-making. Yeah, that makes sense. So AI systems are designed to mimic these
human cognitive functions.\n\nI should probably differentiate between different types of AI.
There's the concept of narrow AI versus general AI. Narrow AI is what we see today, like voice
assistants, recommendation systems on Netflix or Amazon, self-driving cars. These are all
specialized in specific tasks. Then there's general AI, which is hypothetical and would be able to
handle any intellectual task a human can. But we're not there
yet.","role":"assistant","reasoning_content":null},
"finish_reason":"length"}],"created":1773393742,"model":"nvidia/Llama-3_3-Nemotron-Super-49B-
v1_5","object":"chat.completion","usage":{"prompt_tokens":19,"completion_tokens":200,"total_tokens
":219}}
root@phase3-nvidia:~#
```


Configuration 2: Deploy Aggregated Serving with Dual Workers (TP=2)

This configuration demonstrates Dynamo's **multi-worker serving architecture** where **two worker processes are deployed,** each utilizing **two GPUs with tensor parallelism (TP=2).** A single Dynamo frontend coordinates the workers and distributes incoming requests across the available worker pool.

This setup is designed to increase overall throughput and concurrency, making it suitable for high-traffic environments where handling multiple requests in parallel is more important than optimizing single-request latency, all while maintaining the simplicity of a single API endpoint. The automatic load balancing eliminates the need for client-side request distribution logic.

Implementation Details

Defines how compute resources and processes are organized for dual-worker deployment.

- **Process Architecture:** Single container runs three processes (1 frontend + 2 workers)
- **GPU Allocation:** Worker 1 → GPUs 0-1 (TP=2) | Worker 2 → GPUs 2-3 (TP=2)
- **Port Configuration:** Each worker uses unique DYN_SYSTEM_PORT and ZMQ communication ports
- **Load Distribution:** Dynamo frontend automatically balances requests across registered workers
- **Worker Initialization:** Workers are started sequentially to ensure proper registration with the frontend

Create Dual Worker Launch Script

A single launch script starts the Dynamo frontend and then launches two separate vLLM worker processes. Each worker runs with TP=2 on different GPU pairs and registers to the same frontend for automatic load balancing.

Ensure current directory is ~/dynamo

```
Pwd
```

```
root@phase3-nvidia:~/dynamo# pwd
/root/dynamo
root@phase3-nvidia:~/dynamo#
```

Create the dual worker launch script

```
cat << 'EOF' > container/launch_llm_dual_tp2.sh
#!/bin/bash
set -e
export PYTHONHASHSEED=0
MODEL="nvidia/Llama-3_3-Nemotron-Super-49B-v1_5"

# Pre-download model with retry logic to handle rate limits
echo "Pre-downloading model: $MODEL"
python3 << 'PYEOF'
import time
from huggingface_hub import snapshot_download

model_id = "nvidia/Llama-3_3-Nemotron-Super-49B-v1_5"
max_retries = 6
retry_delay = 60

for attempt in range(max_retries):
    try:
        print(f"Download attempt {attempt + 1}/{max_retries}")
        snapshot_download(
            repo_id=model_id,
            cache_dir="/home/dynamo/.cache/huggingface/hub",
            resume_download=True,
            max_workers=4
        )
        print("Model downloaded successfully!")
        break
    except Exception as e:
        if "429" in str(e) or "Too Many Requests" in str(e):
            if attempt < max_retries - 1:
                wait_time = retry_delay * (2 ** attempt)
                print(f"Rate limit hit. Waiting {wait_time} seconds before retry...")
                time.sleep(wait_time)
            else:
                print("Max retries reached. Exiting.")
                raise
```

```
        else:
            raise
PYEOF

echo "Starting Dynamo Frontend..."
python -m dynamo.frontend &
FRONTEND_PID=$!

# Wait for frontend to initialize
sleep 10

echo "Starting vLLM Worker 1 (GPUs 0-1, TP=2) with model: $MODEL"
CUDA_VISIBLE_DEVICES=0,1 DYN_SYSTEM_PORT=8081 python -m dynamo.vllm \
  --model "$MODEL" \
  --tensor-parallel-size 2 \
  --max-model-len 32768 \
  --trust-remote-code \
  --enforce-eager \
  --connector none &
WORKER1_PID=$!

# Wait for Worker 1 to initialize before starting Worker 2
sleep 30

echo "Starting vLLM Worker 2 (GPUs 2-3, TP=2) with model: $MODEL"
CUDA_VISIBLE_DEVICES=2,3 DYN_SYSTEM_PORT=8082 python -m dynamo.vllm \
  --model "$MODEL" \
  --tensor-parallel-size 2 \
  --max-model-len 32768 \
  --trust-remote-code \
  --enforce-eager \
  --connector none &
WORKER2_PID=$!

echo "All services started:"
echo "Frontend PID: $FRONTEND_PID (port 8000)"
echo "Worker 1 PID: $WORKER1_PID (GPUs 0-1)"
echo "Worker 2 PID: $WORKER2_PID (GPUs 2-3)"

# Wait for all processes
wait
EOF
```

```
root@phase3-nvidia:~/dynamo# cat << 'EOF' > container/launch_llm_dual_tp2.sh
#!/bin/bash
set -e

export PYTHONHASHSEED=0
MODEL="nvidia/Llama-3_3-Nemotron-Super-49B-v1_5"

# Pre-download model with retry logic to handle rate limits
echo "Pre-downloading model: $MODEL"
python3 << 'PYEOF'
import time
```

```
from huggingface_hub import snapshot_download

model_id = "nvidia/Llama-3_3-Nemotron-Super-49B-v1_5"
max_retries = 6
retry_delay = 60

for attempt in range(max_retries):
    try:
        print(f"Download attempt {attempt + 1}/{max_retries}")
        snapshot_download(
            repo_id=model_id,
            cache_dir="/home/dynamo/.cache/huggingface/hub",
            resume_download=True,

EOFtit for all processesKER2_PID (GPUs 2-3)"DYN_VLLM_KV_EVENT_PORT=20082 python -m dynamo.vllm
root@phase3-nvidia:~/dynamo#
```

Expected Result

- Script file created

Fix line endings in the script file and make it executable

```
sed -i 's/\r$//' container/launch_llm_dual_tp2.sh
```

```
root@phase3-nvidia:~/dynamo# sed -i 's/\r$//' container/launch_llm_dual_tp2.sh
root@phase3-nvidia:~/dynamo#
```

```
chmod +x container/launch_llm_dual_tp2.sh
```

```
root@phase3-nvidia:~/dynamo# chmod +x container/launch_llm_dual_tp2.sh
root@phase3-nvidia:~/dynamo#
```

Expected Output:

- No output (silent success)

Clean Up Previous Deployments

Ensure that any existing LLM containers are stopped and removed before deploying the dual-worker configuration. This prevents GPU resource conflicts and ensures a clean deployment state.

Stop and Remove Existing Containers

```
docker stop dynamo-llm dynamo-llm-disaggregated dynamo-llm-instance1 dynamo-llm-instance2
2>/dev/null || true
docker rm dynamo-llm dynamo-llm-disaggregated dynamo-llm-instance1 dynamo-llm-instance2
2>/dev/null || true
```

```
root@phase3-nvidia:~/dynamo# docker stop dynamo-llm dynamo-llm-disaggregated dynamo-llm-instance1
dynamo-llm-instance2 2>/dev/null || true

root@phase3-nvidia:~/dynamo# docker rm dynamo-llm dynamo-llm-disaggregated dynamo-llm-instance1
dynamo-llm-instance2 2>/dev/null || true

root@phase3-nvidia:~/dynamo#
```

Expected Output:

- Container names displayed (if running)
- No output if no containers exist

Deploy Dual Workers as a Background Service

The dual-worker deployment runs as a single background container that manages the Dynamo frontend and both worker processes. This setup ensures all components remain active across SSH disconnections, container failures, and system reboots, while maintaining proper coordination between the frontend and workers.

Run the Dual Worker Container

```
docker run -d --name dynamo-llm-dual-tp2 --restart always --gpus '"device=0,1,2,3"' --network host
--ipc=host --ulimit memlock=-1 --ulimit stack=67108864 -e HF_TOKEN=$HF_TOKEN -e PYTHONHASHSEED=0 -
v $(pwd)/container:/workspace -v $(pwd)/container/.cache:/home/dynamo/.cache -w /workspace -u
1000:1000 $DYNAMO_IMAGE bash -c "/workspace/launch_llm_dual_tp2.sh"
```

```
root@phase3-nvidia:~/dynamo# docker run -d --name dynamo-llm-dual-tp2 --restart always --gpus
"device=0,1,2,3" --network host --ipc=host --ulimit memlock=-1 --ulimit stack=67108864 -e
HF_TOKEN=$HF_TOKEN -e PYTHONHASHSEED=0 -v $(pwd)/container:/workspace -v
$(pwd)/container/.cache:/home/dynamo/.cache -w /workspace -u 1000:1000 $DYNAMO_IMAGE bash -c
"/workspace/launch_llm_dual_tp2.sh"

bf5f89fc158a6df1a6d3b12c8f30d4e95b50806c845f20524934ec74e8787986

root@phase3-nvidia:~/dynamo#
```

Container Runtime Parameters

- **-d** - Runs the container in detached (background) mode
- **--name dynamo-llm-dual-tp2** - Assigns a fixed container name for management
- **--restart always** - Ensures automatic restart on failure or reboot
- **--gpus '"device=0,1,2,3"'** - Allocates all GPUs for both workers
- **--network host** - Enables host networking for Dynamo service discovery
- **--ipc=host** - Shares host IPC namespace for efficient multi-GPU communication
- **--ulimit memlock=-1** - Removes memory lock limits for GPU workloads
- **--ulimit stack=67108864** - Increases stack size for deep model execution

- -e HF_TOKEN=$HF_TOKEN - Provides Hugging Face authentication
- -e PYTHONHASHSEED=0 - Ensures deterministic Python behavior
- -v $(pwd)/container:/workspace - Mounts working directory
- -v $(pwd)/container/.cache:/home/dynamo/.cache - Persists model cache
- -w /workspace - Sets working directory inside container
- -u 1000:1000 - Runs container as UID/GID 1000

Expected Output:

- Container ID returned (64-character hex string)

Monitor Dual Worker Initialization

Initialization includes model pre-download, Dynamo frontend startup, and sequential registration of both workers. Each worker registers with the shared frontend, enabling automatic load balancing across the worker pool.

Monitor container logs in real-time

```
docker logs -f dynamo-llm-dual-tp2
```

```
root@phase3-nvidia:~/dynamo# docker logs -f dynamo-llm-dual-tp2
==========
== CUDA ==
==========

CUDA Version 13.0.2

Container image Copyright (c) 2016-2023, NVIDIA CORPORATION & AFFILIATES. All rights reserved.

This container image and its contents are governed by the NVIDIA Deep Learning Container License.
By pulling and using the container, you accept the terms and conditions of this license:
https://developer.nvidia.com/ngc/nvidia-deep-learning-container-license

A copy of this license is made available in this container at /NGC-DL-CONTAINER-LICENSE for your
convenience.

Pre-downloading model: nvidia/Llama-3_3-Nemotron-Super-49B-v1_5
/opt/dynamo/venv/lib/python3.12/site-packages/huggingface_hub/file_download.py:949: FutureWarning:
`resume_download` is deprecated and will be removed in version 1.0.0. Downloads always resume when
possible. If you want to force a new download, use `force_download=True`.
warnings.warn(
Download attempt 1/6
```

```
Fetching 47 files: 100% |████████████████████████████████████████| 47/47 [00:00<00:00,
21094.95it/s]
Model downloaded successfully!
Starting Dynamo Frontend...
```

```
2026-03-12T09:02:42.908419Z  INFO dynamo_runtime::distributed: Initializing KV store discovery
backend
2026-03-12T09:02:42.908525Z  INFO dynamo_runtime::pipeline::network::manager: Initializing
NetworkManager with TCP request plane mode=tcp host=192.0.2.101 port=0s-assigned
2026-03-12T09:02:42.921778Z  INFO dynamo_llm::http::service::service_v2: Starting HTTP(S) service
proto="HTTP" address="0.0.0.0:8000"
Starting vLLM Worker 1 (GPUs 0-1, TP=2) with model: nvidia/Llama-3_3-Nemotron-Super-49B-v1_5
[2026-03-12 09:02:58] INFO __init__.py:53: dynamo.nixl_connect: Utilizing CuPy to enable GPU
acceleration.
[2026-03-12 09:02:59] INFO encode_worker_handler.py:39: Using cupy for array operations (GPU
mode).
2026-03-12T09:02:59.194799Z  INFO args.create_kv_transfer_config: Using VLLM defaults for
kv_transfer_config
2026-03-12T09:02:59.194858Z  INFO args.create_kv_events_config: Using env-var
DYN_VLLM_KV_EVENT_PORT=20081 to create kv_events_config
2026-03-12T09:02:59.194977Z  INFO args.overwrite_args: Using kv_events_config for publishing vLLM
kv events over zmq: KVEventsConfig(enable_kv_cache_events=True, publisher='zmq',
endpoint='tcp://*:20081', replay_endpoint=None, buffer_steps=10000, hwm=100000,
max_queue_size=100000, topic='/use_kv_events=True')
2026-03-12T09:02:59.215583Z  INFO dynamo_runtime::distributed: Initializing KV store discovery
backend
INFO 03-12 09:03:36 [model.py:1545] Using max model len 32768
INFO 03-12 09:03:36 [scheduler.py:229] Chunked prefill is enabled with
max_num_batched_tokens=2048.
INFO 03-12 09:03:36 [vllm.py:630] Asynchronous scheduling is enabled.
INFO 03-12 09:03:36 [vllm.py:637] Disabling NCCL for DP synchronization when using async
scheduling.
WARNING 03-12 09:03:36 [vllm.py:665] Enforce eager set, overriding optimization level to -O0
INFO 03-12 09:03:36 [vllm.py:765] Cudagraph is disabled under eager mode
Loading safetensors checkpoint shards:   0% Completed | 0/21 [00:00<?, ?it/s]
Loading safetensors checkpoint shards:   5% Completed | 1/21 [00:00<00:16, 1.21it/s]
Loading safetensors checkpoint shards:  10% Completed | 2/21 [00:01<00:15, 1.24it/s]
Loading safetensors checkpoint shards:  14% Completed | 3/21 [00:02<00:14, 1.23it/s]
Loading safetensors checkpoint shards:  19% Completed | 4/21 [00:03<00:13, 1.29it/s]
Loading safetensors checkpoint shards:  24% Completed | 5/21 [00:03<00:12, 1.26it/s]
Loading safetensors checkpoint shards:  29% Completed | 6/21 [00:04<00:11, 1.30it/s]
2026-03-12T09:03:42.756811Z  INFO core::__init__: Initializing a V1 LLM engine (v0.14.1) with
config: model='nvidia/Llama-3_3-Nemotron-Super-49B-v1_5', speculative_config=None,
tokenizer='nvidia/Llama-3_3-Nemotron-Super-49B-v1_5', skip_tokenizer_init=False,
tokenizer_mode=auto, revision=None, tokenizer_revision=None, trust_remote_code=True,
dtype=torch.bfloat16, max_seq_len=32768, download_dir=None, load_format=auto,
tensor_parallel_size=2, pipeline_parallel_size=1, data_parallel_size=1,
disable_custom_all_reduce=False, quantization=None, enforce_eager=True,
enable_return_routed_experts=False, kv_cache_dtype=auto, device_config=cuda,
structured_outputs_config=StructuredOutputsConfig(backend='auto', disable_fallback=False,
disable_any_whitespace=False, disable_additional_properties=False, reasoning_parser='',
reasoning_parser_plugin='', enable_i
2026-03-12T09:04:16.855648Z  INFO dynamo_llm::hub: ModelExpress download completed successfully
for model: nvidia/Llama-3_3-Nemotron-Super-49B-v1_5
Loading safetensors checkpoint shards: 100% Completed | 21/21 [00:16<00:00, 1.29it/s]
Loading safetensors checkpoint shards: 100% Completed | 21/21 [00:16<00:00, 1.29it/s]
(Worker_TP0 pid=2772)
2026-03-12T09:04:17.098937Z  INFO default_loader.load_weights: Loading weights took 16.34 seconds
2026-03-12T09:04:17.359703Z  INFO dynamo_llm::http::service::service_v2: chat endpoints enabled
```



```
2026-03-12T09:04:17.359758Z  INFO dynamo_llm::http::service::service_v2: completion endpoints
enabled
2026-03-12T09:04:17.386259Z  INFO dynamo_llm::discovery::watcher: Chat completions is ready
2026-03-12T09:04:17.401310Z  INFO dynamo_llm::discovery::watcher: Completions is ready
2026-03-12T09:04:17.401328Z  INFO dynamo_llm::discovery::watcher: added model
model_name="nvidia/Llama-3_3-Nemotron-Super-49B-v1_5" namespace="dynamo"
2026-03-12T09:04:17.516757Z  INFO gpu_model_runner.load_model: Model loading took 46.47 GiB memory
and 22.186278 seconds
2026-03-12T09:04:20.080224Z  INFO gpu_worker.determine_available_memory: Available KV cache
memory: 23.15 GiB
2026-03-12T09:04:20.352734Z  INFO kv_cache_utils.report_kv_cache_config: GPU KV cache size: 24,
7,696 tokens
2026-03-12T09:04:20.352829Z  INFO kv_cache_utils.report_kv_cache_config: Maximum concurrency for
32,768 tokens per request: 7.56x
2026-03-12T09:04:20.516822Z  INFO core.initialize_kv_caches: init engine (profile, create kv
cache, warmup model) took 2.62 seconds
2026-03-12T09:04:38.550821Z  INFO modelexpress_common::providers::huggingface: Downloaded model
files for nvidia/Llama-3_3-Nemotron-Super-49B-v1_5
2026-03-12T09:04:38.550911Z  INFO dynamo_llm::hub: ModelExpress download completed successfully
for model: nvidia/Llama-3_3-Nemotron-Super-49B-v1_5
2026-03-12T09:04:38.550999Z  INFO dynamo_llm::discovery::watcher: added model
model_name="nvidia/Llama-3_3-Nemotron-Super-49B-v1_5" namespace="dynamo"
```

Expected Output:

- Log output showing model download, frontend startup, both worker initializations and workers registering to frontend

Press Ctrl+C to exit the log stream (container continues running)

Key Log Indicators

- Model downloaded successfully! - Model weights downloaded
- Starting Dynamo Frontend... - Frontend initialization (port 8000)
- Starting vLLM Worker 1 (GPUs 0-1, TP=2) - First worker startup
- Starting vLLM Worker 2 (GPUs 2-3, TP=2) - Second worker startup
- VllmEngineMonitor initialized - Worker health monitoring active (appears once per worker)

Filter Important Logs Only

Dual worker initialization generates extensive logs from multiple processes. This filtered view shows only the essential milestones, enabling quick confirmation that both workers are starting correctly and registering to the frontend without the need to wade through verbose output.

```
docker logs dynamo-llm-dual-tp2 2>&1 | grep -iE "Model downloaded successfully|Starting Dynamo
Frontend|Starting vLLM Worker [12]|All services started|Frontend PID|Worker [12]
PID|VllmEngineMonitor initialized|added model"
```

```
root@phase3-nvidia:~/dynamo# docker logs dynamo-llm-dual-tp2 2>&1 | grep -iE "Model downloaded
successfully|Starting Dynamo Frontend|Starting vLLM Worker [12]|All services started|Frontend
PID|Worker [12] PID|VllmEngineMonitor initialized|added model"
Model downloaded successfully!
```

```
Starting Dynamo Frontend...
Starting VLLM Worker 1 (GPUs 0-1, TP=2) with model: nvidia/Llama-3_3-Nemotron-Super-49B-v1_5
Starting VLLM Worker 2 (GPUs 2-3, TP=2) with model: nvidia/Llama-3_3-Nemotron-Super-49B-v1_5
All services started.
Frontend PID: 100 (port 8000)
Worker 1 PID: 636 (GPUs 0-1)
Worker 2 PID: 1637 (GPUs 2-3)
2026-03-12T09:04:06.569453Z  INFO engine_monitor.__init__: VllmEngineMonitor initialized and
health check task started.
2026-03-12T09:04:17.401328Z  INFO dynamo_llm::discovery::watcher: added model
model_name="nvidia/Llama-3_3-Nemotron-Super-49B-v1_5" namespace="dynamo"
2026-03-12T09:04:28.278655Z  INFO engine_monitor.__init__: VllmEngineMonitor initialized and
health check task started.
2026-03-12T09:04:38.550999Z  INFO dynamo_llm::discovery::watcher: added model
model_name="nvidia/Llama-3_3-Nemotron-Super-49B-v1_5" namespace="dynamo"
root@phase3-nvidia:~#
```

Verify Dual Worker Deployment

Validate that the container is running and GPU resources are correctly allocated across both workers before proceeding to load balancing tests.

Check container status:

```
docker ps | grep dynamo-llm-dual-tp2
```

```
root@phase3-nvidia:~/dynamo# docker ps | grep dynamo-llm-dual-tp2
bf5f89fc158a   nvcr.io/nvidia/ai-dynamo/vllm-runtime:0.9.0-cuda13   "/opt/nvidia/nvidia_…"   8
minutes ago   Up 8 minutes
                                                                    dynamo-llm-dual-tp2
root@phase3-nvidia:~/dynamo#
```

Expected Output:

- Container shows Up status

Verify GPU Allocation Across Both Workers

```
nvidia-smi
root@phase3-nvidia:~# nvidia-smi
Tue Mar 10 14:02:37 2026
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 580.95.05    Driver Version: 580.95.05    CUDA Version: 13.0     |
+-----------------------------------------------------------------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf          Pwr:Usage/Cap | Memory-Usage  | GPU-Util  Compute M. |
|                                        |                       |               MIG M. |
+-----------------------------------------------------------------------------+
|   0  NVIDIA A100-SXM4-80GB  On | 00000000:1C:00.0 Off | 0 |
| N/A   40C   P0              74W / 500W | 74317MiB / 81920MiB | 0%      Default |
|                                        |                      | Disabled |
+-----------------------------------------------------------------------------+
|   1  NVIDIA A100-SXM4-80GB  On | 00000000:3E:00.0 Off | 0 |
```



```
| N/A   39C   P0             75W / 500W | 74317MiB / 81920MiB | 0%      Default |
|                                 |                      | Disabled |
+-----------------------------------------------------------------------------+
|   2  NVIDIA A100-SXM4-80GB  On | 00000000:4F:00.0 Off | 0 |
| N/A   38C   P0             66W / 500W | 74349MiB / 81920MiB | 0%      Default |
|                                 |                      | Disabled |
+-----------------------------------------------------------------------------+
|   3  NVIDIA A100-SXM4-80GB  On | 00000000:60:00.0 Off | 0 |
| N/A   39C   P0             71W / 500W | 74349MiB / 81920MiB | 0%      Default |
|                                 |                      | Disabled |
+-----------------------------------------------------------------------------+
|   4  NVIDIA A100-SXM4-80GB  On | 00000000:9E:00.0 Off | 0 |
| N/A   39C   P0             68W / 500W | 0MiB / 81920MiB | 0%      Default |
|                                 |                      | Disabled |
+-----------------------------------------------------------------------------+
|   5  NVIDIA A100-SXM4-80GB  On | 00000000:BE:00.0 Off | 0 |
| N/A   38C   P0             70W / 500W | 0MiB / 81920MiB | 0%      Default |
|                                 |                      | Disabled |
+-----------------------------------------------------------------------------+
|   6  NVIDIA A100-SXM4-80GB  On | 00000000:CE:00.0 Off | 0 |
| N/A   40C   P0             67W / 500W | 0MiB / 81920MiB | 0%      Default |
|                                 |                      | Disabled |
+-----------------------------------------------------------------------------+
|   7  NVIDIA A100-SXM4-80GB  On | 00000000:DE:00.0 Off | 0 |
| N/A   40C   P0             66W / 500W | 0MiB / 81920MiB | 0%      Default |
|                                 |                      | Disabled |
+-----------------------------------------------------------------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
| GPU   GI   CI        PID   Type   Process name                  GPU Memory Usage |
|=============================================================================|
|   0   N/A  N/A   1499405      C   VLLM::Worker_TP0                   74362MiB |
|   1   N/A  N/A   1499412      C   VLLM::Worker_TP1                   74362MiB |
|   2   N/A  N/A   1499423      C   VLLM::Worker_TP2                   74362MiB |
|   3   N/A  N/A   1499439      C   VLLM::Worker_TP3                   74362MiB |
+-----------------------------------------------------------------------------+

root@phase3-nvidia:~/dynamo#
```

Expected Output:

- GPUs 0-1: Active processes from Worker 1
- GPUs 2-3: Active processes from Worker 2
- GPUs 4-7: No active processes (available)

Validate Load Balancing Across Workers

This test verifies that requests are distributed across both workers by observing GPU utilization and response behaviour under concurrent load.

Note: This test requires two terminals.

Terminal 1: Send Sequential Long-Running Requests

```
root@phase3-nvidia:~ #
MODEL_NAME="nvidia/Llama-3_3-Nemotron-Super-49B-v1_5"
URL="http://localhost:8000/v1/chat/completions"
echo "=== Sending Request A at $(date '+%H:%M:%S') ==="
curl -s "$URL" -H "Content-Type: application/json" -d "{
  \"model\": \"$MODEL_NAME\",
  \"messages\": [
    {\"role\": \"user\", \"content\": \"Write a comprehensive 2000-word technical deep-dive on
tensor parallelism architecture, implementation strategies, and performance optimization
techniques in distributed inference systems. This is REQUEST-A for load balancing verification.\"}
  ],
  \"max_tokens\": 1500,
  \"temperature\": 0.7
}" > /tmp/request_A.json &

sleep 15

echo "=== Sending Request B at $(date '+%H:%M:%S') ==="
curl -s "$URL" -H "Content-Type: application/json" -d "{
  \"model\": \"$MODEL_NAME\",
  \"messages\": [
    {\"role\": \"user\", \"content\": \"Write a comprehensive 2000-word technical analysis of
horizontal scaling patterns, load distribution mechanisms, and automatic request routing in modern
AI inference frameworks. This is REQUEST-B for load balancing verification.\"}
  ],
  \"max_tokens\": 1500,
  \"temperature\": 0.7
}" > /tmp/request_B.json &

wait
echo "=== Both requests completed at $(date '+%H:%M:%S') ==="
=== Sending Request A at 07:44:26 ===
[1] 1686608
=== Sending Request B at 07:44:41 ===
[2] 1686678
[1]  Done                    curl -s "$URL" -H "Content-Type: application/json" -d "{
  \"model\": \"$MODEL_NAME\",
  \"messages\": [
    {\"role\": \"user\", \"content\": \"Write a comprehensive 2000-word technical deep-dive on
tensor parallelism architecture, implementation strategies, and performance optimization
techniques in distributed inference systems. This is REQUEST-A for load balancing verification.\"}
  ],
  \"max_tokens\": 1500,
  \"temperature\": 0.7
}" > /tmp/request_A.json
[2]+  Done                    curl -s "$URL" -H "Content-Type: application/json" -d "{
  \"model\": \"$MODEL_NAME\",
  \"messages\": [
```



```
    {\"role\": \"user\", \"content\": \"Write a comprehensive 2000-word technical analysis of
horizontal scaling patterns, load distribution mechanisms, and automatic request routing in modern
AI inference frameworks. This is REQUEST-B for load balancing verification.\"}
  ],
  \"max_tokens\": 1500,
  \"temperature\": 0.7
}" > /tmp/request_B.json
=== Both requests completed at 07:45:34 ===
root@phase3-nvidia:~#
```

Terminal 2: Monitor GPU Utilization

```
root@phase3-nvidia:~# watch -n 1 "nvidia-smi --query-gpu=index,name,utilization.gpu,memory.used --
format=csv,noheader,nounits"
```

Press Ctrl + C to stop monitoring after GPU utilization verification.

```
Every 1.0s: nvidia-smi --query-gpu=index,n...  phase3-nvidia:

0, NVIDIA A100-SXM4-80GB, 100, 74321
1, NVIDIA A100-SXM4-80GB, 100, 74321
2, NVIDIA A100-SXM4-80GB, 0, 74353
3, NVIDIA A100-SXM4-80GB, 0, 74353
4, NVIDIA A100-SXM4-80GB, 0, 0
5, NVIDIA A100-SXM4-80GB, 0, 0
6, NVIDIA A100-SXM4-80GB, 0, 0
7, NVIDIA A100-SXM4-80GB, 0, 0
```

```
Every 1.0s: nvidia-smi --query-gpu=index,n...  phase3-nvidia:

0, NVIDIA A100-SXM4-80GB, 100, 74321
1, NVIDIA A100-SXM4-80GB, 100, 74321
2, NVIDIA A100-SXM4-80GB, 100, 74353
3, NVIDIA A100-SXM4-80GB, 100, 74353
4, NVIDIA A100-SXM4-80GB, 0, 0
5, NVIDIA A100-SXM4-80GB, 0, 0
6, NVIDIA A100-SXM4-80GB, 0, 0
7, NVIDIA A100-SXM4-80GB, 0, 0
```

```
Every 1.0s: nvidia-smi --query-gpu=index,n...  phase3-nvidia:

0, NVIDIA A100-SXM4-80GB, 0, 74321
1, NVIDIA A100-SXM4-80GB, 0, 74321
2, NVIDIA A100-SXM4-80GB, 100, 74353
3, NVIDIA A100-SXM4-80GB, 100, 74353
4, NVIDIA A100-SXM4-80GB, 0, 0
5, NVIDIA A100-SXM4-80GB, 0, 0
6, NVIDIA A100-SXM4-80GB, 0, 0
7, NVIDIA A100-SXM4-80GB, 0, 0
```


Expected Load Balancing Behavior

- **0-15 seconds:** Only one GPU pair active (either 0-1 OR 2-3) shows utilization (e.g.,`100, 100, 0, 0`)
- **15+ seconds:** Both GPU pairs show utilization simultaneously (e.g., `100,100,100,100`)
- **After the first request completes:** Only one GPU pair (serving the remaining request) shows utilization (e.g., `0, 0, 100, 100`)
- **Clear Evidence:** Sequential activation proves automatic load balancing across workers
- **Observation Window:** ~30-60 seconds of overlapping GPU activity for verification

Verify Load Distribution via Response Outputs

```
cat /tmp/out_1.json | jq '.choices[0].message.content' | head -3
cat /tmp/out_2.json | jq '.choices[0].message.content' | head -3
```

```
root@phase3-nvidia:~# cat /tmp/out_1.json | jq '.choices[0].message.content' | head -3
"<think>\nOkay, I need to write a comprehensive technical deep-dive on tensor parallelism in large language model inference. Let me start by understanding what tensor parallelism is. From what I remember, tensor parallelism is a method of splitting the computation of a neural network across multiple devices, like GPUs or TPUs, by dividing the tensors (matrices) involved in the layers. This is different from data parallelism, which splits the data across devices but keeps the model on each device. So, tensor parallelism allows for larger models by distributing the model's parameters across devices.\n\nFirst, I should outline the structure of the article. The user wants it to cover architecture, implementation details, and performance characteristics. Let me break it down into sections.\n\n1. **Introduction**: Explain the need for tensor parallelism in LLMs, the challenges of scaling models, and how tensor parallelism addresses these.\n2. **Background**: Briefly explain parallelism types in deep learning—data, model, pipeline, and tensor parallelism. Compare them.\n3. **Architecture of Tensor Parallelism**: How the model is split across devices. For example, in a transformer layer, splitting the attention heads or the feed-forward layers. Maybe talk about the placement of operations on different devices and communication patterns.\n4. **Implementation Details**: How to actually implement tensor parallelism. Frameworks like PyTorch or TensorFlow support this? Or custom implementations? Discuss tensor splitting techniques, communication primitives (like all-reduce, all-gather), and synchronization.\n5. **Performance Characteristics**: Discuss the trade-offs. Latency, throughput, communication overhead, memory usage. How does tensor parallelism compare to data or pipeline parallelism? When is it most effective?\n6. **Case Studies**: Examples of systems using tensor parallelism, like Megatron-LM, DeepSpeed, or others. How they handle tensor parallelism and their results.\n7. **Challenges and Limitations**: What are the current issues with tensor parallelism? Like load balancing, increased communication, complexity in implementation.\n8. **Future Directi"
```

```
root@phase3-nvidia:~# cat /tmp/out_2.json | jq '.choices[0].message.content' | head -3
"<think>\nOkay, I need to write a comprehensive 3000-word technical deep-dive on tensor parallelism
in large language model inference. Let me start by understanding what tensor parallelism is. From
what I remember, tensor parallelism is a method to split the workload of a neural network across
```

```
multiple GPUs or devices by dividing the tensors involved in the computations. This is different

from data parallelism, which splits the data across devices. But I should make sure I have the

correct definition.\n\nFirst, the introduction should explain the context: large language models

(LLMs) are huge, so training and inference require distributing the computation. Tensor
parallelism

is one of the techniques used for this. I should mention the challenges of LLMs, like the number
of parameters and the computational requirements. Then, introduce tensor parallelism as a solution
that splits the model itself across devices, allowing for more efficient use of resources compared
to data parallelism.\n\nNext, the architecture section. I need to explain how tensor parallelism
works at a high level. Maybe start with the concept of splitting tensors along different
dimensions. For example, in a matrix multiplication, you can split the input and output matrices
across devices.


Each device handles a portion of the computation, and then they communicate to combine the
results. I should mention how this applies to different layers in a neural network, like attention
layers and feed-forward layers.\n\nThen, implementation details. Here, I should go into more
specifics. How exactly is the splitting done? For instance, in a linear layer, the weight matrix
can be split along the head dimension (output) or the hidden dimension (input). Each device would
have a portion of the weight matrix. Then, during computation, each device processes its part, and
allreduce operations are used to combine the results. I should talk about the communication
patterns, like all-gather, reduce-scatter, etc., and how they affect performance.\n\nAlso, the
integration with frameworks like PyTorch or TensorFlow. Maybe mention libraries like NVIDIA's
Megatron-LM or Microsoft's DeepSpeed that implement
```

Expected Output:

- Distinct response outputs for each request, indicating they were processed independently across different workers

Service Endpoint Information

The dual-worker deployment exposes a single OpenAI-compatible API endpoint that automatically distributes requests across both registered workers. This enables higher throughput while maintaining a simple, unified client interface.

Endpoint Details

- **URL:** http://<NVIDIA_GPU_SERVER_IP>:8000/v1/chat/completions
- **Model:** nvidia/Llama-3_3-Nemotron-Super-49B-v1_5
- **Context Length:** 32K tokens (configurable up to 128K)
- **API Standard:** OpenAI-compatible
- **Authentication:** None (internal network deployment)
- **Architecture:**
  - Single frontend
  - 2 workers (Worker 1: GPUs 0-1, Worker 2: GPUs 2-3, both TP=2)
  - Load Balancing: Automatic via Dynamo frontend

Example Python Integration

```
import httpx
# Single endpoint - Dynamo handles load balancing automatically
response = httpx.post(
    "http://< NVIDIA_GPU_SERVER_IP>:8000/v1/chat/completions",
    json={
        "model": "nvidia/Llama-3_3-Nemotron-Super-49B-v1_5",
        "messages": [
            {"role": "system", "content": "You are a helpful assistant."},
            {"role": "user", "content": "Explain horizontal scaling in AI inference."}
        ],
        "max_tokens": 500,
        "temperature": 0.7
    }
)
result = response.json()
answer = result["choices"][0]["message"]["content"]
print(answer)
```

Example cURL Request

```
curl -X POST http://< NVIDIA_GPU_SERVER_IP>:8000/v1/chat/completions \
  -H "Content-Type: application/json" \
  -d '{
    "model": "nvidia/Llama-3_3-Nemotron-Super-49B-v1_5",
    "messages": [{"role": "user", "content": "What is horizontal scaling?"}],
    "max_tokens": 200,
    "temperature": 0.7
  }'
```

```
root@phase3-nvidia:~# curl -X POST http://192.0.2.101:8000/v1/chat/completions \
-H "Content-Type: application/json" \
-d '{
  "model": "nvidia/Llama-3_3-Nemotron-Super-49B-v1_5",
  "messages": [{"role": "user", "content": "What is horizontal scaling?"}],
  "max_tokens": 200,
  "temperature": 0.7}'{"id":"chatcmpl-07a2fdae-7736-4b35-b379-
c5d586445ec8","choices":[{"index":0,"message":{"content":"<think>\nOkay, so I need to understand
what horizontal scaling is. Let me start by breaking down the term. \"Horizontal\" makes me think
of something that's spread out sideways, and \"scaling\" usually refers to increasing or
decreasing capacity. So maybe horizontal scaling has something to do with adding more resources
side by side?\n\nI remember hearing about scaling in the context of servers or computers. There's
also vertical scaling, right? Vertical scaling might be about adding more power to a single
machine, like upgrading the CPU or RAM. So horizontal scaling would be the opposite—adding more
machines instead of making one machine stronger.\n\nLet me think of an example. If a website is
getting a lot of traffic, the server might get overloaded. To handle more users, you could either
upgrade the existing server (vertical) or add more servers to share the load (horizontal). That
makes sense. So horizontal scaling is about distributing the workload across multiple servers or
instances.\n\nBut wait, how exactly does that work?", "role":"assistant", "reasoning_content"
:null},"finish_reason":"length"}],"created":1773393635,"model":"nvidia/Llama-3_3-Nemotron-Super-
49B-v1_5","object":"chat.completion","usage":{"prompt_tokens":20,"completion_tokens"
:200,"total_tokens":220}} root@phase3-nvidia:~#
```

Note: Replace <NVIDIA_GPU_SERVER_IP> with the internal IP address of the GPU server

Alternative Configuration: Distributed Multi-Container Deployment

The Dual TP=2 deployment described above uses a single container running multiple processes (1 frontend + 2 workers). For production environments requiring maximum fault isolation and independent component lifecycle management, an alternative distributed architecture can be deployed using separate containers for the frontend and each worker.

This section describes the distributed multi-container approach as an optional alternative. Developers can implement this architecture if their operational requirements demand independent component restarts, better fault isolation, or preparation for multi-node scaling.

Architecture Comparison: Single vs Multi-Container

Understanding the trade-offs between single-container and multi-container deployments helps inform the architectural decision based on operational requirements.

Aspect	Single Container	Multi-Container
Deployment Complexity	Simple (1 docker run)	Complex (3 docker run commands)
Fault Isolation	All processes share same failure domain	Independent component failures
Component Restarts	Restart all together	Restart frontend or workers independently
Monitoring	Single container to watch	3 containers to monitor
Communication	In-process (fastest)	Network-based (slight overhead)
Scaling	Add workers requires code change	Add worker containers dynamically
Production Readiness	Good for single-node	Better for multi-node clusters
Operational Overhead	Low	Higher

Distributed Architecture Overview

The distributed multi-container architecture separates the Dynamo frontend and vLLM workers into independent containers. Workers register with the frontend via etcd service discovery, enabling dynamic worker addition/removal and independent lifecycle management.

Component Distribution

- **Container 1 (Frontend):** Dynamo frontend process exposing port 8000
- **Container 2 (Worker 1):** vLLM worker on GPUs 0-1 (TP=2), registers to etcd via Distributed Runtime
- **Container 3 (Worker 2):** vLLM worker on GPUs 2-3 (TP=2), registers to etcd via Distributed Runtime
- **Infrastructure:** etcd (service discovery) and NATS (messaging) - already deployed via docker-compose

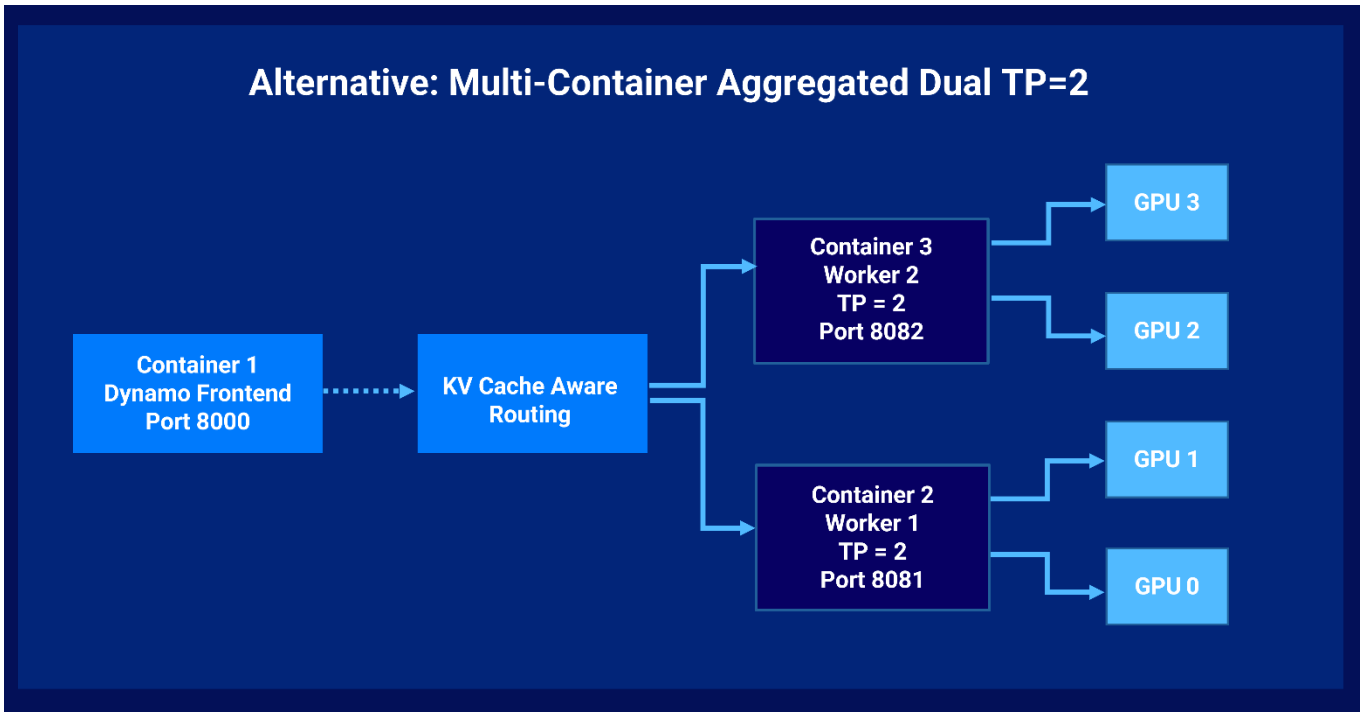


Figure 11 - Aggregated Serving (Dual Workers, TP=2) with Multi-Container Deployment on NVIDIA Dynamo

Communication Flow

- Workers start and automatically register their endpoints to etcd via Dynamo's Distributed Runtime component (from dynamo.runtime)
- Frontend watches etcd prefix paths (/services/{namespace}/{component}/{endpoint}) and receives real-time updates when workers register
- Frontend routes requests to discovered workers via their DYN_SYSTEM_PORT
- Workers process requests and return responses
- Dynamic discovery: workers can be added/removed at any time without frontend restart (automatic lease-based cleanup)

Deployment Steps (Informational)

The following steps outline how to deploy the distributed multi-container architecture. (*These commands are provided for reference.)*

Step 1: Ensure Infrastructure Services are Running

Verify etcd and NATS are running:

```
docker compose -f ~/dynamo/deploy/docker-compose.yml ps
```

Expected Output:

- Both dynamo-etcd and dynamo-nats containers showing "Up" status

Step 2: Deploy Worker 1 Container

Launch the first worker on GPUs 0-1:

```
docker run -d --name dynamo-worker-1 --restart always --gpus '"device=0,1"' --network host --
ipc=host --ulimit memlock=-1 --ulimit stack=67108864 -e HF_TOKEN=$HF_TOKEN -e PYTHONHASHSEED=0 -e
ETCD_ENDPOINTS=http://localhost:2379 -e DYN_SYSTEM_PORT=8081 -v ~/dynamo/container:/workspace -v
~/dynamo/container/.cache:/home/dynamo/.cache -w /workspace -u 1000:1000 $DYNAMO_IMAGE bash -c
"python -m dynamo.vllm --model nvidia/Llama-3_3-Nemotron-Super-49B-v1_5 --tensor-parallel-size 2 -
-max-model-len 32768 --trust-remote-code --enforce-eager --connector none"
```

Step 3: Deploy Worker 2 Container

Launch the second worker on GPUs 2-3:

```
docker run -d --name dynamo-worker-2 --restart always --gpus '"device=2,3"' --network host --
ipc=host --ulimit memlock=-1 --ulimit stack=67108864 -e HF_TOKEN=$HF_TOKEN -e PYTHONHASHSEED=0 -e
ETCD_ENDPOINTS=http://localhost:2379 -e DYN_SYSTEM_PORT=8082 -v ~/dynamo/container:/workspace -v
~/dynamo/container/.cache:/home/dynamo/.cache -w /workspace -u 1000:1000 $DYNAMO_IMAGE bash -c
"python -m dynamo.vllm --model nvidia/Llama-3_3-Nemotron-Super-49B-v1_5 --tensor-parallel-size 2 -
-max-model-len 32768 --trust-remote-code --enforce-eager --connector none"
```

Step 4: Deploy Frontend Container

Launch the Dynamo frontend:

```
docker run -d --name dynamo-frontend --restart always --network host -e PYTHONHASHSEED=0 -e
ETCD_ENDPOINTS=http://localhost:2379 -v ~/dynamo/container:/workspace -w /workspace -u 1000:1000
$DYNAMO_IMAGE bash -c "python -m dynamo.frontend --router-mode kv --http-port 8000"
```

Step 5: Verify Worker Registration

Check that both workers registered with the frontend:

```
docker logs dynamo-frontend | grep -i "worker.*registered"
```

Expected Output:

- Log entries showing both workers (ports 8081 and 8082) successfully registered

Step 6: Test Load Balancing

Send test requests to verify automatic load balancing:

```
MODEL_NAME="nvidia/Llama-3_3-Nemotron-Super-49B-v1_5"
for i in {1..5}; do
  curl -s http://localhost:8000/v1/chat/completions -H "Content-Type: application/json" -d "{
    \"model\": \"$MODEL_NAME\",
    \"messages\": [{\"role\": \"user\", \"content\": \"Test request $i\"}],
    \"max_tokens\": 20
  }" | jq '.choices[0].message.content'
done
```

Expected Output:

- Responses from both workers, demonstrating load distribution

Operational Considerations

The distributed multi-container architecture introduces operational complexity that should be carefully evaluated against the benefits of fault isolation and independent component management.

Advantages

- **Independent Restarts:** Restart individual workers without affecting the frontend or other workers
- **Fault Isolation:** Worker crash doesn't bring down the entire service
- **Dynamic Scaling:** Add or remove worker containers without redeploying the frontend
- **Production Patterns:** Aligns with microservices and container orchestration best practices

Challenges

- **Startup Coordination:** No required startup order due to dynamic etcd watch mechanism via Distributed Runtime, but starting workers first is recommended so frontend is immediately ready to serve
- **Network Overhead:** Inter-container communication adds latency compared to in-process communication
- **Monitoring Complexity:** Multiple containers require coordinated logging and health checks
- **Debugging:** Distributed traces across containers more complex than single-process debugging
- **Lease Management:** Workers maintain etcd leases (10-second TTL) with automatic cleanup on crash, but network issues can cause false removals

When to Use Multi-Container Architecture

- Production environments with strict uptime requirements
- Multi-node GPU clusters (future scaling beyond single server)
- Environments with frequent worker updates or restarts

When to Use Single-Container Architecture

- Single-server deployments
- Development and testing environments
- Simpler operational requirements

Deploy LLM with Disaggregated Serving on NVIDIA Dynamo

The deployment of a Large Language Model (LLM) using the disaggregated serving architecture in NVIDIA Dynamo is designed for workloads that require fine-grained control over inference latency and efficient utilization of GPU resources. In this approach, the prefill phase (prompt processing) and decode phase (token generation) are executed on separate worker processes across dedicated GPU sets.

This separation enables independent optimization of Time to First Token (TTFT) and Inter-Token Latency (ITL), resulting in improved latency consistency under concurrent workloads. It is particularly suited for retrieval-augmented and document-driven use cases, where input contexts are large and output responses are relatively short.

The deployment involves selecting an appropriate model, planning GPU and memory allocation, configuring runtime parameters, and deploying dedicated prefill and decode workers. Proper coordination between these components ensures efficient execution and stable inference performance.

Model Selection for Disaggregated Mode

This section outlines the model characteristics and resource requirements for deploying LLM workers in a disaggregated architecture.

- **Model:** nvidia/Llama-3_3-Nemotron-Super-49B-v1_5 (49B parameters, ~100GB)
- **Architecture:** Pure transformer model - fully compatible with NIXL connector
- **Context Window:** 32K tokens (configurable up to 128K)
- **GPU Requirements:** TP=2 per worker (~50GB per GPU + ~30GB for KV cache)

Memory Implications (Critical)

Disaggregated deployment introduces additional memory overhead due to separate model instances for prefill and decode workers.

- **Aggregated Mode (TP=4):** 1 model copy across 4 GPUs = ~100GB total weights
- **Disaggregated Mode:** Each worker loads full model copy = ~400GB total weights (4× multiplier)
- **Per-Worker KV Headroom:** ~110GB available per GPU pair after model weights

- **Context Length Impact:** NIXL transfer overhead scales linearly
  - 8K: ~8 ms
  - 32K: ~32 ms
  - 128K: May become a dominant factor

Architectural Characteristics

The disaggregated architecture separates prompt processing and token generation across dedicated GPU workers.

- **Decode Workers:** GPUs 0-1 (TP=2) - handle token generation
- **Prefill Workers:** GPUs 2-3 (TP=2) - handle prompt processing
- **NIXL:** Manages KV cache transfer between prefill and decode workers
- **Tensor Parallelism:** Each worker uses multiple GPUs with --tensor-parallel-size 2

Stop Aggregated Worker (If Running)

Before deploying disaggregated workers, any existing aggregated worker must be stopped to free GPU resources. Both architectures cannot run simultaneously on the same GPU set due to resource conflicts.

Stop and Remove Aggregated Worker

```
docker stop dynamo-llm
```

```
root@phase3-nvidia:~ # docker stop dynamo-llm
dynamo-llm
root@phase3-nvidia:~#
```

```
docker rm dynamo-llm
```

```
root@phase3-nvidia:~# docker rm dynamo-llm
dynamo-llm
root@phase3-nvidia:~#
```

Expected Output:

- Container name is echoed twice (once for stop, once for remove)

Verify GPU Availability

```
nvidia-smi
```

```
root@phase3-nvidia:~/dynamo# nvidia-smi
Thu Mar 12 09:12:03 2026
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 580.95.05    Driver Version: 580.95.05    CUDA Version: 13.0     |
+-----------------------------------------+------------------------+----------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf          Pwr:Usage/Cap | Memory-Usage  | GPU-Util  Compute M. |
|                                        |                        |              MIG M. |
+-----------------------------------------+------------------------+----------+
|   0  NVIDIA A100-SXM4-80GB  On | 00000000:1C:00.0 Off | 0 |
| N/A   40C   P0             74W / 500W | 74317MiB / 81920MiB | 0%      Default |
|                                        |                      | Disabled |
+-----------------------------------------+------------------------+----------+
|   1  NVIDIA A100-SXM4-80GB  On | 00000000:3E:00.0 Off | 0 |
| N/A   39C   P0             75W / 500W | 74317MiB / 81920MiB | 0%      Default |
|                                        |                      | Disabled |
+-----------------------------------------+------------------------+----------+
|   2  NVIDIA A100-SXM4-80GB  On | 00000000:4F:00.0 Off | 0 |
| N/A   38C   P0             66W / 500W | 74349MiB / 81920MiB | 0%      Default |
|                                        |                      | Disabled |
+-----------------------------------------+------------------------+----------+
|   3  NVIDIA A100-SXM4-80GB  On | 00000000:60:00.0 Off | 0 |
| N/A   39C   P0             71W / 500W | 74349MiB / 81920MiB | 0%      Default |
|                                        |                      | Disabled |
+-----------------------------------------+------------------------+----------+
|   4  NVIDIA A100-SXM4-80GB  On | 00000000:9E:00.0 Off | 0 |
| N/A   39C   P0             68W / 500W | 0MiB / 81920MiB | 0%      Default |
|                                        |                      | Disabled |
+-----------------------------------------+------------------------+----------+
|   5  NVIDIA A100-SXM4-80GB  On | 00000000:BE:00.0 Off | 0 |
| N/A   38C   P0             70W / 500W | 0MiB / 81920MiB | 0%      Default |
|                                        |                      | Disabled |
+-----------------------------------------+------------------------+----------+
|   6  NVIDIA A100-SXM4-80GB  On | 00000000:CE:00.0 Off | 0 |
| N/A   40C   P0             67W / 500W | 0MiB / 81920MiB | 0%      Default |
|                                        |                      | Disabled |
+-----------------------------------------+------------------------+----------+
|   7  NVIDIA A100-SXM4-80GB  On | 00000000:DE:00.0 Off | 0 |
| N/A   40C   P0             66W / 500W | 0MiB / 81920MiB | 0%      Default |
|                                        |                      | Disabled |
+-----------------------------------------+------------------------+----------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
| GPU   GI   CI        PID   Type   Process name                  GPU Memory Usage |
|=============================================================================|
| No running processes found                                                  |
+-----------------------------------------------------------------------------+
```

Expected Output:

- All GPUs show near 0% utilization and minimal memory usage

Create Disaggregated Launch Script

The disaggregated launch script initializes the Dynamo frontend and deploys separate **decode** and **prefill workers** on dedicated GPU sets. In this configuration, the decode worker uses GPUs 0-1 and the prefill worker uses GPUs 2-3, with each worker configured using **tensor parallelism (TP=2)** to efficiently distribute the model across multiple GPUs.

The deployment uses the vLLM runtime for optimized model execution, enabling efficient handling of large models through KV cache management and parallel processing. The selected model must be compatible with disaggregated serving and NIXL-based KV cache transfer to ensure correct coordination between worker types.

The NIXL connector facilitates KV cache communication between prefill and decode workers, allowing prompt processing and token generation to operate independently. The prefill worker processes input prompts and generates KV cache, while the decode worker performs token generation using the transferred cache.

This architecture improves throughput and latency consistency under concurrent workloads by allowing prefill and decode phases to scale independently across GPU resources.

Navigate to the Dynamo Directory

```
cd ~/dynamo
```

```
root@phase3-nvidia:~ # cd ~/dynamo
root@phase3-nvidia:~/dynamo#
```

Create Launch Script

```
cat << 'EOF' > container/launch_llm_disaggregated.sh
#!/bin/bash
set -e

export PYTHONHASHSEED=0
MODEL="nvidia/Llama-3_3-Nemotron-Super-49B-v1_5"

echo "Starting Dynamo Frontend..."
python -m dynamo.frontend &

echo "Starting DISAGGREGATED mode deployment with model: $MODEL"

echo "Starting Decode Worker (GPUs 0-1, TP=2)..."
CUDA_VISIBLE_DEVICES=0,1 python3 -m dynamo.vllm \
  --model "$MODEL" \
  --tensor-parallel-size 2 \
  --max-model-len 32768 \
  --trust-remote-code \
  --enforce-eager \
```

```
  --connector nixl \
  --is-decode-worker &

echo "Starting Prefill Worker (GPUs 2-3, TP=2)..."
CUDA_VISIBLE_DEVICES=2,3 python3 -m dynamo.vllm \
  --model "$MODEL" \
  --tensor-parallel-size 2 \
  --max-model-len 32768 \
  --trust-remote-code \
  --enforce-eager \
  --connector nixl \
  --is-prefill-worker \
  --kv-events-config '{"publisher":"zmq","topic":"kv-
events","endpoint":"tcp://*:20082","enable_kv_cache_events":true}' &

echo "All services starting... waiting for initialization..."
sleep 60

echo "Deployment ready!"
wait
EOF


root@phase3-nvidia:~/dynamo# cat << 'EOF' > container/launch_llm_disaggregated.sh
#!/bin/bash
set -e

export PYTHONHASHSEED=0
MODEL="nvidia/Llama-3_3-Nemotron-Super-49B-v1_5"

# Function to check if a port is available
check_port() {
    local port=$1
    if netstat -tlnp 2>/dev/null | grep -q ":$port "; then
        echo "ERROR: Port $port is already in use"
        netstat -tlnp | grep ":$port "
        return 1
    fi
    return 0
}

# Function to wait for worker registration
wait_for_worker_registration() {
    local worker_type=$1
    local max_attempts=30
    local attempt=0

EOF
echo "Monitor logs with: docker logs -f dynamo-llm-disaggregated"
prefill=$PREFILL_PID register"
root@phase3-nvidia:~/dynamo#
```

Expected Output:

- Script file created successfully

Fix Line Endings for cross-platform compatibility

```
sed -i 's/\r$//' container/launch_llm_disaggregated.sh
```

```
root@phase3-nvidia:~/dynamo# sed -i 's/\r$//' container/launch_llm_disaggregated.sh
root@phase3-nvidia:~/dynamo#
```

Make Launch Script Executable

```
chmod +x container/launch_llm_disaggregated.sh
```

```
root@phase3-nvidia:~/dynamo# chmod +x container/launch_llm_disaggregated.sh
root@phase3-nvidia:~/dynamo#
```

Pre-download the model to avoid race conditions

Pre-download the model to the shared cache directory before starting multiple workers. This prevents concurrent download attempts by multiple processes and ensures faster, reliable worker initialization.

Download Model to Shared Cache

```
docker run --rm --gpus '"device=0"' -e HF_TOKEN=$HF_TOKEN -v
$(pwd)/container/.cache:/home/dynamo/.cache -u 1000:1000 $DYNAMO_IMAGE python3 -c "from
huggingface_hub import snapshot_download; snapshot_download('nvidia/Llama-3_3-Nemotron-Super-49B-
v1_5', cache_dir='/home/dynamo/.cache/huggingface/hub')"
```

```
root@phase3-nvidia:~/dynamo# docker run --rm --gpus '"device=0"' -e HF_TOKEN=$HF_TOKEN -v
$(pwd)/container/.cache:/home/dynamo/.cache -u 1000:1000 $DYNAMO_IMAGE python3 -c "from
huggingface_hub import snapshot_download; snapshot_download('nvidia/Llama-3_3-Nemotron-Super-49B-
v1_5', cache_dir='/home/dynamo/.cache/huggingface/hub')"

==========
== CUDA ==
==========

CUDA Version 13.0.2
Container image Copyright (c) 2016-2023, NVIDIA CORPORATION & AFFILIATES. All rights reserved.

This container image and its contents are governed by the NVIDIA Deep Learning Container License.
By pulling and using the container, you accept the terms and conditions of this license:
https://developer.nvidia.com/ngc/nvidia-deep-learning-container-license

A copy of this license is made available in this container at /NGC-DL-CONTAINER-LICENSE for your
convenience.

Fetching 47 files: 100%|████████████████████████████████████| 47/47 [00:46<00:00, 1.02it/s]
root@phase3-nvidia:~/dynamo#
```

Expected Output:

- Model download progress displayed
- Container exits successfully after completion

Note: This step typically takes 2-5 minutes depending on network speed. The model is downloaded once and shared by all workers.

Clean Up Previous Deployments

Before deploying disaggregated workers, ensure that no previous containers or processes are holding required resources or port bindings. This helps prevent worker startup issues and ZMQ/NIXL port conflicts during registration.

Stop and Remove Existing Containers

```
docker stop dynamo-llm dynamo-llm-disaggregated 2>/dev/null || true
docker rm dynamo-llm dynamo-llm-disaggregated 2>/dev/null || true
```

```
root@phase3-nvidia:~/dynamo# docker stop dynamo-llm dynamo-llm-disaggregated 2>/dev/null || true
root@phase3-nvidia:~/dynamo# docker rm dynamo-llm dynamo-llm-disaggregated 2>/dev/null || true
root@phase3-nvidia:~/dynamo#
```

Release NIXL/ZMQ Ports

```
sudo netstat -tlnp | grep -E ":(5600|20082)" | awk '{print $7}' | cut -d'/' -f1 | xargs -r sudo kill -9 2>/dev/null || true
```

```
root@phase3-nvidia:~/dynamo# sudo netstat -tlnp | grep -E ":(5600|20082)" | awk '{print $7}' | cut -d'/' -f1 | xargs -r sudo kill -9 2>/dev/null || true
root@phase3-nvidia:~/dynamo#
```

Allow Time for Port Cleanup

```
Sleep 5
```

```
root@phase3-nvidia:~/dynamo# sleep 5
root@phase3-nvidia:~/dynamo#
```

Expected Output:

- No output (silent success)
- Required ports are available for the new deployment

Deploy Disaggregated Workers as a Background Daemon

The disaggregated deployment runs as a background container with an automatic restart policy. This ensures service continuity across SSH disconnections, container failures, and system reboots. Host networking is used for worker discovery and NIXL communication.

Verify Environment Variables

```
echo $HF_TOKEN
echo $DYNAMO_IMAGE
```

Expected Output:

- Both variables display valid values

Set Variables (If Not Defined)

```
export HF_TOKEN=hf_ xxxxxxxxxxxxxxxx
export DYNAMO_IMAGE=nvcr.io/nvidia/ai-dynamo/vllm-runtime:0.9.0-cuda13
```

Run Disaggregated LLM Container

```
docker run -d --name dynamo-llm-disaggregated --restart always --gpus '"device=0,1,2,3"' --network host --ipc=host --ulimit memlock=-1 --ulimit stack=67108864 -e HF_TOKEN=$HF_TOKEN -e PYTHONHASHSEED=0 -v $(pwd)/container:/workspace -v $(pwd)/container/.cache:/home/dynamo/.cache -w /workspace -u 1000:1000 $DYNAMO_IMAGE bash -c "/workspace/launch_llm_disaggregated.sh"
```

```
root@phase3-nvidia:~/dynamo# docker run -d --name dynamo-llm-disaggregated --restart always --gpus '"device=0,1,2,3"' --network host --ipc=host --ulimit memlock=-1 --ulimit stack=67108864 -e HF_TOKEN=$HF_TOKEN -e PYTHONHASHSEED=0 -v $(pwd)/container:/workspace -v $(pwd)/container/.cache:/home/dynamo/.cache -w /workspace -u 1000:1000 $DYNAMO_IMAGE bash -c "/workspace/launch_llm_disaggregated.sh"
Unable to find image 'nvcr.io/nvidia/ai-dynamo/vllm-runtime:0.9.0' locally
0.9.0: Pulling from nvidia/ai-dynamo/vllm-runtime
2cd52cbb1ebe: Pulling fs layer
644e9b203583: Pulling fs layer
9c3d619183d2: Pulling fs layer
5a8b1a552210: Pulling fs layer
28ccdd303f71: Pulling fs layer
e87500e69896: Pulling fs layer
02cb0e091e33: Pulling fs layer
d9f5ed6b9489: Pulling fs layer
15a17189b2df: Pulling fs layer
f62ec7ec74c9: Pulling fs layer
7f7602a82106: Pulling fs layer
4f4fb700ef54: Pulling fs layer
2234a8943651: Pulling fs layer
f4bf79a484c1: Pulling fs layer
fe4bc3f62624: Pulling fs layer
369c971ba68f: Pulling fs layer
```



```
3688b2605ef5: Pulling fs layer
6fdb3c6afc23: Pull complete
0dcadf247b6: Pull complete
1fa303b8ab97: Pull complete
6dc49fc78db9: Pull complete
1808a1121fd4: Pull complete
c377909f72ed: Pull complete
ce266796a2b9: Pull complete
381e243bf02b: Pull complete
b49af1a534ea: Pull complete
c4e0a943b22c: Pull complete
b662802b398a: Pull complete
02559cd4bc8d: Pull complete
3eb37f00800a: Pull complete
579eb022ae11: Pull complete
72488ede6291: Pull complete
6d3463095547: Pull complete
2211e62042c6: Pull complete
4f6ffa539847: Pull complete
b3aea135f215: Pull complete
54902e72b210: Pull complete
Digest: sha256:707a1eb2c734642a8c6c1e1681caa240f59a16a85be690e1bc4d792d94824c6f
Status: Downloaded newer image for nvcr.io/nvidia/ai-dynamo/vllm-runtime:0.9.0
57a57c217b349b9b1910c1967dcd139dec36255867925995aada14514b4f3d85
root@phase3-nvidia:~/dynamo#
```

Container Runtime Parameters

- **-d:** Runs the container in background (daemon mode)
- **--name dynamo-llm-disaggregated:** Assigns a name for easier management
- **--restart always:** Ensures automatic restart on failure or system reboot
- **--gpus '"device=0,1,2,3"':** Allocates GPUs 0–3 for all workers
- **--network host:** Required for worker discovery and NIXL communication
- **--ipc=host:** Enables shared memory for inter-process communication
- **--ulimit memlock=-1:** Removes memory lock limits for GPU workloads
- **--ulimit stack=67108864:** Increases stack size for large model execution
- **-e HF_TOKEN:** Provides Hugging Face authentication
- **-v $(pwd)/container/.cache:** Persists model and runtime cache
- **-u 1000:1000:** Runs container with correct user permissions

Expected Output:

- Container ID (64-character hex string)

Monitor Disaggregated Workers Initialization

Initialization includes Dynamo frontend startup, decode and prefill worker launch, and establishment of NIXL-based KV cache communication. Monitoring container logs provides visibility into each stage of the startup process.

View Logs in Real Time

```
docker logs -f dynamo-llm-disaggregated
```

```
root@phase3-nvidia: ~/dynamo# docker logs -f dynamo-llm-disaggregated

==========
== CUDA ==
==========

CUDA Version 12.9.1

Container image Copyright (c) 2016-2023, NVIDIA CORPORATION & AFFILIATES. All rights reserved.

This container image and its contents are governed by the NVIDIA Deep Learning Container License.
By pulling and using the container, you accept the terms and conditions of this license:
https://developer.nvidia.com/ngc/nvidia-deep-learning-container-license

A copy of this license is made available in this container at /NGC-DL-CONTAINER-LICENSE for your
convenience.

Checking port availability...
Starting Dynamo Frontend...
2026-03-11T06:37:08.832465Z  INFO dynamo_runtime::distributed: Initializing KV store discovery
backend
2026-03-11T06:37:08.832586Z  INFO dynamo_runtime::pipeline::network::manager: Initializing
NetworkManager with TCP request plane mode=tcp host=192.0.2.101 port=OS-assigned
2026-03-11T06:37:08.845289Z  INFO dynamo_llm::http::service::service_v2: Starting HTTP(S) service
protocol="HTTP" address="0.0.0.0:8000"
Starting DISAGGREGATED mode deployment with model: nvidia/Llama-3_3-Nemotron-Super-49B-v1_5
Starting Decode worker (GPUs 0-1, TP=2)...
[2026-03-11 06:37:24] INFO __init__.py:53: dynamo.nixl_connect: Utilizing CuPy to enable GPU
acceleration.
[2026-03-11 06:37:25] INFO encode_worker_handler.py:39: Using cupy for array operations (GPU
mode).
2026-03-11T06:37:25.636554Z  INFO args.create_kv_transfer_config: Creating kv_transfer_config from
--connector [nixl]
2026-03-11T06:37:25.636662Z  INFO args.create_kv_events_config: Decode worker detected
(is_decode_worker=True): kv_events_config disabled (decode workers don't publish KV events)
2026-03-11T06:37:25.636684Z  INFO args.overwrite_args: Using kv_events_config for publishing vLLM
KV events over zmq: None (use_kv_events=False)
2026-03-11T06:37:25.652881Z  INFO dynamo_runtime::distributed: Initializing KV store discovery
backend
2026-03-11T06:37:25.652984Z  INFO dynamo_runtime::pipeline::network::manager: Initializing
NetworkManager with TCP request plane mode=tcp host=192.0.2.101 port=OS-assigned
```

```
2026-03-11T06:38:08.765062Z  WARN multiproc_executor.set_multiprocessing_worker_envs: Reducing
Torch parallelism from 112 threads to 1 to avoid unnecessary CPU contention. Set OMP_NUM_THREADS
in the external environment to tune this value as needed.
Loading safetensors checkpoint shards: 33% Completed | 7/21 [00:05<00:11, 1.25it/s]
Loading safetensors checkpoint shards: 38% Completed | 8/21 [00:06<00:10, 1.22it/s]
Loading safetensors checkpoint shards: 43% Completed | 9/21 [00:07<00:09, 1.22it/s]
Loading safetensors checkpoint shards: 48% Completed | 10/21 [00:08<00:09, 1.21it/s]
Loading safetensors checkpoint shards: 52% Completed | 11/21 [00:08<00:07, 1.26it/s]
Loading safetensors checkpoint shards: 57% Completed | 12/21 [00:09<00:07, 1.25it/s]
Loading safetensors checkpoint shards: 62% Completed | 13/21 [00:10<00:05, 1.46it/s]
Loading safetensors checkpoint shards: 67% Completed | 14/21 [00:10<00:04, 1.47it/s]
Loading safetensors checkpoint shards: 71% Completed | 15/21 [00:11<00:04, 1.38it/s]
2026-03-11T06:38:15.541470Z  INFO parallel_state.init_distributed_environment: world_size=2 rank=0
local_rank=0 distributed_init_method=tcp://127.0.0.1:36059 backend=nccl
Loading safetensors checkpoint shards: 76% Completed | 16/21 [00:12<00:03, 1.34it/s]
Loading safetensors checkpoint shards: 81% Completed | 17/21 [00:13<00:03, 1.29it/s]
Loading safetensors checkpoint shards: 86% Completed | 18/21 [00:14<00:02, 1.25it/s]
Loading safetensors checkpoint shards: 90% Completed | 19/21 [00:14<00:01, 1.24it/s]
Loading safetensors checkpoint shards: 95% Completed | 20/21 [00:15<00:00, 1.23it/s]
2026-03-11T06:40:35.936962Z  INFO http-request: dynamo_runtime::discovery::kv_store:
KVStoreDiscovery::list: query=AllEndpoints, prefix=v1/instances, bucket=v1/instances, entries=5
method=GET uri=/health version=HTTP/1.1
2026-03-11T06:40:37.946681Z  INFO http-request: dynamo_runtime::discovery::kv_store:
KVStoreDiscovery::list: query=AllEndpoints, prefix=v1/instances, bucket=v1/instances, entries=5
method=GET uri=/health version=HTTP/1.1
2026-03-11T06:40:39.956973Z  INFO http-request: dynamo_runtime::discovery::kv_store:
KVStoreDiscovery::list: query=AllEndpoints, prefix=v1/instances, bucket=v1/instances, entries=5
method=GET uri=/health version=HTTP/1.1
2026-03-11T06:40:41.967137Z  INFO http-request: dynamo_runtime::discovery::kv_store:
KVStoreDiscovery::list: query=AllEndpoints, prefix=v1/instances, bucket=v1/instances, entries=5
method=GET uri=/health version=HTTP/1.1

WARNING: prefill worker registration timeout after 60 seconds
Prefill worker may need more time to register
Performing final health check...

2026-03-11T06:40:43.976374Z  INFO http-request: dynamo_runtime::discovery::kv_store:
KVStoreDiscovery::list: query=AllEndpoints, prefix=v1/instances, bucket=v1/instances, entries=5
method=GET uri=/health version=HTTP/1.1

✓ Deployment ready! Frontend is responding on port 8000
Process IDs: Frontend=96, Decode=632, Prefill=1500
Monitor logs with: docker logs -f dynamo-llm-disaggregated
docker logs -f dynamo-llm-disaggregateddocker logs -f dynamo-llm-disaggregated
```

Expected Output:

- Log output showing frontend startup, worker initialization, and service readiness

Press Ctrl + C to exit the log stream (container continues running)

Key Log Indicators

- Starting Dynamo Frontend... - Frontend initialization
- Starting Decode Worker (GPUs 0-1, TP=2)... - Decode worker startup
- Starting Prefill Worker (GPUs 2-3, TP=2)... - Prefill worker startup
- All services starting... waiting for initialization... - Initialization delay before readiness
- Chat completions is ready - Chat endpoint ready
- Completions is ready - Completions endpoint ready
- added model - Model successfully registered
- "☑ Deployment ready! Frontend is responding on port 8000": All workers initialized and ready

Filter Key Logs Only

Disaggregated mode produces complex logs from frontend, prefill workers, decode workers, and NIXL connector initialization. This filtered command cuts through the noise to show only the deployment milestones that matter for verification.

```
docker logs dynamo-llm-disaggregated 2>&1 | grep -iE "Starting Dynamo Frontend|Starting Decode Worker|Starting Prefill Worker|All services starting|Chat completions is ready|Completions is ready|added model|Deployment ready"
```

```
root@phase3-nvidia:~/dynamo# docker logs dynamo-llm-disaggregated 2>&1 | grep -iE "Starting Dynamo Frontend|Starting Decode Worker|Starting Prefill Worker|All services starting|Chat completions is ready|Completions is ready|added model|Deployment ready"
Starting Dynamo Frontend...
Starting Decode Worker (GPUs 0-1, TP=2)...
Starting Prefill Worker (GPUs 2-3, TP=2)...
All services starting... waiting for initialization...
2026-03-13T10:35:21.515362Z  INFO dynamo_llm::discovery::watcher: Chat completions is ready
2026-03-13T10:35:21.530385Z  INFO dynamo_llm::discovery::watcher: Completions is ready
2026-03-13T10:35:21.530403Z  INFO dynamo_llm::discovery::watcher: added model
model_name="nvidia/Llama-3.3-Nemotron-Super-49B-v1.5" namespace="dynamo"
✓ Deployment ready! Frontend is responding on port 8000
root@phase3-nvidia:~/dynamo#
```

Expected Output:

- Starting Dynamo Frontend...
- Starting Decode Worker (GPUs 0-1, TP=2)...
- Starting Prefill Worker (GPUs 2-3, TP=2)...
- All services starting... waiting for initialization...
- Chat completions is ready
- Completions is ready

- added model model_name="nvidia/Llama-3_3-Nemotron-Super-49B-v1_5" namespace="dynamo"
- ☑ Deployment ready! Frontend is responding on port 8000

Service Endpoint Information

In disaggregated mode, the Dynamo frontend exposes a single OpenAI-compatible API endpoint on port 8000. Client requests are handled by the frontend, which automatically routes prompt processing to prefill workers and token generation to decode workers. KV cache transfer between workers is managed through NIXL.

Endpoint Details

- **URL:** http://<NVIDIA_GPU_SERVER_IP>:8000/v1/chat/completions
- **Model:** nvidia/Llama-3_3-Nemotron-Super-49B-v1_5
- **Context Length:** 32K tokens (configurable up to 128K)
- **API Standard:** OpenAI-compatible
- **Authentication:** None (internal network deployment)
- **Architecture:**
  - Decode workers: GPUs 0-1 (TP=2)
  - Prefill workers: GPUs 2-3 (TP=2)

Example Python Integration

```
import httpx
response = httpx.post(
    "http://<NVIDIA_GPU_SERVER_IP>:8000/v1/chat/completions",
    json={
        "model": "nvidia/Llama-3_3-Nemotron-Super-49B-v1_5",
        "messages": [
            {"role": "system", "content": "You are a helpful assistant."},
            {"role": "user", "content": "Explain disaggregated inference."}
        ],
        "max_tokens": 500,
        "temperature": 0.7
    }
)
result = response.json()

answer = result["choices"][0]["message"]["content"]

print(answer)
```

Example cURL Request

```
curl -X POST http://<NVIDIA_GPU_SERVER_IP>:8000/v1/chat/completions \
  -H "Content-Type: application/json" \
  -d '{
    "model": "nvidia/Llama-3_3-Nemotron-Super-49B-v1_5",
    "messages": [{"role": "user", "content": "What is AI?"}],
    "max_tokens": 200,
    "temperature": 0.7
  }'
```

```
root@phase3-nvidia:~/dynamo# curl -X POST http://192.0.2.101:8000/v1/chat/completions \
-H "Content-Type: application/json" \
-d '{
  "model": "nvidia/Llama-3.3-Nemotron-Super-49B-v1.5",
  "messages": [{"role": "user", "content": "What is AI?"}],
  "max_tokens": 200,
  "temperature": 0.7
}'
{"id":"chatcmpl-a0a236df-3bc9-4cfb-bb27-
44b52fb0f815","choices":[{"index":0,"message":{"content":"<think>\nOkay, the user is asking,
\"What is AI?\" Let me start by breaking down the term. AI stands for Artificial Intelligence,
right? But I need to explain it in a way that's easy to understand. Maybe start with the basic
definition. AI refers to machines or software that can perform tasks that usually require human
intelligence. But what exactly does that include?\n\nI should mention the different types of AI.
There's the distinction between narrow AI and general AI. Narrow AI is what we see today, like
voice assistants or recommendation systems. General AI is more theoretical, having human-like
versatility. I should explain that difference clearly.\n\nAlso, key concepts like machine
learning, deep learning, and natural language processing are important. Maybe give examples of
each. Machine learning uses algorithms to learn from data, like spam filters. Deep learning uses
neural networks, similar to the human brain, for image recognition. NLP allows machines to
understand and generate human language, like chatbots.\n\nApplications
are\"role\":\"assistant\",\"reasoning_content\":null},"finish_reason":"length"}],"created":1731732
141,"model":"nvidia/Llama-3.3-Nemotron-Super-49B-
v1.5","object":"chat.completion","usage":{"prompt_tokens":19,"completion_tokens":200,"total_tokens
":219}}
root@phase3-nvidia:~/dynamo#
```

Note: Replace <NVIDIA_GPU_SERVER_IP> with the internal IP address of the GPU server.

Verify Disaggregated Workers

Validates that the container is running, decode and prefill workers are initialized, GPU allocation is correct, and the disaggregated workflow is functioning as expected.

Check Container Status

```
docker ps | grep dynamo-llm-disaggregated
```

```
root@phase3-nvidia:~/dynamo# docker ps | grep dynamo-llm-disaggregated
cf99483414b0   nvcr.io/nvidia/ai-dynamo/vllm-runtime:0.9.0-cuda13   "/opt/nvidia/nvidia_…"   5
minutes ago   Up 5 minutes   dynamo-llm-disaggregated
```



```
root@phase3-nvidia:~/dynamo#
```

Expected Output:

- Container shows Up status

Verify GPU Allocation Across Worker Types

```
nvidia-smi
```

```
root@phase3-nvidia:~/dynamo# nvidia-smi
Tue Mar 10 15:11:11 2026
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 580.95.05              Driver Version: 580.95.05      CUDA Version: 13.0     |
+-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA A100-SXM4-80GB      On      | 00000000:1C:00.0 Off |                    Off |
| N/A   39C    P0             74W / 500W  | 74349MiB / 81920MiB    | 0%           Default |
|                                         |                        |             Disabled |
+-----------------------------------------+------------------------+----------------------+
|   1  NVIDIA A100-SXM4-80GB      On      | 00000000:3E:00.0 Off |                    Off |
| N/A   38C    P0             75W / 500W  | 74349MiB / 81920MiB    | 0%           Default |
|                                         |                        |             Disabled |
+-----------------------------------------+------------------------+----------------------+
|   2  NVIDIA A100-SXM4-80GB      On      | 00000000:4F:00.0 Off |                    Off |
| N/A   37C    P0             66W / 500W  | 74317MiB / 81920MiB    | 0%           Default |
|                                         |                        |             Disabled |
+-----------------------------------------+------------------------+----------------------+
|   3  NVIDIA A100-SXM4-80GB      On      | 00000000:60:00.0 Off |                    Off |
| N/A   38C    P0             71W / 500W  | 74317MiB / 81920MiB    | 0%           Default |
|                                         |                        |             Disabled |
+-----------------------------------------+------------------------+----------------------+
|   4  NVIDIA A100-SXM4-80GB      On      | 00000000:9E:00.0 Off |                    Off |
| N/A   39C    P0             68W / 500W  |     0MiB / 81920MiB    | 0%           Default |
|                                         |                        |             Disabled |
+-----------------------------------------+------------------------+----------------------+
|   5  NVIDIA A100-SXM4-80GB      On      | 00000000:BE:00.0 Off |                    Off |
| N/A   38C    P0             70W / 500W  |     0MiB / 81920MiB    | 0%           Default |
|                                         |                        |             Disabled |
+-----------------------------------------+------------------------+----------------------+
|   6  NVIDIA A100-SXM4-80GB      On      | 00000000:CE:00.0 Off |                    Off |
| N/A   39C    P0             66W / 500W  |     0MiB / 81920MiB    | 0%           Default |
|                                         |                        |             Disabled |
+-----------------------------------------+------------------------+----------------------+
|   7  NVIDIA A100-SXM4-80GB      On      | 00000000:DE:00.0 Off |                    Off |
| N/A   39C    P0             66W / 500W  |     0MiB / 81920MiB    | 0%           Default |
|                                         |                        |             Disabled |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                        GPU Memory       |
```

```
|        ID   ID                                                  Usage            |
|=============================================================================|
|    0   N/A  N/A    1517704      C   VLLM::Worker_TP0                74340MiB         |
|    1   N/A  N/A    1517721      C   VLLM::Worker_TP1                74340MiB         |
|    2   N/A  N/A    1517712      C   VLLM::Worker_TP0                74308MiB         |
|    3   N/A  N/A    1517724      C   VLLM::Worker_TP1                74308MiB         |
+-----------------------------------------------------------------------------+
```

Expected Output:

- GPUs 0-1: Active processes from decode workers with corresponding memory usage
- GPUs 2-3: Active processes from prefill workers with corresponding memory usage
- Confirms both worker types are loaded and ready for disaggregated inference

Test Disaggregated Workflow Functionality

```
MODEL_NAME="nvidia/Llama-3_3-Nemotron-Super-49B-v1_5"

# Basic endpoint test to confirm disaggregated architecture is working
curl -s http://localhost:8000/v1/chat/completions -H "Content-Type: application/json" -d "{
  \"model\": \"$MODEL_NAME\",
  \"messages\": [{\"role\": \"user\", \"content\": \"Test disaggregated inference\"}],
  \"max_tokens\": 50
}" | jq '.choices[0].message.content'
```

```
root@phase3-nvidia:~/dynamo# MODEL_NAME="nvidia/Llama-3.3-Nemotron-Super-49B-v1.5"

# Basic endpoint test to confirm disaggregated architecture is working
curl -s http://localhost:8000/v1/chat/completions -H "Content-Type: application/json" -d '{
  "model": "'"$MODEL_NAME"'",
  "messages": [{"role": "user", "content": "Test disaggregated inference"}],
  "max_tokens": 50
}' | jq '.choices[0].message.content'
"<think>\nOkay, the user mentioned \"Test disaggregated inference.\" I need to figure out what
they're asking for. Let me start by breaking down the terms. \"Disaggregated inference\" probably
refers to the process of performing inference (like"
root@phase3-nvidia:~/dynamo#
```

Expected Output:

- Generated text is returned in the .choices[0].message.content field
- Response confirms the request was processed successfully through the disaggregated serving workflow

Check Worker and NIXL Connector Status in Logs

```
docker logs dynamo-llm-disaggregated 2>&1 | grep -E "Starting DISAGGREGATED mode|NIXL is
available|Decode worker detected|Chat completions is ready|Completions is ready|added model"
```

```
root@phase3-nvidia:~/dynamo# docker logs dynamo-llm-disaggregated 2>&1 | grep -E "Starting
DISAGGREGATED model|NIXL is available|Decode worker detected|Chat completions is ready|Completions
is ready|added model"
Starting DISAGGREGATED model deployment with model: nvidia/Llama-3.3-Nemotron-Super-49B-v1.5
2026-03-13T10:33:58.871522Z  INFO args.create_kv_events_config: Decode worker detected
(is_decode_worker=true): kv_events_config disabled (decode workers don’t publish KV events)
2026-03-13T10:34:58.490907Z  INFO nixl_connector: NIXL is available
2026-03-13T10:34:58.490931Z  INFO nixl_connector: NIXL is available
2026-03-13T10:35:10.244345Z  INFO nixl_connector: NIXL is available
2026-03-13T10:35:10 [nixl_connector.py:97] NIXL is available
2026-03-13T10:35:20.209382Z  INFO nixl_connector: NIXL is available
2026-03-13T10:35:20.209418Z  INFO nixl_connector: NIXL is available
2026-03-13T10:35:21.515362Z  INFO dynamo_llm::discovery::watcher: Chat completions is ready
2026-03-13T10:35:21.530385Z  INFO dynamo_llm::discovery::watcher: Completions is ready
2026-03-13T10:35:21.530403Z  INFO dynamo_llm::discovery::watcher: added model
model_name="nvidia/Llama-3.3-Nemotron-Super-49B-v1.5" namespace="dynamo"
2026-03-13T10:35:31.634320Z  INFO nixl_connector: NIXL is available
root@phase3-nvidia:~/dynamo#
```

Expected Output:

- Log entries confirming disaggregated startup, worker initialization, model registration, and service readiness

Test Concurrent Requests to Verify Disaggregated Processing

```
for i in {1..5}; do
  curl -s http://localhost:8000/v1/chat/completions -H "Content-Type: application/json" -d "{
    \"model\": \"$MODEL_NAME\",
    \"messages\": [{\"role\": \"user\", \"content\": \"Disaggregated test request $i with longer
prompt to utilize prefill workers effectively\"}],
    \"max_tokens\": 30
  }" &
done
wait
echo "All requests completed"
```

```
root@phase3-nvidia:~/dynamo# for i in {1..5}; do

curl -s http://localhost:8000/v1/chat/completions -H "Content-Type: application/json" -d "{

  \"model\": \"$MODEL_NAME\",

  \"messages\": [{\"role\": \"user\", \"content\": \"Disaggregated test request $i with longer
prompt to utilize prefill workers effectively\"}],

  \"max_tokens\": 30

}" &

done

wait
```

```
echo "All requests completed"

[1] 1744226

[2] 1744227

[3] 1744228

[4] 1744229

[5] 1744230

{"id":"chatcmpl-9d297dd5-cd1e-4e34-8254-
5361a3cb90dc","choices":[{"index":0,"message":{"content":"<think>\nOkay, the user is asking about
a \"disaggregated test request 1 with longer prompt to utilize prefill workers effectively.\"
Hmm...","role":"assistant","reasoning_content":null},"finish_reason":"length"}],"created":17734001
49,"model":"nvidia/Llama-3.3-Nemotron-Super-49B-
v1.5","object":"chat.completion","usage":{"prompt_tokens":31,"completion_tokens":30,"total_tokens"
:61}}[1]  Done                    curl -s http://localhost:8000/v1/chat/completions -H "Content-
Type: application/json" -d "{

  \"model\": \"$MODEL_NAME\",

  \"messages\": [{\"role\": \"user\", \"content\": \"Disaggregated test request $i with longer
prompt to utilize prefill workers effectively\"}],

  \"max_tokens\": 30

}"

{"id":"chatcmpl-924ae3ff-5901-4ce4-b0d9-
1747d9d5c975","choices":[{"index":0,"message":{"content":"<think>\nOkay, the user is asking about
a \"disaggregated test request 3 with a longer prompt to utilize prefill workers effectively.\"
Hmm...","role":"assistant","reasoning_content":null},"finish_reason":"length"}],"created":17734001
49,"model":"nvidia/Llama-3.3-Nemotron-Super-49B-
v1.5","object":"chat.completion","usage":{"prompt_tokens":31,"completion_tokens":30,"total_tokens"
:61,"prompt_tokens_details":{"audio_tokens":null,"cached_tokens":163}}}[2]  Done
curl -s http://localhost:8000/v1/chat/completions -H "Content-Type: application/json" -d "{

  \"model\": \"$MODEL_NAME\",

  \"messages\": [{\"role\": \"user\", \"content\": \"Disaggregated test request $i with longer
prompt to utilize prefill workers effectively\"}],

  \"max_tokens\": 30

}"

{"id":"chatcmpl-35a716be-ac41-49fe-891f-
01788d56628a","choices":[{"index":0,"message":{"content":"<think>\nOkay, the user is asking about
\"Disaggregated test request 2 with longer prompt to utilize prefill workers effectively.\"
Hmm...","role":"assistant","reasoning_content":null},"finish_reason":"length"}],"created":17734001
49,"model":"nvidia/Llama-3.3-Nemotron-Super-49B-
v1.5","object":"chat.completion","usage":{"prompt_tokens":31,"completion_tokens":30,"total_tokens"
:61,"prompt_tokens_details":{"audio_tokens":null,"cached_tokens":163}}}[3]  Done
curl -s http://localhost:8000/v1/chat/completions -H "Content-Type: application/json" -d "{

  \"model\": \"$MODEL_NAME\",

  \"messages\": [{\"role\": \"user\", \"content\": \"Disaggregated test request $i with longer
prompt to utilize prefill workers effectively\"}],

  \"max_tokens\": 30

}"
```



```
{"id":"chatcmpl-7e1d24e1-e9bc-4a8e-b306-
560675ee86ba","choices":[{"index":0,"message":{"content":"<think>\nOkay, let's tackle this user's
request. They mentioned \"Disaggregated test request 4 with longer prompt to utilize prefill
workers
effectively.\"","role":"assistant","reasoning_content":null},"finish_reason":"length"}],"created":
1773400149,"model":"nvidia/Llama-3.3-Nemotron-Super-49B-
v1.5","object":"chat.completion","usage":{"prompt_tokens":31,"completion_tokens":30,"total_tokens"
:61,"prompt_tokens_details":{"audio_tokens":null,"cached_tokens":163}}}[4]  Done
curl -s http://localhost:8000/v1/chat/completions -H "Content-Type: application/json" -d "{

  \"model\": \"$MODEL_NAME\",

  \"messages\": [{\"role\": \"user\", \"content\": \"Disaggregated test request $i with longer
prompt to utilize prefill workers effectively\"}],

  \"max_tokens\": 30

}"

{"id":"chatcmpl-f89affc2-ecfd-4dfc-b927-
0a810644382a","choices":[{"index":0,"message":{"content":"<think>\nOkay, the user is asking about
a \"disaggregated test request 5 with longer prompt to utilize prefill workers
effectively.\"","role":"assistant","reasoning_content":null},"finish_reason":"length"}],"created":
1773400149,"model":"nvidia/Llama-3.3-Nemotron-Super-49B-
v1.5","object":"chat.completion","usage":{"prompt_tokens":31,"completion_tokens":30,"total_tokens"
:61,"prompt_tokens_details":{"audio_tokens":null,"cached_tokens":163}}}[5]+  Done
curl -s http://localhost:8000/v1/chat/completions -H "Content-Type: application/json" -d "{

  \"model\": \"$MODEL_NAME\",

  \"messages\": [{\"role\": \"user\", \"content\": \"Disaggregated test request $i with longer
prompt to utilize prefill workers effectively\"}],

  \"max_tokens\": 30

}"

All requests completed

root@phase3-nvidia:~/dynamo
```

Expected Output:

- Multiple successful JSON responses are returned
- Confirms requests are processed through the disaggregated serving pipeline
- Prefill and decode workers remain active during concurrent request handling

Deploy Embedding Model using vLLM

Embedding models convert text into dense vector representations that capture semantic meaning. These embeddings are essential for RAG systems, enabling semantic search, document retrieval, and similarity matching. The BAAI/bge-m3 model used in this deployment generates high-quality multilingual embeddings suitable for large-context inputs.

In this architecture, embedding inference is deployed using standalone vLLM rather than NVIDIA Dynamo. Dynamo's backends are purpose-built for generative LLM inference, supporting features such as disaggregated serving, KV cache routing, and prefill/decode separation. These capabilities are specific to autoregressive token generation and are not applicable to embedding or reranking workloads.

Standalone vLLM serve is therefore used for embedding, reranking, and document parsing tasks, as it provides efficient, stateless inference without the orchestration overhead required for generative LLM pipelines. This separation ensures that each component in the system is aligned with its intended workload.

Service Architecture (Embedding & Other Auxiliary Models)

- Port 8000: Dynamo Frontend → LLM Worker (GPUs 0-3)
- Port 8001: vLLM Serve → Embedding Model (GPU 4)
- Port 8002: vLLM Serve → Reranker Model (GPU 5)
- Port 8003: vLLM Serve → Document Parser Model (GPU 6)

Deployment Characteristics

- Single-GPU deployment (sufficient for embedding workloads)
- Simpler architecture reduces operational complexity
- Direct vLLM deployment provides lower latency for embedding requests
- OpenAI-compatible embeddings endpoint for RAG pipeline integration

Setup vLLM Cache Directory

Standalone vLLM services (embedding, reranker, and document parser) require write access to cache directories for storing model weights and runtime artifacts. A shared cache directory is used across all services to reduce duplication and simplify management.

Prepare Cache Directory Structure

Ensure the working directory is the **home** directory:

```
cd
```

```
root@phase3-nvidia:~/dynamo# cd
root@phase3-nvidia:~#
```

```
mkdir -p ~/vllm-models/.cache/huggingface
mkdir -p ~/vllm-models/.cache/vllm
mkdir -p ~/vllm-models/.cache/flashinfer
```

```
root@phase3-nvidia:~# mkdir -p ~/vllm-models/.cache/huggingface
root@phase3-nvidia:~# mkdir -p ~/vllm-models/.cache/vllm
root@phase3-nvidia:~# mkdir -p ~/vllm-models/.cache/flashinfer
root@phase3-nvidia:~#
```

Set Directory Ownership

Assign ownership of the cache directory to the container user (UID 1000) to ensure write access during model download and runtime operations.

```
sudo chown -R 1000:1000 ~/vllm-models/.cache
```

```
root@phase3-nvidia:~# sudo chown -R 1000:1000 ~/vllm-models/.cache
root@phase3-nvidia:~#
```

Configure Directory Permissions

Apply appropriate permissions to allow read/write access for the container user while maintaining controlled access for others.

```
sudo chmod -R 775 ~/vllm-models/.cache
```

```
root@phase3-nvidia:~# sudo chmod -R 775 ~/vllm-models/.cache
root@phase3-nvidia:~#
```

Verify Directory Permissions

```
ls -la ~/vllm-models/.cache/
```

```
root@phase3-nvidia:~# ls -la ~/vllm-models/.cache/
total 20
drwxr-xr-x 5 linuxuser linuxuser 4096 Mar 10 15:52 .
drwxr-xr-x 3 root      root      4096 Mar  4 22:02 ..
drwxrwxr-x 2 linuxuser linuxuser 4096 Mar 10 15:52 flashinfer
drwxrwxr-x 2 linuxuser linuxuser 4096 Mar 10 15:52 huggingface
drwxrwxr-x 2 linuxuser linuxuser 4096 Mar 10 15:52 vllm
root@phase3-nvidia:~#
```

Expected Output:

- huggingface/, vllm/, and flashinfer/ directories are present
- Ownership is assigned to UID 1000 (may appear as a username depending on system)
- Permissions reflect read/write access for owner and group

Deploy Embedding Worker Container

The embedding model is deployed as an independent vLLM container with dedicated GPU allocation, Hugging Face authentication, and a shared cache directory. On first launch, the container automatically retrieves model weights and initializes the vLLM runtime configured for embedding generation.

Model Selection: BAAI/bge-m3

The **BAAI/bge-m3** model is selected for embedding generation due to its performance and compatibility with vLLM.

- Standard BERT-based architecture (no custom implementation required)
- Generates 1024-dimensional embeddings
- Supports up to 8192 token context length
- Production-ready and widely adopted
- Natively supported by vLLM pooling runner

Set vLLM Runtime Image

Ensure the Dynamo-compatible vLLM runtime image is defined, if not done already:

```
export DYNAMO_IMAGE=nvcr.io/nvidia/ai-dynamo/vllm-runtime:0.9.0-cuda13
```

```
root@phase3-nvidia:~# export DYNAMO_IMAGE=nvcr.io/nvidia/ai-dynamo/vllm-runtime:0.9.0-cuda13
root@phase3-nvidia:~#
```

Note on Container Image Choice: We use the same **DYNAMO_IMAGE (NVIDIA Dynamo vLLM runtime)** for standalone services to ensure version consistency across all deployments. This guarantees that the LLM, embedding, reranker, and parser models all use compatible vLLM versions, preventing potential compatibility issues between different vLLM releases.

Launch Embedding Worker

Deploy the embedding service as a standalone vLLM container:

```
docker run -d --name vllm-embedding --restart always --gpus '"device=4"' --network host --ipc=host
--ulimit memlock=-1 --ulimit stack=67108864 -e HF_TOKEN=$HF_TOKEN -v ~/vllm-
models/.cache:/root/.cache/huggingface $DYNAMO_IMAGE vllm serve BAAI/bge-m3 --runner pooling --
dtype float16 --max-model-len 8192 --port 8001 --enforce-eager --trust-remote-code
```

```
root@phase3-nvidia:~# docker run -d --name vllm-embedding --restart always --gpus
'"device=4"' --network host --ipc=host --ulimit memlock=-1 --ulimit stack=67108864 -e
HF_TOKEN=$HF_TOKEN -v ~/vllm-models/.cache:/root/.cache/huggingface $DYNAMO_IMAGE vllm
serve BAAI/bge-m3 --runner pooling --dtype float16 --max-model-len 8192 --port 8001 --
enforce-eager --trust-remote-code

de6fac87337bca49efcc9ca7cac7e113b2705dacebf2deae772ffd29dfaf5d2f

root@phase3-nvidia:~#
```

Container Runtime Parameters

- **--name vllm-embedding:** Defines the container name for management
- **--restart always:** Ensures automatic restart on failure or system reboot
- **--gpus '"device=4"':** Allocates GPU 4 for embedding workloads
- **--network host:** Enables host networking without port mapping
- **--ipc=host:** Provides shared memory for GPU operations
- **--ulimit memlock=-1:** Removes memory lock limitations for GPU workloads
- **--ulimit stack=67108864:** Increases stack size for deep model execution
- **-e HF_TOKEN:** Passes Hugging Face token for model download
- **-v ~/vllm-models/.cache:** Mounts shared cache directory for model reuse
- **$DYNAMO_IMAGE:** Uses the same vLLM runtime image as Dynamo
- **vllm serve:** Invokes native vLLM serving interface
- **--runner pooling:** Enables embedding mode instead of text generation
- **--dtype float16:** Optimizes memory usage and performance
- **--max-model-len 8192:** Sets maximum supported context length
- **--port 8001:** Exposes embedding service on port 8001

- --enforce-eager: Disables CUDA graph execution for compatibility
- --trust-remote-code: Allows execution of model-specific code

Expected Output:

- A container ID (64-character hexadecimal string) indicating successful deployment.

Monitor Embedding Model Initialization

Model initialization includes downloading model weights (first run only), loading the model into GPU memory, and starting the vLLM API server. Monitoring container logs provides visibility into each stage and helps identify initialization issues.

Monitor Container Logs

```
docker logs -f vllm-embedding
root@phase3-nvidia:~# docker logs -f vllm-embedding

==========
== CUDA ==
==========

CUDA Version 13.0.2

Container image Copyright (c) 2016-2023, NVIDIA CORPORATION & AFFILIATES. All rights reserved.

This container image and its contents are governed by the NVIDIA Deep Learning Container License.
By pulling and using the container, you accept the terms and conditions of this license:
https://developer.nvidia.com/ngc/nvidia-deep-learning-container-license

A copy of this license is made available in this container at /NGC-DL-CONTAINER-LICENSE for your
convenience.

[2026-03-10 16:06:45] INFO font_manager.py:1639: generated new fontManager
(APIServer pid=1) INFO 03-10 16:06:46 [api_server.py:1272] vLLM API server version 0.14.1
(APIServer pid=1) INFO 03-10 16:06:46 [utils.py:263] non-default args: {'model_tag': 'BAAI/bge-
m3', 'port': 8001, 'model': 'BAAI/bge-m3', 'runner': 'pooling', 'trust_remote_code': True,
'dtype': 'float16', 'max_model_len': 8192, 'enforce_eager': True}
(APIServer pid=1) The argument `trust_remote_code` is to be used with Auto classes. It has no
effect here and is ignored.
(APIServer pid=1) The argument `trust_remote_code` is to be used with Auto classes. It has no
effect here and is ignored.
(APIServer pid=1) INFO 03-10 16:06:52 [config.py:889] Found sentence-transformers tokenize
configuration
(APIServer pid=1) INFO 03-10 16:06:58 [model.py:530] Resolved architecture: XLMRobertaModel
(APIServer pid=1) INFO 03-10 16:06:58 [config.py:775] Found sentence-transformers modules
configuration
(APIServer pid=1) INFO 03-10 16:06:58 [config.py:802] Found pooling configuration.
(APIServer pid=1) INFO 03-10 16:06:58 [model.py:1866] Downcasting torch.float32 to torch.float16.
(APIServer pid=1) INFO 03-10 16:06:58 [model.py:1545] Using max model len 8192
(APIServer pid=1) INFO 03-10 16:06:59 [vllm.py:630] Asynchronous scheduling is enabled.
```

```
(APIServer pid=1) INFO 03-10 16:06:59 [vllm.py:637] Disabling NCCL for DP synchronization when
using a sync scheduling
(APIServer pid=1) WARNING 03-10 16:06:59 [vllm.py:665] Enforce eager set, overriding optimization
level to O0
(APIServer pid=1) INFO 03-10 16:06:59 [vllm.py:765] Cudagraph is disabled under eager mode
(EngineCore_DPO pid=511) INFO 03-10 16:07:18 [core.py:97] Initializing a V1 LLM engine (v0.14.1)
with config: model='BAAI/bge-m3', speculative_config=None, tokenizer='BAAI/bge-m3',
skip_tokenizer_init=False, tokenizer_mode=auto, revision=None, tokenizer_revision=None,
trust_remote_code=True, dtype=torch.float16, max_seq_len=8192, download_dir=None,
load_format=auto, tensor_parallel_size=1, pipeline_parallel_size=1, data_parallel_size=1,
disable_custom_all_reduce=False, quantization=None, enforce_eager=True, kv_cache_dtype=auto,
device_config=cuda, structured_outputs_config=StructuredOutputsConfig(backend='auto',
disable_fallback=False, disable_any_whitespace=False, disable_additional_properties=False,
reasoning_parser='', reasoning_parser_plugin='', enable_in_reasoning=False),
observability_config=ObservabilityConfig(show_hidden_metrics_for_version=None,
otlp_traces_endpoint=None, collect_detailed_traces=None), kv_cache_metrics=False,
kv_cache_metrics_sample_rate=0.0, cudagraph_metrics=False, enable_layerwise_nvtx_tracing=False,
enable_mfu_metrics=False, enable_mem_pool_metrics=False, enable_logging_iteration_details=False,
seed=0, served_model_name=BAAI/bge-m3, enable_prefill_cache=False, chunked_prefill_enabled=False,
pooler_config=PoolerConfig(pooling_type=None, normalize=None, dimensions=None,
enable_chunked_processing=False, max_embed_len=None, softmax=None, activation=None,
use_activation=True, logit_bias=None, step_tag_id=None, returned_token_ids=None),
compilation_config={'level': None, 'model': <CompilationMode.NONE: 0>, 'debug_dump_path': None,
'cache_dir': '', 'compile_cache_save_format': 'binary', 'backend': 'inductor', 'custom_ops':
['all'], 'splitting_ops': [], 'compile_mm_encoder': False, 'compile_sizes': [],
'compile_ranges_split_points': [8192], 'inductor_compile_config':
{'enable_auto_functionalized_v2': False}, 'combo_kernels': True, 'benchmark_combo_kernel': True},
cudagraph_mode=<CUDAGraphMode.NONE: 0>, cudagraph_num_of_warmups=0, cudagraph_capture_sizes=[],
cudagraph_copy_inputs=False, cudagraph_specialize_lora=False, use_inductor_graph_partition=False,
pass_config={'enable_norm_quant': False, 'fuse_acc_quant': False, 'fuse_attn_quant': False,
'eliminate_noops': False}, enable_sparsity=False, use_fused_gemm=False, fuse_gemm_comms=False,
fuse_allreduce_mms=False, enable_prefetch=False, max_cudagraph_capture_size=0,
dynamic_shapes_config={'type': <DynamicShapesType.BACKED: 'backed'>, 'evaluate_guards': False},
enable_spill=False, use_inductor=False, debug_disable_cuda_graphs=False,
cuda_graph_num_of_warmups=0, cudagraph_capture_sizes=None, cudagraph_copy_inputs=False,
cudagraph_specialize_lora=False, use_inductor_graph_partition=False,
pass_config={'enable_norm_quant': False, 'fuse_acc_quant': False, 'fuse_attn_quant': False,
'eliminate_noops': False}, enable_sparsity=False, use_fused_gemm=False, fuse_gemm_comms=False,
fuse_allreduce_mms=False, enable_prefetch=False, max_cudagraph_capture_size=0,
dynamic_shapes_config={'type': <DynamicShapesType.BACKED: 'backed'>, 'evaluate_guards': False},
enable_spill=False, use_inductor=False, debug_disable_cuda_graphs=False)
(EngineCore_DPO pid=511) INFO 03-10 16:07:19 [parallel_state.py:1214] world_size=1 rank=0
local_rank=0 distributed_init_method=tcp://192.0.2.101:57305 backend=nccl
(EngineCore_DPO pid=511) INFO 03-10 16:07:29 [parallel_state.py:1425] rank 0 in world size 1 is
assigned as DP rank 0, PP rank 0, TP rank 0, EP rank N/A
(EngineCore_DPO pid=511) INFO 03-10 16:07:30 [gpu_model_runner.py:3808] Starting to load model
BAAI/bge-m3...
(EngineCore_DPO pid=511) INFO 03-10 16:07:46 [cuda.py:351] Using FLASH_ATTN attention backend out
of potential backends: ('FLASH_ATTN', 'TRITON_ATTN', 'FLEX_ATTENTION')
(EngineCore_DPO pid=511) INFO 03-10 16:08:08 [weight_utils.py:510] Time spent downloading weights
for BAAI/bge-m3: 17.122018 seconds
Loading pt checkpoint shards:   0% Completed | 0/1 [00:00<?, ?it/s]
Loading pt checkpoint shards: 100% Completed | 1/1 [00:01<00:00,  1.32s/it]
(EngineCore_DPO pid=511) INFO 03-10 16:08:10 [default_loader.py:291] Loading weights took 1.60
seconds
```



```
(EngineCore_DPO pid=511) INFO 03-10 16:08:11 [gpu_model_runner.py:3905] Model loading took 1.06
GiB memory and 39.666561 seconds
(EngineCore_DPO pid=511) INFO 03-10 16:08:11 [core.py:273] init engine (profile, create kv cache,
warmup model) took 0.26 seconds
(EngineCore_DPO pid=511) INFO 03-10 16:08:12 [config.py:889] Found sentence-transformers tokenize
configuration
(EngineCore_DPO pid=511) INFO 03-10 16:08:12 [vllm.py:630] Asynchronous scheduling is enabled.
(EngineCore_DPO pid=511) WARNING 03-10 16:08:12 [vllm.py:672] Inductor compilation was disabled by
user settings, optimization settings that are only active during inductor compilation will be
ignored.
(EngineCore_DPO pid=511) INFO 03-10 16:08:12 [vllm.py:765] Cudagraph is disabled under eager mode
(APIServer pid=1) INFO 03-10 16:08:13 [api_server.py:1014] Supported tasks: ['token_embed',
'embed']
(APIServer pid=1) INFO 03-10 16:08:13 [api_server.py:1346] Starting vLLM API server on
http://0.0.0.0:8001
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:38] Available routes are:
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:46] Route: /openapi.json, Methods: HEAD, GET
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:46] Route: /docs, Methods: HEAD, GET
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:46] Route: /docs/oauth2-redirect, Methods:
HEAD, GET
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:46] Route: /redoc, Methods: HEAD, GET
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:46] Route: /scale_elastic_ep, Methods: POST
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:46] Route: /is_scaling_elastic_ep, Methods:
POST
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:46] Route: /tokenize, Methods: POST
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:46] Route: /detokenize, Methods: POST
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:46] Route: /inference/v1/generate, Methods:
POST
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:46] Route: /pause, Methods: POST
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:46] Route: /resume, Methods: POST
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:46] Route: /is_paused, Methods: GET
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:46] Route: /metrics, Methods: GET
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:46] Route: /health, Methods: GET
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:46] Route: /load, Methods: GET
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:46] Route: /v1/models, Methods: GET
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:46] Route: /version, Methods: GET
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:46] Route: /v1/responses, Methods: POST
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:46] Route: /v1/responses/{response_id},
Methods: GET
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:46] Route: /v1/responses/{response_id}/cancel,
Methods: POST
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:46] Route: /v1/messages, Methods: POST
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:46] Route: /v1/chat/completions, Methods: POST
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:46] Route: /v1/completions, Methods: POST
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:46] Route: /v1/audio/transcriptions, Methods:
POST
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:46] Route: /v1/audio/translations, Methods:
POST
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:46] Route: /ping, Methods: GET
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:46] Route: /ping, Methods: POST
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:46] Route: /invocations, Methods: POST
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:46] Route: /classify, Methods: POST
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:46] Route: /v1/embeddings, Methods: POST
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:46] Route: /score, Methods: POST
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:46] Route: /v1/score, Methods: POST
```

```
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:46] Route: /rerank, Methods: POST
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:46] Route: /v1/rerank, Methods: POST
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:46] Route: /v2/rerank, Methods: POST
(APIServer pid=1) INFO 03-10 16:08:13 [launcher.py:46] Route: /pooling, Methods: POST
(APIServer pid=1) INFO:     Started server process [1]
(APIServer pid=1) INFO:     Waiting for application startup.
(APIServer pid=1) INFO:     Application startup complete.
```

Key Log Indicators

- "vLLM API server version 0.14.1": API server initialization
- "Starting to load model BAAI/bge-m3...": Model loading begins
- "Loading weights took X.XX seconds": Model weights loaded into GPU memory
- "Model loading took X.XX GiB memory": Model fully loaded and ready
- "Started server process [1]": Server process initialized
- "Waiting for application startup.": Application initialization in progress
- "Application startup complete.": Embedding service ready to accept requests

Press Ctrl+C to exit the log stream. The container continues running in the background.

Expected Output:

Log sequence showing vLLM API server initialization, model loading progress, and service startup completion

View Filtered Log Indicators Only

Embedding model logs include detailed tokenizer loading, model architecture detection, and API server startup messages. This filtered view extracts only the key status updates needed to confirm successful deployment.

```
docker logs vllm-embedding 2>&1 | grep -E "vLLM API server version|Starting to load model|Loading weights took|Model loading took|Started server process|Waiting for application startup|Application startup complete|/v1/embeddings"
```

```
root@phase3-nvidia:~# docker logs vllm-embedding 2>&1 | grep -E "vLLM API server version|Starting to load model|Loading weights took|Model loading took|Started server process|Waiting for application startup|Application startup complete|/v1/embeddings"
(APIServer pid=1) INFO 03-13 11:14:01 [api_server.py:1272] vLLM API server version 0.14.1
(EngineCore_DPO pid=511) INFO 03-13 11:14:45 [gpu_model_runner.py:3808] Starting to load model BAAI/bge-m3...
(EngineCore_DPO pid=511) INFO 03-13 11:15:25 [default_loader.py:291] Loading weights took 1.59 seconds
(EngineCore_DPO pid=511) INFO 03-13 11:15:25 [gpu_model_runner.py:3905] Model loading took 1.06 GiB memory and 39.764120 seconds
(APIServer pid=1) INFO 03-13 11:15:27 [launcher.py:46] Route: /v1/embeddings, Methods: POST
```

```
(APIServer pid=1) INFO:     Started server process [1]
(APIServer pid=1) INFO:     Waiting for application startup.
(APIServer pid=1) INFO:     Application startup complete.
root@phase3-nvidia:~#
```

Expected Output:

- "vLLM API server version 0.14.1" - API server initialization
- "Starting to load model BAAI/bge-m3..." - Model loading begins
- "Loading weights took X.XX seconds" - Model weights loaded
- "Model loading took X.XX GiB memory and XX.XX seconds" - Model fully initialized
- "Started server process [1]" - Server ready
- "Waiting for application startup." - Initialization in progress
- "Application startup complete." - Service ready
- "Route: /v1/embeddings, Methods: POST" - Embedding endpoint available

Verify Container Status

```
docker ps | grep vllm-embedding
```

```
root@phase3-nvidia:~# docker ps | grep vllm-embedding
de6fac87337b   nvcr.io/nvidia/ai-dynamo/vllm-runtime:0.9.0-cuda13   "/opt/nvidia/nvidia_…"   3
minutes ago   Up 3 minutes   vllm-embedding
root@phase3-nvidia:~#
```

Expected Output:

- Container is running with status "**Up X minutes**", confirming successful deployment.

Service Endpoint Information

The embedding service exposes an OpenAI-compatible endpoint for converting text into vector representations. This endpoint is used during document ingestion and query encoding within the RAG pipeline.

Endpoint Details

- **URL:** http://<NVIDIA_GPU_SERVER_IP>:8001/v1/embeddings
- **Model:** BAAI/bge-m3
- **Dimensions:** 1024
- **Max Tokens:** 8192

- **API Standard:** OpenAI-compatible
- **Authentication:** None (internal network deployment)

Example Python Integration

```
import httpx
response = httpx.post(
    "http://<NVIDIA_GPU_SERVER_IP>:8001/v1/embeddings",
    json={
        "model": "BAAI/bge-m3",
        "input": ["Text to embed", "Another text to embed"]
    }
)
result = response.json()
embeddings = [item["embedding"] for item in result["data"]]
# Each embedding is a list of 1024 floats
```

Example cURL Request

```
curl -X POST http://<NVIDIA_GPU_SERVER_IP>:8001/v1/embeddings \
  -H "Content-Type: application/json" \
  -d '{
    "model": "BAAI/bge-m3",
    "input": ["Document 1", "Document 2", "Document 3"]
}'
```

```
root@phase3-nvidia:~# curl -X POST http://192.0.2.101:8001/v1/embeddings \
-H "Content-Type: application/json" \
-d '{
  "model": "BAAI/bge-m3",
  "input": ["Document 1", "Document 2", "Document 3"]
}'
{"id":"embd-8f1efa90f1566496","object":"list","created":1773400874,"model":"BAAI/bge-
m3","data":[{"index":0,"object":"embedding","embedding":[0.010384537279605865,0.028552992269396782
,-0.0495014190673828,-0.009407063014805317,-0.004887368530035019,-0.011864199303905972,-
0.0030781454406678677,0.028857892379164696,0.007712177932262421,0.03127915784716606,0.027458937013
1874,-0.00850133090572548,-0.0094249984249447262,-0.045519784009290314,0.003194724675267935,-
0.026346951723098755,0.036605942994356155,0.005766198039054871,-0.018204325810074806,-
0.002914485754445195,-0.027028491720557213,-0.030597615987062454, 0.05617335066199303,
0.03086664527654648,-0.0077525321394205 09,0.020571377369844735,0.018383679911494255,-
0.040390290319919586, -0.0012890995454879 76, -0.04207620769739151,0.02799699828028679,
0.005169849377125502,0.016159702092409134,-0.014061273075640202,-0.07149009046566662,
0.020571785047650337, -0.013370763510465622,-0.020751139149069786,0.004580226261168718,-
0.04548391327261925,0.032086245715618134,0.013083798818826653,-0.0030534842517226934,-
0.021504420787096024,-0.05567116290330887,-0.030561747029423714,-0.00699028865000725,
0.017970489188208744,0.022419121116399765,-0.021217456087470055,-0.05194061994552612,
0.00408539021605167,-0.09720932182733223,-0.01140684913348702,-0.03952939345839653,-0.009039389908631325,-
0.023262079805135727,0.043188195675611496, 0.007114273191988468,
0.061446327716111214,0.03292920440543595,0.02989814057946205,0.043259937316179276,0.010483181104063988,-
0.0013171234168112278,-0.017173046246170998,-0.03264241567373276,0.012801318662241101,-
0.02130713318317978,0.01624265336431563,-0.0803501307964325,0.02785351686179638,-
```

0.05000360682606697,0.00912906602025032,-0.013828113675117493,0.00861631142348051,0.02356697991490364,-0.038094568997621536,0.01879619061946869,0.0037529608234763145,-0.018249165266752243,-0.010312795639038086,-0.01190903689712286,0.035117313265800476,-0.02912692166864872]}]}

Note: Replace <NVIDIA_GPU_SERVER_IP> with the internal IP address of the GPU server hosting the embedding service.

Verify Embedding Model Endpoint

Endpoint verification confirms that the embedding service is responding correctly and generating vector outputs. This step validates both the model metadata endpoint and the embedding inference endpoint.

Check Available Models

```
curl http://localhost:8001/v1/models
```

```
root@phase3-nvidia:~ # curl http://localhost:8001/v1/models
{"object":"list","data":[{"id":"BAAI/bge-m3","object":"model","created":1773159572,"owned_by":"vllm","root":"BAAI/bge-m3","parent":null,"max_model_len":8192,"permission":[{"id":"modelperm-a667b9af075363","object":"model_permission","created":1773159572,"allow_create_engine":false,"allow_sampling":true,"allow_logprobs":true,"allow_search_indices":false,"allow_view":true,"allow_fine_tuning":false,"organization":"*","group":null,"is_blocking":false}]}]}
root@phase3-nvidia:~#
```

Expected Output:

- JSON response listing BAAI/bge-m3 with model metadata

Test Embedding Generation

```
curl -X POST http://localhost:8001/v1/embeddings \
  -H "Content-Type: application/json" \
  -d '{"model": "BAAI/bge-m3", "input": "test query for RAG"}' | jq
```

```
root@phase3-nvidia:~# curl -X POST http://localhost:8001/v1/embeddings \
-H "Content-Type: application/json" \
-d '{"model": "BAAI/bge-m3", "input": "test query for RAG"}' | jq
{
  "id": "embd-8e02b796bb97940a",
  "object": "list",
  "created": 1773159627,
  "model": "BAAI/bge-m3",
  "data": [
    {
      "index": 0,
      "object": "embedding",
```



```
      "embedding": [
        -0.0035039647482335567,
        0.011504187248647213,
        -0.00032656308030709624,
        -0.008644182235002518,
        -0.028655052185058594,
        0.001875732559710741,
        0.024676712229847908,
        0.006792512256652117,
        -0.0634334459900856,
        -0.015620028600969703,
        0.00896501634269528,
        0.05459676682949066,
        0.0027935467660427094
      ]
    }
  ],
  "usage": {
    "prompt_tokens": 8,
    "total_tokens": 8,
    "completion_tokens": 0,
    "prompt_tokens_details": null
  }
}
root@phase3-nvidia:~#
```


Expected Output:

- JSON response containing embedding vector in data[0].embedding (1024-dimensional array).

Deploy Reranker Model using vLLM

Reranker models improve retrieval quality in RAG systems by evaluating query-document pairs and assigning relevance scores. Unlike embedding models, rerankers directly compute semantic relevance, enabling more accurate ranking of retrieved results before passing them to the LLM.

The **BAAI/bge-reranker-v2-m3** model is deployed as a standalone vLLM service to provide efficient and scalable reranking for RAG pipelines.

Deployment Characteristics

- Single GPU deployment optimized for reranking workloads
- Cohere-compatible /rerank endpoint for standard integrations
- OpenAI-style /score endpoint for flexibility
- Supports batch processing of multiple documents per query

Deploy Reranker Worker Container

The reranker model is deployed as a standalone vLLM container using the shared model cache directory. On startup, the container downloads model weights (first run only), loads the model into GPU memory, and initializes the API server.

The container exposes both Cohere-compatible and OpenAI-style endpoints for flexibility in integration. vLLM automatically detects the BertForSequenceClassification architecture and enables reranking mode.

Verify Environment Variables

```
echo $HF_TOKEN
```

```
root@phase3-nvidia :~# echo $HF_TOKEN
hf_ xxxxxxxxxxxxxxxx
root@phase3-nvidia :~ #
```

```
echo $DYNAMO_IMAGE
```

```
root@phase3-nvidia :~# echo $DYNAMO_IMAGE
hf_ xxxxxxxxxxxxxxxx
root@phase3-nvidia :~ #
```

If DYNAMO_IMAGE is empty, set it using:

```
export DYNAMO_IMAGE=nvcr.io/nvidia/ai-dynamo/vllm-runtime:0.9.0-cuda13
root@phase3-nvidia :~# export DYNAMO_IMAGE=nvcr.io/nvidia/ai-dynamo/vllm-runtime:0.9.0-cuda13
root@phase3-nvidia :~#
```

Deploy Standalone Reranker Worker

```
docker run -d --name vllm-reranker --restart always --gpus '"device=5"' --network host --ipc=host
--ulimit memlock=-1 --ulimit stack=67108864 -e HF_TOKEN=$HF_TOKEN -v ~/vllm-
models/.cache:/root/.cache/huggingface $DYNAMO_IMAGE vllm serve BAAI/bge-reranker-v2-m3 --dtype
float16 --max-model-len 8192 --port 8002 --enforce-eager --trust-remote-code
```

```
root@phase3-nvidia :~# docker run -d --name vllm-reranker --restart always --gpus '"device=5"' --
network host --ipc=host --ulimit memlock=-1 --ulimit stack=67108864 -e HF_TOKEN=$HF_TOKEN -v
~/vllm-models/.cache:/root/.cache/huggingface $DYNAMO_IMAGE vllm serve BAAI/bge-reranker-v2-m3 --
dtype float16 --max-model-len 8192 --port 8002 --enforce-eager --trust-remote-code
f87ee00b18e4939aec66232d4384b869cfe4afef74239273d6dc29d555267528
root@phase3-nvidia :~#
```


Container Runtime Parameters

- **--name vllm-reranker:** Defines the container name for management
- **--restart always:** Ensures automatic restart on failure or system reboot
- **--gpus '"device=5"':** Allocates GPU 5 for reranking workloads
- **--network host:** Enables host networking without port mapping
- **--ipc=host:** Provides shared memory for GPU operations
- **--ulimit memlock=-1:** Removes memory lock limitations for GPU workloads
- **--ulimit stack=67108864:** Increases stack size for model execution
- **-e HF_TOKEN:** Passes Hugging Face token for model download
- **-v ~/vllm-models/.cache:** Mounts shared cache directory for model reuse
- **$DYNAMO_IMAGE:** Uses the same runtime image as other vLLM services
- **vllm serve:** Invokes native vLLM serving interface
- **--dtype float16:** Optimizes memory usage and performance
- **--max-model-len 8192:** Sets maximum supported input length
- **--port 8002:** Exposes reranker service on port 8002
- **--enforce-eager:** Disables CUDA graph execution for compatibility
- **--trust-remote-code:** Allows execution of model-specific code

Expected Output:

- Container ID (64-character hexadecimal string)

Monitor Reranker Model Initialization

Model initialization includes downloading weights (first run only), loading the model into GPU memory, and starting the API server. Monitoring logs confirms successful startup.

Monitor Container Logs

```
docker logs -f vllm-reranker
```

- Wait approximately 2–5 minutes for complete initialization.
- Look for the following INFO:
  - Started server process
  - Waiting for application startup.
  - Application startup complete.

- Press Ctrl+C to exit the log stream. The container continues running in the background.

```
root@phase3-nvidia:~# docker logs -f vllm-reranker

========
== CUDA ==
========

CUDA Version 13.0.2

Container image Copyright (c) 2016-2023, NVIDIA CORPORATION & AFFILIATES. All rights reserved.

This container image and its contents are governed by the NVIDIA Deep Learning Container License.
By pulling and using the container, you accept the terms and conditions of this license:
https://developer.nvidia.com/ngc/nvidia-deep-learning-container-license

A copy of this license is made available in this container at /NGC-DL-CONTAINER-LICENSE for your
convenience.

[2026-03-10 16:29:38] INFO font_manager.py:1639: generated new fontManager
(APIServer pid=1) INFO 03-10 16:29:39 [api_server.py:1272] vLLM API server version 0.14.1
(APIServer pid=1) INFO 03-10 16:29:39 [utils.py:263] non-default args: {'model_tag': 'BAAI/bge-
reranker-v2-m3', 'port': 8002, 'model': 'BAAI/bge-reranker-v2-m3', 'trust_remote_code': True,
'dtype': 'float16', 'max_model_len': 8192, 'enforce_eager': True}
(APIServer pid=1) The argument `trust_remote_code` is to be used with Auto classes. It has no
effect here and is ignored.
(APIServer pid=1) The argument `trust_remote_code` is to be used with Auto classes. It has no
effect here and is ignored.
(APIServer pid=1) INFO 03-10 16:29:51 [model.py:800] Resolved `--runner auto` to `--runner
pooling`. Pass `--runner` explicitly to silence this message.
(APIServer pid=1) INFO 03-10 16:29:51 [model.py:530] Resolved architecture:
XLMRobertaForSequenceClassification
(APIServer pid=1) INFO 03-10 16:29:51 [model.py:1866] Downcasting torch.float32 to torch.float16.
(APIServer pid=1) INFO 03-10 16:29:51 [model.py:1545] Using max model len 8192
(APIServer pid=1) INFO 03-10 16:29:51 [vllm.py:630] Asynchronous scheduling is enabled.
(APIServer pid=1) INFO 03-10 16:29:51 [vllm.py:637] Disabling NCCL for DP synchronization when
using async scheduling.
(APIServer pid=1) WARNING 03-10 16:29:51 [vllm.py:665] Enforce eager set, overriding optimization
level to -O0
(APIServer pid=1) INFO 03-10 16:29:51 [vllm.py:765] Cudagraph is disabled under eager mode
```

```
(EngineCore_DPO pid=511) INFO 03-10 16:30:10 [core.py:97] Initializing a V1 LLM engine (v0.14.1)
with config: model='BAAI/bge-reranker-v2-m3', speculative_config=None, tokenizer='BAAI/bge-
reranker-v2-m3', skip_tokenizer_init=False, tokenizer_mode=auto, revision=None,
tokenizer_revision=None, trust_remote_code=True, dtype=torch.float16, max_seq_len=8192,
download_dir=None, load_format=auto, tensor_parallel_size=1, pipeline_parallel_size=1,
data_parallel_size=1, disable_custom_all_reduce=False, quantization=None, enforce_eager=True,
enable_return_routed_experts=False, kv_cache_dtype=auto, device_config=cuda,
structured_outputs_config=StructuredOutputsConfig(backend='auto', disable_fallback=False,
disable_any_whitespace=False, disable_additional_properties=False, reasoning_parser='',
reasoning_parser_plugin='', enable_in_reasoning=False),
observability_config=ObservabilityConfig(show_hidden_metrics_for_version=None,
otlp_traces_endpoint=None, collect_detailed_traces=None, kv_cache_metrics=False,
kv_cache_metrics_sample=0.1, cudagraph_metrics=False, enable_layerwise_nvtx_tracing=False,
enable_mfu_metrics=False, enable_mm_processor_stats=False,
enable_logging_iteration_details=False), seed=0, served_model_name=BAAI/bge-reranker-v2-m3,
enable_prefix_caching=False, enable_chunked_prefill=False,
pooler_config=PoolerConfig(pooling_type=None, seq_pooling_type='CLS', tok_pooling_type='ALL',
normalize=None, dimensions=None, enable_chunked_processing=None, max_embed_len=None, softmax=None,
activation=None, use_activation=True, logit_bias=None, step_tag_id=None, returned_token_ids=None),
compilation_config={'level': None, 'mode': <CompilationMode.NONE: 0>}, debug_dump_path=None,
cache_dir='', compile_cache_save_format='binary', backend='inductor', custom_ops=['all'],
splitting_ops=[], compile_mm_encoder=False, compile_sizes=[], compile_ranges_split_points=[8192],
inductor_compile_config={'enable_auto_functionalized_v2': False}, combo_kernels=True,
benchmark_combo_kernel=True, inductor_passes={}, cudagraph_mode=<CUDAGraphMode.NONE: 0>,
cudagraph_num_of_warmups=0, cudagraph_capture_sizes=[], cudagraph_copy_inputs=False,
cudagraph_specialize_lora=True, use_inductor_graph_partition=False, pass_config={'use_norm_quant':
False, 'fuse_act_quant': False, 'fuse_attn_quant': False, 'eliminate_noops': False, 'enable_sp':
False, 'fuse_gemm_comms': False, 'fuse_allreduce_comms': False}, max_cudagraph_capture_size=0,
dynamic_shapes_config={'type': <DynamicShapesType.BACKED: 'backed'>, 'evaluate_guards': False,
'assume_32_bit_indexing': True}, local_cache_dir=None)
(EngineCore_DPO pid=511) INFO 03-10 16:30:11 [parallel_state.py:1214] world_size=1 rank=0
local_rank=0 distributed_init_method=tcp://192.0.2.101:47903 backend=nccl
(EngineCore_DPO pid=511) INFO 03-10 16:30:21 [parallel_state.py:1425] rank 0 in world size 1 is
assigned as DP rank 0, PP rank 0, TP rank 0, EP rank N/A
(EngineCore_DPO pid=511) INFO 03-10 16:30:22 [gpu_model_runner.py:3808] Starting to load model
BAAI/bge-reranker-v2-m3...
(EngineCore_DPO pid=511) INFO 03-10 16:30:38 [cuda.py:351] Using FLASH_ATTN attention backend out
of potential backends: ('FLASH_ATTN', 'TRITON_ATTN', 'FLEX_ATTENTION')
(EngineCore_DPO pid=511) INFO 03-10 16:31:00 [weight_utils.py:510] Time spent downloading weights
for BAAI/bge-reranker-v2-m3: 16.985564 seconds
(EngineCore_DPO pid=511) INFO 03-10 16:31:00 [weight_utils.py:550] No model.safetensors.index.json
found in remote.
Loading safetensors checkpoint shards:   0% Completed | 0/1 [00:00<?, ?it/s]
Loading safetensors checkpoint shards: 100% Completed | 1/1 [00:00<00:00, 13.62it/s]
(EngineCore_DPO pid=511) INFO 03-10 16:31:01 [default_loader.py:291] Loading weights took 0.30
seconds
(EngineCore_DPO pid=511) INFO 03-10 16:31:01 [gpu_model_runner.py:3905] Model loading took 1.06
GiB memory and 38.238101 seconds
(EngineCore_DPO pid=511) INFO 03-10 16:31:01 [core.py:273] init engine (profile, create kv cache,
warm up model) took 0.28 seconds
(EngineCore_DPO pid=511) INFO 03-10 16:31:03 [vllm.py:630] Asynchronous scheduling is enabled.
(EngineCore_DPO pid=511) WARNING 03-10 16:31:03 [vllm.py:672] Inductor compilation was disabled by
user settings, optimizations settings that are only active during inductor compilation will be
ignored.
(EngineCore_DPO pid=511) INFO 03-10 16:31:03 [vllm.py:765] Cudagraph is disabled under eager mode
```



```
(APIServer pid=1) INFO 03-10 16:31:03 [api_server.py:1014] Supported tasks: ['classify',
'token_classify', 'score']
(APIServer pid=1) INFO 03-10 16:31:03 [api_server.py:1346] Starting vLLM API server 0 on
http://0.0.0.0:8002
(APIServer pid=1) INFO 03-10 16:31:03 [launcher.py:38] Available routes are:
(APIServer pid=1) INFO 03-10 16:31:03 [launcher.py:46] Route: /openapi.json, Methods: GET, HEAD
(APIServer pid=1) INFO 03-10 16:31:03 [launcher.py:46] Route: /docs, Methods: GET, HEAD
(APIServer pid=1) INFO 03-10 16:31:03 [launcher.py:46] Route: /scale_elastic_ep, Methods: POST
(APIServer pid=1) INFO 03-10 16:31:03 [launcher.py:46] Route: /is_scaling_elastic_ep, Methods:
POST
(APIServer pid=1) INFO 03-10 16:31:03 [launcher.py:46] Route: /tokenize, Methods: POST
(APIServer pid=1) INFO 03-10 16:31:03 [launcher.py:46] Route: /detokenize, Methods: POST
(APIServer pid=1) INFO 03-10 16:31:03 [launcher.py:46] Route: /inference/v1/generate, Methods:
POST
(APIServer pid=1) INFO 03-10 16:31:03 [launcher.py:46] Route: /pause, Methods: POST
(APIServer pid=1) INFO 03-10 16:31:03 [launcher.py:46] Route: /resume, Methods: POST
(APIServer pid=1) INFO 03-10 16:31:03 [launcher.py:46] Route: /is_paused, Methods: GET
(APIServer pid=1) INFO 03-10 16:31:03 [launcher.py:46] Route: /metrics, Methods: GET
(APIServer pid=1) INFO 03-10 16:31:03 [launcher.py:46] Route: /health, Methods: GET
(APIServer pid=1) INFO 03-10 16:31:03 [launcher.py:46] Route: /load, Methods: GET
(APIServer pid=1) INFO 03-10 16:31:03 [launcher.py:46] Route: /v1/models, Methods: GET
(APIServer pid=1) INFO 03-10 16:31:03 [launcher.py:46] Route: /version, Methods: GET
(APIServer pid=1) INFO 03-10 16:31:03 [launcher.py:46] Route: /v1/responses, Methods: POST
(APIServer pid=1) INFO 03-10 16:31:03 [launcher.py:46] Route: /v1/responses/{response_id},
Methods: GET
(APIServer pid=1) INFO 03-10 16:31:03 [launcher.py:46] Route: /v1/responses/{response_id}/cancel,
Methods: POST
(APIServer pid=1) INFO 03-10 16:31:03 [launcher.py:46] Route: /v1/messages, Methods: POST
(APIServer pid=1) INFO 03-10 16:31:03 [launcher.py:46] Route: /v1/chat/completions, Methods: POST
(APIServer pid=1) INFO 03-10 16:31:03 [launcher.py:46] Route: /v1/completions, Methods: POST
(APIServer pid=1) INFO 03-10 16:31:03 [launcher.py:46] Route: /v1/audio/transcriptions, Methods:
POST
(APIServer pid=1) INFO 03-10 16:31:03 [launcher.py:46] Route: /v1/audio/translations, Methods:
POST
(APIServer pid=1) INFO 03-10 16:31:03 [launcher.py:46] Route: /ping, Methods: GET
(APIServer pid=1) INFO 03-10 16:31:03 [launcher.py:46] Route: /ping, Methods: POST
(APIServer pid=1) INFO 03-10 16:31:03 [launcher.py:46] Route: /invocations, Methods: POST
(APIServer pid=1) INFO 03-10 16:31:03 [launcher.py:46] Route: /classify, Methods: POST
(APIServer pid=1) INFO 03-10 16:31:03 [launcher.py:46] Route: /v1/embeddings, Methods: POST
(APIServer pid=1) INFO 03-10 16:31:03 [launcher.py:46] Route: /score, Methods: POST
(APIServer pid=1) INFO 03-10 16:31:03 [launcher.py:46] Route: /v1/score, Methods: POST
(APIServer pid=1) INFO 03-10 16:31:03 [launcher.py:46] Route: /rerank, Methods: POST
(APIServer pid=1) INFO 03-10 16:31:03 [launcher.py:46] Route: /v1/rerank, Methods: POST
(APIServer pid=1) INFO 03-10 16:31:03 [launcher.py:46] Route: /v2/rerank, Methods: POST
(APIServer pid=1) INFO 03-10 16:31:03 [launcher.py:46] Route: /pooling, Methods: POST
(APIServer pid=1) INFO:
(APIServer pid=1) INFO:     Started server process [1]
(APIServer pid=1) INFO:     Waiting for application startup.
(APIServer pid=1) INFO:     Application startup complete.
```


View Filtered Log Indicators Only

Reranker model initialization logs contain extensive details about model architecture detection, weight loading, and task configuration. This filtered command shows only the essential status messages to quickly verify successful deployment without scrolling through verbose initialization output.

```
docker logs vllm-reranker 2>&1 | grep -E "vLLM API server
version|XLMRobertaForSequenceClassification|Starting to load model|Loading weights took|Model
loading took|Supported tasks.*classify.*score|Started server process|Waiting for application
startup|Application startup complete|/rerank"
```

```
root@phase3-nvidia:~# docker logs vllm-reranker 2>&1 | grep -E "vLLM API server
version|XLMRobertaForSequenceClassification|Starting to load model|Loading weights took|Model
loading took|Supported tasks.*classify.*score|Started server process|Waiting for application
startup|Application startup complete|/rerank"
(APIServer pid=1) INFO 03-13 11:22:18 [api_server.py:1272] vLLM API server version 0.14.1
(APIServer pid=1) INFO 03-13 11:22:29 [model.py:530] Resolved architecture:
XLMRobertaForSequenceClassification
(EngineCore_DPO pid=511) INFO 03-13 11:23:02 [gpu_model_runner.py:3808] Starting to load model
BAAI/bge-reranker-v2-m3...
(EngineCore_DPO pid=511) INFO 03-13 11:23:40 [default_loader.py:291] Loading weights took 0.30
seconds
(EngineCore_DPO pid=511) INFO 03-13 11:23:41 [gpu_model_runner.py:3905] Model loading took 1.06
GiB memory and 38.577223 seconds
(APIServer pid=1) INFO 03-13 11:23:43 [api_server.py:1014] Supported tasks: ['classify', 'score',
'token_classify']
(APIServer pid=1) INFO 03-13 11:23:43 [launcher.py:46] Route: /rerank, Methods: POST
(APIServer pid=1) INFO 03-13 11:23:43 [launcher.py:46] Route: /v1/rerank, Methods: POST
(APIServer pid=1) INFO 03-13 11:23:43 [launcher.py:46] Route: /v2/rerank, Methods: POST
(APIServer pid=1) INFO:     Started server process [1]
(APIServer pid=1) INFO:     Waiting for application startup.
(APIServer pid=1) INFO:     Application startup complete.
root@phase3-nvidia:~#
```

Key Log Indicators

- "vLLM API server version 0.14.1": API server initialization
- "Resolved architecture: XLMRobertaForSequenceClassification": Reranker architecture detected
- "Starting to load model BAAI/bge-reranker-v2-m3...": Model loading begins
- "Loading weights took X.XX seconds": Model weights loaded
- "Model loading took X.XX GiB memory": Model fully initialized
- "Supported tasks: ['classify', 'score', 'token_classify']": Reranking capabilities enabled
- "Started server process [1]": Server process ready
- "Application startup complete.": Service ready

Expected Output:

- Filtered logs confirming model initialization and endpoint availability
- Log sequence showing model loading and API server startup

Verify Container Status

```
docker ps | grep vllm-reranker
```

```
root@phase3-nvidia:~# docker ps | grep vllm-reranker
f87ee00b18e4   nvcr.io/nvidia/ai-dynamo/vllm-runtime:0.9.0-cuda13   "/opt/nvidia/nvidia_…"   3 minutes ago   Up 3 minutes   vllm-reranker
root@phase3-nvidia:~#
```

Expected Output:

- Container running with status "**Up X minutes**"

Service Endpoint Information

The reranker service exposes endpoints for scoring query-document relevance, enabling improved retrieval accuracy in RAG pipelines.

Endpoint Details

- Primary URL: http://<NVIDIA_GPU_SERVER_IP>:8002/rerank (Cohere-compatible)
- Alternative URL: http://<NVIDIA_GPU_SERVER_IP>:8002/score (OpenAI-style)
- Model: BAAI/bge-reranker-v2-m3
- Max Tokens: 8192
- API Standard: Cohere-compatible (primary), OpenAI-style (alternative)
- Authentication: None (internal network deployment)

Example Python Integration (Cohere-style):

```
import httpx

response = httpx.post(
    "http://<NVIDIA_GPU_SERVER_IP>:8002/rerank",
    json={
        "model": "BAAI/bge-reranker-v2-m3",
        "query": "What is AI?",
        "documents": [
            "AI is artificial intelligence",
            "Machine learning is a subset of AI",
            "Unrelated text about cooking"
        ]
    }
)

result = response.json()
# Returns: {"results": [{"index": 0, "relevance_score": 0.98}, ...]}
# Documents are sorted by relevance score (highest first)
```

Example Python Integration (OpenAI-style):

```
import httpx

response = httpx.post(
    "http://<NVIDIA_GPU_SERVER_IP>:8002/score",
    json={
        "model": "BAAI/bge-reranker-v2-m3",
        "query": "What is AI?",
        "documents": [
            "AI is artificial intelligence",
            "Machine learning is a subset of AI",
            "Unrelated text about cooking"
        ]
    }
)

result = response.json()
# Returns: {"scores": [0.98, 0.85, 0.12]}
# Scores correspond to documents in the same order as input
```

Note: Replace `<NVIDIA_GPU_SERVER_IP>` with the internal IP address of the GPU server.

Verify Reranker Model Endpoint

Endpoint verification confirms that the reranker service is responding correctly and can evaluate query-document relevance. The service exposes two endpoint styles: Cohere-compatible /rerank and OpenAI-style /score.

Check Available Models

```
curl http://localhost:8002/v1/models
```

```
root@phase3-nvidia:~# curl http://localhost:8002/v1/models

{"object":"list","data":[{"id":"BAAI/bge-reranker-v2-
m3","object":"model","created":1773160474,"owned_by":"vllm","root":"BAAI/bge-reranker-v2-
m3","parent":null,"max_model_len":8192,"permission":[{"id":"modelperm-
887b2da78a2a84e","object":"model_permission","created":1773160474,"allow_create_engine":false,"all
ow_sampling":true,"allow_logprobs":true,"allow_search_indices":false,"allow_view":true,"allow_fine
_tuning":false,"organization":"*","group":null,"is_blocking":false}]}]}

root@phase3-nvidia:~#
```

Expected Output:

- JSON response listing **BAAI/bge-reranker-v2-m3** with model metadata

Test Rerank Endpoint (Cohere-compatible)

```
curl -X POST http://localhost:8002/rerank \
  -H "Content-Type: application/json" \
  -d '{
    "model": "BAAI/bge-reranker-v2-m3",
    "query": "What is AI?",
    "documents": [
      "AI is artificial intelligence",
      "Unrelated text about cooking"
    ]
  }' | jq
```

```
root@phase3-nvidia:~# curl -X POST http://localhost:8002/rerank \
-H "Content-Type: application/json" \
-d '{
  "model": "BAAI/bge-reranker-v2-m3",
  "query": "What is AI?",
  "documents": [
    "AI is artificial intelligence",
    "Unrelated text about cooking"
  ]
}' | jq
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   542  100   368  100   174   2662   1259 --:--:-- --:--:-- --:--:--  3927
{
  "id": "rerank-944b7bc085e87317",
```

```
  "model": "BAAI/bge-reranker-v2-m3",
  "usage": {
    "prompt_tokens": 26,
    "total_tokens": 26
  },
  "results": [
    {
      "index": 0,
      "document": {
        "text": "AI is artificial intelligence",
        "multi_modal": null
      },
      "relevance_score": 0.9986361861228943
    },
    {
      "index": 1,
      "document": {
        "text": "Unrelated text about cooking",
        "multi_modal": null
      },
      "relevance_score": 0.00001601198346179443
    }
  ]
}
root@phase3-nvidia:~#
```

Expected Output:

- JSON response containing **results** array with document indices and relevance scores (sorted by highest relevance)

Test Score Endpoint (OpenAI-style)

```
curl -X POST http://localhost:8002/score \
  -H "Content-Type: application/json" \
  -d '{
    "model": "BAAI/bge-reranker-v2-m3",
    "text_1": "What is AI?",
    "text_2": [
      "AI is artificial intelligence",
      "Unrelated text about cooking"
    ]
  }' | jq
```

```
root@phase3-nvidia:~# curl -X POST http://localhost:8002/score \
-H "Content-Type: application/json" \
-d '{
  "model": "BAAI/bge-reranker-v2-m3",
  "text_1": "What is AI?",
  "text_2": [
    "AI is artificial intelligence",
    "Unrelated text about cooking"
  ]
```



```
}' | jq
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   497  100   325  100   172  11973   6336 --:--:-- --:--:-- --:--:-- 18407
{
  "id": "score-86a0db3cca5a2b2f",
  "object": "list",
  "created": 1773160665,
  "model": "BAAI/bge-reranker-v2-m3",
  "data": [
    {
      "index": 0,
      "object": "score",
      "score": 0.9986361861228943
    },
    {
      "index": 1,
      "object": "score",
      "score": 0.00001601198346179444 3
    }
  ],
  "usage": {
    "prompt_tokens": 26,
    "total_tokens": 26,
    "completion_tokens": 0,
    "prompt_tokens_details": null
  }
}
root@phase3-nvidia:~#
```

Expected Output:

- JSON response containing scores array with relevance scores corresponding to input documents

Deploy Document Parser Model using vLLM

Document parser models extract structured content from document images and convert visual layouts into machine-readable formats such as Markdown. The **nvidia/NVIDIA-Nemotron-Parse-v1.1** model used in this deployment is a vision-language model designed to process document images, including PDF pages, scans, and screenshots, while preserving formatting, tables, and hierarchical structure.

This capability is important for RAG workflows because it enables:

- Document ingestion from image-based sources without external OCR tools
- Processing of scanned documents with layout and structure preservation
- Parsing of screenshots and visual documents into structured text
- Extraction of tables and hierarchical content for indexing and retrieval

The deployment uses standalone vLLM with vision model support and exposes an OpenAI-compatible chat completions endpoint that accepts base64-encoded images as input.

Deployment Characteristics

- Single GPU deployment, sufficient for document parsing workloads
- Vision-language model with image input support
- OpenAI-compatible API with vision extensions
- Structured output in Markdown format

Deploy Document Parser Worker Container

The document parser model is deployed as a standalone vLLM container using the shared cache directory configured earlier under "Setup vLLM Cache Directory." The service exposes an OpenAI-compatible chat completions endpoint that accepts image inputs and returns structured Markdown output with bounding boxes and semantic classes. Required dependencies are installed at runtime to support vision-based processing.

Ensure Environment Variables are set:

```
echo $HF_TOKEN
```

```
root@phase3-nvidia :~ # echo $HF_TOKEN
hf_ xxxxxxxxxxxxxxxx
root@phase3-nvidia :~ #
```

```
echo $DYNAMO_IMAGE
```

```
root@phase3-nvidia :~ # echo $DYNAMO_IMAGE
nvcr.io/nvidia/ai-dynamo/vllm-runtime:0.9.0-cuda13
root@phase3-nvidia :~ #
```

Expected Output:

- Both variables should display their values

If DYNAMO_IMAGE is not set:

```
export DYNAMO_IMAGE=nvcr.io/nvidia/ai-dynamo/vllm-runtime:0.9.0-cuda13
```

```
root@phase3-nvidia:~# export DYNAMO_IMAGE=nvcr.io/nvidia/ai-dynamo/vllm-runtime:0.9.0-cuda13
root@phase3-nvidia:~#
```

Run Document Parser Container

```
docker run -d --name vllm-parser --restart always --gpus '"device=6"' --network host --ipc=host --ulimit memlock=-1 --ulimit stack=67108864 -e HF_TOKEN=$HF_TOKEN -v ~/vllm-models/.cache:/root/.cache/huggingface $DYNAMO_IMAGE bash -c "python -m pip install simsimd stringzilla && python -m pip install --no-deps albumentations==2.0.8 albucore==0.0.24 && python -m pip install --no-deps timm==1.0.22 open_clip_torch && vllm serve nvidia/NVIDIA-Nemotron-Parse-v1.1 --dtype bfloat16 --max-num-seqs 8 --limit-mm-per-prompt '{\"image\": 1}' --trust-remote-code --attention-backend TRITON_ATTN --port 8003"
```

```
root@phase3-nvidia:~# docker run -d --name vllm-parser --restart always --gpus '"device=6"' --network host --ipc=host --ulimit memlock=-1 --ulimit stack=67108864 -e HF_TOKEN=$HF_TOKEN -v ~/vllm-models/.cache:/root/.cache/huggingface $DYNAMO_IMAGE bash -c "python -m pip install simsimd stringzilla && python -m pip install --no-deps albumentations==2.0.8 albucore==0.0.24 && python -m pip install --no-deps timm==1.0.22 open_clip_torch && vllm serve nvidia/NVIDIA-Nemotron-Parse-v1.1 --dtype bfloat16 --max-num-seqs 8 --limit-mm-per-prompt '{\"image\": 1}' --trust-remote-code --attention-backend TRITON_ATTN --port 8003"
22b3bd16bb57a962f4b7e99ddea999f983fe1179c02f4267f8b67e91f87d7d1a
root@phase3-nvidia:~#
```

Expected Output:

- Container ID (64-character hexadecimal string)

Container Runtime Parameters

- **--name vllm-parser:** Defines the container name for management
- **--restart always:** Ensures automatic restart on failure or system reboot
- **--gpus '"device=6"':** Allocates GPU 6 for document parsing workloads
- **--network host:** Enables host networking without port mapping
- **--ipc=host:** Provides shared memory for GPU operations
- **--ulimit memlock=-1:** Removes memory lock limitations for GPU workloads
- **--ulimit stack=67108864:** Increases stack size for model execution
- **-e HF_TOKEN:** Passes Hugging Face token for model download
- **-v ~/vllm-models/.cache:** Mounts shared cache directory for model reuse
- **$DYNAMO_IMAGE:** Uses the same runtime image as other vLLM services
- **bash -c:** Installs required dependencies before starting the vLLM server
- **pip install simsimd stringzilla:** Core dependencies for image processing stack
- **pip install --no-deps albumentations albucore:** Image processing libraries (avoids dependency conflicts)
- **pip install --no-deps timm open_clip_torch:** Vision model dependencies
- **vllm serve:** Invokes vLLM serving interface with vision model support

- **--dtype bfloat16:** Uses bfloat16 precision for optimized performance
- **--max-num-seqs 8:** Limits concurrent sequences for memory management
- **--limit-mm-per-prompt:** Allows one image per request
- **--attention-backend TRITON_ATTN:** Optimized attention backend for **NVIDIA** GPUs
- **--port 8003:** Exposes document parser service on port 8003

Monitor Document Parser Model Initialization

Model initialization for vision-language models involves downloading model weights, installing runtime dependencies, loading both vision and language components into GPU memory, and starting the API server. Monitoring container logs provides visibility into each stage of initialization and helps confirm successful deployment.

Monitor Container Logs

```
docker logs -f vllm-parser
```

```
root@phase3-nvidia:~# docker logs -f vllm-parser

==========
== CUDA ==
==========

CUDA Version 13.0.2

Container image Copyright (c) 2016-2023, NVIDIA CORPORATION & AFFILIATES. All rights reserved.

This container image and its contents are governed by the NVIDIA Deep Learning Container License.
By pulling and using the container, you accept the terms and conditions of this license:
https://developer.nvidia.com/ngc/nvidia-deep-learning-container-license

A copy of this license is made available in this container at /NGC-DL-CONTAINER-LICENSE for your
convenience.

Collecting simsimd
  Downloading simsimd-6.5.16-cp312-cp312-
manylinux2014_x86_64.manylinux_2_17_x86_64.manylinux_2_28_x86_64.whl.metadata (70 kB)
Collecting stringzilla
  Downloading stringzilla-4.6.0-cp312-cp312-
manylinux_2_17_x86_64.manylinux2014_x86_64.manylinux_2_28_x86_64.whl.metadata (121 kB)
Downloading simsimd-6.5.16-cp312-cp312-
manylinux2014_x86_64.manylinux_2_17_x86_64.manylinux_2_28_x86_64.whl (583 kB)

Installing collected packages: timm, open_clip_torch
Successfully installed open_clip_torch-3.3.0 timm-1.0.22
[2026-03-10 16:42:41] INFO font_manager.py:1639: generated new fontManager
(APIserver pid=1) INFO 03-10 16:42:42 [api_server.py:1272] vLLM API server version 0.14.1
```

```
(APIserver pid=1) INFO 03-10 16:42:42 [utils.py:263] non-default args: {'model_tag':
'nvidia/NVIDIA-Nemotron-Parser-v1.1', 'port': 8003, 'model': 'nvidia/NVIDIA-Nemotron-Parser-v1.1',
'trust_remote_code': True, 'dtype': 'bfloat16', 'attention_backend': 'TRITON_ATTN',
'limit_mm_per_prompt': {'image': 1}, 'max_num_seqs': 8}
(APIserver pid=1) The argument `trust_remote_code` is to be used with Auto classes. It has no
effect here and is ignored.
(APIserver pid=1) The argument `trust_remote_code` is to be used with Auto classes. It has no
effect here and is ignored.
(APIserver pid=1) A new version of the following files was downloaded from
https://huggingface.co/nvidia/NVIDIA-Nemotron-Parse-v1.1:
(APIserver pid=1) - hf_nemotron_parse_config.py
(APIserver pid=1) Make sure to double-check they do not contain any added malicious code. To avoid
downloading new versions of the code file, you can pin a revision.
(APIserver pid=1) A new version of the following files was downloaded from
https://huggingface.co/nvidia/CARDIOv2-H:
(APIserver pid=1) - enable_spectral_reparam.py
(APIserver pid=1) Make sure to double-check they do not contain any added malicious code. To avoid
downloading new versions of the code file, you can pin a revision.
(APIserver pid=1) A new version of the following files was downloaded from
https://huggingface.co/nvidia/CARDIOv2-H:
(APIserver pid=1) - adaptor_mlp.py

Capturing CUDA graphs (decode, FULL): 100%|██████████| 4/4 [00:02<00:00, 1.58it/s]
(EngineCore_DPO pid=612) INFO 03-10 16:44:06 [gpu_model_runner.py:4856] Graph capturing finished
in 3 secs, took 0.03 GiB
(EngineCore_DPO pid=612) INFO 03-10 16:44:06 [core.py:273] init engine (profile, create kv cache,
warm up model) took 10.20 seconds
(EngineCore_DPO pid=612) INFO 03-10 16:44:07 [vllm.py:630] Asynchronous scheduling is enabled.
(EngineCore_DPO pid=612) WARNING 03-10 16:44:07 [vllm.py:908] No piecewise cudagraph for executing
cascade attention. Will fall back to eager execution if a batch runs into cascade attentions.
(APIserver pid=1) INFO 03-10 16:44:07 [api_server.py:1014] Supported tasks: ['generate']
(APIserver pid=1) WARNING 03-10 16:44:07 [model.py:1358] Default sampling parameters have been
overridden by the model's Hugging Face generation config recommended from the model creator. If
this is not intended, please relaunch vLLM instance with `--generation-config vllm`.
(APIserver pid=1) INFO 03-10 16:44:07 [serving_responses.py:224] Using default chat sampling
params from model: {'repetition_penalty': 1.1, 'max_tokens': 9000}
(APIserver pid=1) INFO 03-10 16:44:07 [serving_chat.py:146] Using default chat sampling params
from model: {'repetition_penalty': 1.1, 'max_tokens': 9000}
(APIserver pid=1) INFO 03-10 16:44:07 [serving_chat.py:182] Warming up chat template processing...
(APIserver pid=1) Using a slow image processor as `use_fast` is unset and a slow processor was
saved with this model. `use_fast=True` will be the default behavior in v4.52, even if the model
was saved with a slow processor. This will result in minor differences in outputs. You'll still be
able to use a slow processor with `use_fast=False`.
(APIserver pid=1)
/home/dynamo/.cache/huggingface/modules/transformers_modules/nvidia/NVIDIA_Nemotron_Parse_v1_1/fb3
948d4a35f899fd737010c0b89f26fe49344c8/hf_nemotron_parse_process

(APIserver pid=1) INFO 03-10 16:44:14 [launcher.py:46] Route: /v1/responses, Methods: POST
(APIserver pid=1) INFO 03-10 16:44:14 [launcher.py:46] Route: /v1/responses/{response_id},
Methods: GET
(APIserver pid=1) INFO 03-10 16:44:14 [launcher.py:46] Route: /v1/responses/{response_id}/cancel,
Methods: POST
(APIserver pid=1) INFO 03-10 16:44:14 [launcher.py:46] Route: /v1/messages, Methods: POST
(APIserver pid=1) INFO 03-10 16:44:14 [launcher.py:46] Route: /v1/chat/completions, Methods: POST
(APIserver pid=1) INFO 03-10 16:44:14 [launcher.py:46] Route: /v1/completions, Methods: POST
```



```
(APIserver pid=1) INFO 03-10 16:44:14 [launcher.py:46] Route: /v1/audio/transcriptions, Methods:
POST
(APIserver pid=1) INFO 03-10 16:44:14 [launcher.py:46] Route: /v1/audio/translations, Methods:
POST
(APIserver pid=1) INFO 03-10 16:44:14 [launcher.py:46] Route: /ping, Methods: GET
(APIserver pid=1) INFO 03-10 16:44:14 [launcher.py:46] Route: /ping, Methods: POST
(APIserver pid=1) INFO 03-10 16:44:14 [launcher.py:46] Route: /invocations, Methods: POST
(APIserver pid=1) INFO 03-10 16:44:14 [launcher.py:46] Route: /classify, Methods: POST
(APIserver pid=1) INFO 03-10 16:44:14 [launcher.py:46] Route: /v1/embeddings, Methods: POST
(APIserver pid=1) INFO 03-10 16:44:14 [launcher.py:46] Route: /score, Methods: POST
(APIserver pid=1) INFO 03-10 16:44:14 [launcher.py:46] Route: /v1/score, Methods: POST
(APIserver pid=1) INFO 03-10 16:44:14 [launcher.py:46] Route: /rerank, Methods: POST
(APIserver pid=1) INFO 03-10 16:44:14 [launcher.py:46] Route: /v1/rerank, Methods: POST
(APIserver pid=1) INFO 03-10 16:44:14 [launcher.py:46] Route: /v2/rerank, Methods: POST
(APIserver pid=1) INFO 03-10 16:44:14 [launcher.py:46] Route: /pooling, Methods: POST

(APIserver pid=1) INFO:
(APIserver pid=1) INFO:     Started server process [1]
(APIserver pid=1) INFO:     Waiting for application startup.
(APIserver pid=1) INFO:     Application startup complete.
```

Expected Output:

- Log sequence showing dependency installation, API server initialization, model loading, and document parser service startup completion

Key Log Indicators

- "Successfully installed simsimd-X.X.X stringzilla-X.X.X": Core dependencies installed
- "Successfully installed albucore-X.X.X albumentations-X.X.X": Image processing libraries installed
- "Successfully installed open_clip_torch-X.X.X timm-X.X.X": Vision model dependencies installed
- "vLLM API server version 0.14.1": API server initialization
- "Resolved architecture: NemotronParseForConditionalGeneration": Document parser architecture detected
- "Starting to load model nvidia/NVIDIA-Nemotron-Parse-v1.1...": Model loading begins
- "Loading weights took X.XX seconds": Model weights loaded into GPU memory
- "Model loading took X.XX GiB memory": Model fully loaded and ready
- "Supported tasks: ['generate']": Document parsing capabilities enabled
- "Chat template warmup completed": Vision processing pipeline initialized
- "Started server process [1]": Server process ready
- "Waiting for application startup.": Application initialization in progress
- "Application startup complete.": Document parser service ready to accept requests

Press **Ctrl+C** to exit the log stream. The container continues running in the background.

View Filtered Log Indicators Only

Document parser deployment involves dependency installation, vision model loading, and chat template configuration, generating extensive logs. This filtered view extracts only the critical milestones to confirm successful deployment without wading through detailed installation and initialization output.

```
docker logs vllm-parser 2>&1 | grep -E "Successfully installed.*simsimd|Successfully
installed.*albumentations|Successfully installed.*timm|vLLM API server
version|NemotronParseForConditionalGeneration|Starting to load model|Loading weights took|Model
loading took|Supported tasks.*generate|Chat template warmup completed|Started server
process|Waiting for application startup|Application startup complete"
```

```
root@phase3-nvidia:~#  docker logs vllm-parser 2>&1 | grep -E "Successfully
installed.*simsimd|Successfully installed.*albumentations|Successfully installed.*timm|LLM API
server version|NemotronParseForConditionalGeneration|Starting to load model|Loading weights
took|Model loading took|Supported tasks.*generate|Chat template warmup completed|Started server
process|Waiting for application startup|Application startup complete"

Successfully installed simsimd-6.5.16 stringzilla-4.6.0
Successfully installed albucore-0.0.24 albumentations-2.0.8
Successfully installed open_clip_torch-3.3.0 timm-1.0.22
(APIserver pid=1) INFO 03-13 11:32:40 [api_server.py:1272] vLLM API server version 0.14.1
(APIserver pid=1) INFO 03-13 11:32:59 [model.py:530] Resolved architecture:
NemotronParseForConditionalGeneration
(EngineCore_DPO pid=612) INFO 03-13 11:33:25 [gpu_model_runner.py:3808] Starting to load model
nvidia/NVIDIA-Nemotron-Parse-v1.1...
(EngineCore_DPO pid=612) INFO 03-13 11:33:52 [default_loader.py:291] Loading weights took 0.46
seconds
(EngineCore_DPO pid=612) INFO 03-13 11:33:53 [gpu_model_runner.py:3905] Model loading took 1.75
GiB memory and 27.424612 seconds
(APIserver pid=1) INFO 03-13 11:34:04 [api_server.py:1014] Supported tasks: ['generate']
(APIserver pid=1) INFO 03-13 11:34:11 [serving_chat.py:218] Chat template warmup completed in
6510.5ms
(APIserver pid=1) INFO:     Started server process [1]
(APIserver pid=1) INFO:     Waiting for application startup.
(APIserver pid=1) INFO:     Application startup complete.
root@phase3-nvidia:~#
```

Expected Output:

- "Successfully installed simsimd-6.5.16 stringzilla-4.6.0" - Core dependencies installed
- "Successfully installed albucore-0.0.24 albumentations-2.0.8" - Image processing libraries installed
- "Successfully installed open_clip_torch-3.3.0 timm-1.0.22" - Vision model dependencies installed
- "vLLM API server version 0.14.1" - API server initialization
- "Resolved architecture: NemotronParseForConditionalGeneration" -

Model architecture confirmed

- "Starting to load model nvidia/NVIDIA-Nemotron-Parse-v1.1..." - Model loading begins
- "Loading weights took X.XX seconds" - Model weights loaded
- "Model loading took X.XX GiB memory and XX.XX seconds" - Model fully initialized
- "Supported tasks: ['generate']" - Parsing capability enabled
- "Chat template warmup completed in XXXXms" - Vision pipeline ready
- "Started server process [1]" - Server process ready
- "Waiting for application startup." - Initialization in progress
- "Application startup complete." - Service ready

Check Container Status

```
docker ps | grep vllm-parser
```

```
root@phase3-nvidia:~# docker ps | grep vllm-parser
c46955820640  nvcr.io/nvidia/ai-dynamo/vllm-runtime:0.9.0-cuda13 "/opt/nvidia/nvidia _..." 4
minutes ago     Up 4 minutes
                                                    vllm-parser
root@phase3-nvidia:~#
```

Expected Output:

- Container running with status "**Up X minutes**"

Service Endpoint Information

The document parser service exposes an OpenAI-compatible chat completions endpoint with vision support for processing document images and generating structured output. This service is used in RAG pipelines to convert document images into structured text for indexing and retrieval.

Endpoint Details

- URL: http://<NVIDIA_GPU_SERVER_IP>:8003/v1/chat/completions
- Model: nvidia/NVIDIA-Nemotron-Parse-v1.1
- Input: Base64-encoded images (PNG, JPEG, PDF pages)
- Output: Structured Markdown text
- Max Tokens: 4096
- API Standard: OpenAI-compatible with vision extensions
- Authentication: None (internal network deployment)

Example Python Integration

```
import base64
import httpx

# Read and encode image
with open("document.png", "rb") as f:
    img_b64 = base64.b64encode(f.read()).decode("utf-8")

# Parse document
response = httpx.post(
    "http://<NVIDIA_GPU_SERVER_IP>:8003/v1/chat/completions",
    json={
        "model": "nvidia/NVIDIA-Nemotron-Parse-v1.1",
        "messages": [{
            "role": "user",
            "content": [
                {
                    "type": "text",
                    "text": "</s><s><predict_bbox><predict_classes><output_markdown>"
                },
                {
                    "type": "image_url",
                    "image_url": {
                        "url": f"{img_b64}"
                    }
                }
            ]
        }],
        "max_tokens": 2000,
        "temperature": 0.0,
        "extra_body": {
            "repetition_penalty": 1.1,
            "top_k": 1,
            "skip_special_tokens": False
        }
    }
)

result = response.json()
parsed_content = result["choices"][0]["message"]["content"]
print(parsed_content)
```

Note: Replace `<NVIDIA_GPU_SERVER_IP>` with the internal IP address of the GPU server.

Verify Document Parser Model Endpoint

Endpoint verification confirms that the document parser service is responding correctly and can process document images. This validation includes model availability, test script creation, file transfer, and parsing execution.

Check Available Models

```
curl http://localhost:8003/v1/models
```

```
root@phase3-nvidia:~# curl http://localhost:8003/v1/models
{"object":"list","data":[{"id":"nvidia/NVIDIA-Nemotron-Parse-v1.1","object":"model","created"
:1773161179,"owned_by":"vllm","root":"nvidia/NVIDIA-Nemotron-Parse-v1.1","parent":null,
"max_model_len":9000,"permission":[{"id":"modelperm-bcc35809c44eb8a4","object":"model_permission",
"created":1773161179,"allow_create_engine":false,"allow_sampling":true,"allow_logprobs":true,"allo
w_search_indices":false,"allow_view":true,"allow_fine_tuning":false,"organization":"*","group":nul
l,"is_blocking":false}]}]}root@phase3-nvidia:~#
```

Expected Output:

- JSON response listing nvidia/NVIDIA-Nemotron-Parse-v1.1 with metadata

Create Test Script for Document Parsing

```
cat << 'EOF' > ~/test_parser.py
import base64
import requests
import sys

if len(sys.argv) < 2:
    print("Usage: python test_parser.py <image_path>")
    sys.exit(1)

image_path = sys.argv[1]

# Read and base64-encode the image
with open(image_path, "rb") as f:
    img_b64 = base64.b64encode(f.read()).decode("utf-8")

# Default prompt for extracting bounding boxes, classes, and markdown text
prompt_text = "</s><s><predict_bbox><predict_classes><output_markdown>"

# Send request to vLLM server
response = requests.post(
    "http://localhost:8003/v1/chat/completions",
    json={
        "model": "nvidia/NVIDIA-Nemotron-Parse-v1.1",
        "messages": [
            {
                "role": "user",
```

```
                "content": [
                    {"type": "text", "text": prompt_text},
                    {
                        "type": "image_url",
                        "image_url": {"url": f"{img_b64}"},
                    },
                ],
            }
        ],
        "max_tokens": 4000,
        "temperature": 0.0,
        "extra_body": {
            "repetition_penalty": 1.1,
            "top_k": 1,
            "skip_special_tokens": False,
        },
    },
)
if response.status_code == 200:
    result = response.json()
    print("Parsed document content:")
    print(result["choices"][0]["message"]["content"])
else:
    print(f"Error: {response.status_code}")
    print(response.text)
EOF
```

```
root@phase3-nvidia:~# cat << 'EOF' > ~/test_parser.py
import base64
import requests
import sys
if len(sys.argv) < 2:
    print("Usage: python test_parser.py <image_path>")
    sys.exit(1)
image_path = sys.argv[1]
# Read and base64-encode the image
with open(image_path, "rb") as f:
    img_b64 = base64.b64encode(f.read()).decode("utf-8")
# Default prompt for extracting bounding boxes, classes, and markdown text
prompt_text = "</s><s><predict_bbox><predict_classes><output_markdown>"
# Send request to vLLM server
response = requests.post(
    "http://localhost:8003/v1/chat/completions",
    json={
        "model": "nvidia/NVIDIA-Nemotron-Parse-v1.1",
        "messages": [
            {
EOF print(response. text)onse.status_code}")content"])mage/png; base64, {img_b64}"},
root@phase3-nvidia :~ #
```

Expected Output:

- Python script created at ~/test_parser.py

Upload Test Document Image from the **local machine** to the GPU server

From Windows (PowerShell or CMD):

```
scp "C:\path\to\your\document.png" root@<GPU_SERVER_IP>:~/test-document.png
```

```
C:\Users\admin>scp "C:\Users\admin\Downloads\test-document.png" root@192.0.2.101:~/test-
document.png
root@192.0.2.101's password:
test-document.png                                                    100% 284KB 197.4KB/s 00:01


C:\Users\ admin>
```

From Linux/Mac:

```
scp /path/to/your/document.png root@<GPU_SERVER_IP>:~/test-document.png
```

Note: Replace **<GPU_SERVER_IP>** with the actual IP of the GPU server. Enter password when prompted.

Verify the file is copied to the GPU server

```
ls -lh ~/test-document.png
```

```
root@phase3-nvidia:~# ls -lh ~/test-document.png
-rw-r--r-- 1 root root 285K Mar 10 16:53 /root/test-document.png
root@phase3-nvidia:~#
```

Expected Output:

- File listing showing **test-document.png** with size

Test Document Parsing

In the GPU server, test the parser model:

```
python3 ~/test_parser.py ~/test-document.png
```

```
root@phase3-nvidia:~# python3 ~/test_parser.py ~/test-document.png
Parsed document content:
<x_0.3037><y_0.3992>7 What were the Fund costs for the past
year?<x_0.4463><y_0.4055><class_Section-header>

<x_0.3105><y_0.4062>(based on a hypothetical $10,000 investment)<x_0.4336><y_0.4117><class_Text>

<x_0.3105><y_0.4484>* [A footnote in this section is only allowed under certain
circumstances.]<x_0.5059><y_0.4539><class_Text>
```



```
<x_0.3027><y_0.4641>8 How did the Fund perform last year and what affected its
performance?<x_0.4727><y_0.4758><class_Section-header>

<x_0.3096><y_0.4789>For the 12-month period ended September 30, 2022, the Fund underperformed its
benchmark, the Bloomberg Barclays U.S. Aggregate Bond Index. The Fund invests all of its assets in
Master Total Return Portfolio (the “Master Portfolio”).<x_0.5186><y_0.5039><class_Text>

<x_0.3115><y_0.5102>WHAT FACTORS INFLUENCED PERFORMANCE<x_0.4805><y_0.5148><class_Section-header>

<x_0.3105><y_0.5188>Exposure to credit-sensitive sectors weighed on the Master Portfolio’s
performance relative to the benchmark over the period, most notably U.S. high yield corporate
bonds and Asian corporate bonds. Exposure to securitized assets also detracted from
performance.<x_0.5176><y_0.543><class_Text>

<x_0.5342><y_0.5828>† Secondary Benchmark lorem ipsum dolor sit amet, consectetur adipiscing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.<x_0.6748><y_0.5969><class_List-
item>

<x_0.3086><y_0.4148>\begin{tabular}{ccc}
Class C & Costs of a $10,000 investment & Costs paid as a percentage of a $10,000 investment\\
Class C & $51,39 & 0.50%\\
\end{tabular}<x_0.5156><y_0.4422><class_Table>

<x_0.5352><y_0.4953>13,000
12,000
11,000
10,000
9,000
2012
2013
2014
2015
2016

Total Return Fund ($) BBG U.S. Aggregate Index ($)
Secondary Benchmark ($)<x_0.6758><y_0.5539><class_Picture>
root@phase3-nvidia:~#
```

Expected Output:

Parsed document content including:

- Markdown-formatted text
- Bounding boxes: <bbox>x1,y1,x2,y2</bbox>
- Semantic classes: <class>Title</class>, <class>Section</class>, etc. Tables and mathematical expressions in LaTeX format

Ingestion Pipeline for Qdrant

This step implements the ingestion pipeline for RAG by parsing documents, generating embeddings, and storing vectors in Qdrant. Ingestion can be performed using the indexer.py script (bulk processing) or via the /ingest/file API endpoint (single file ingestion).

Ingestion Pipeline Overview

The ingestion pipeline consists of four stages:

- Document parsing using Nemotron-Parse
- Text chunking for context preservation
- Embedding generation using BGE-M3
- Vector storage in Qdrant

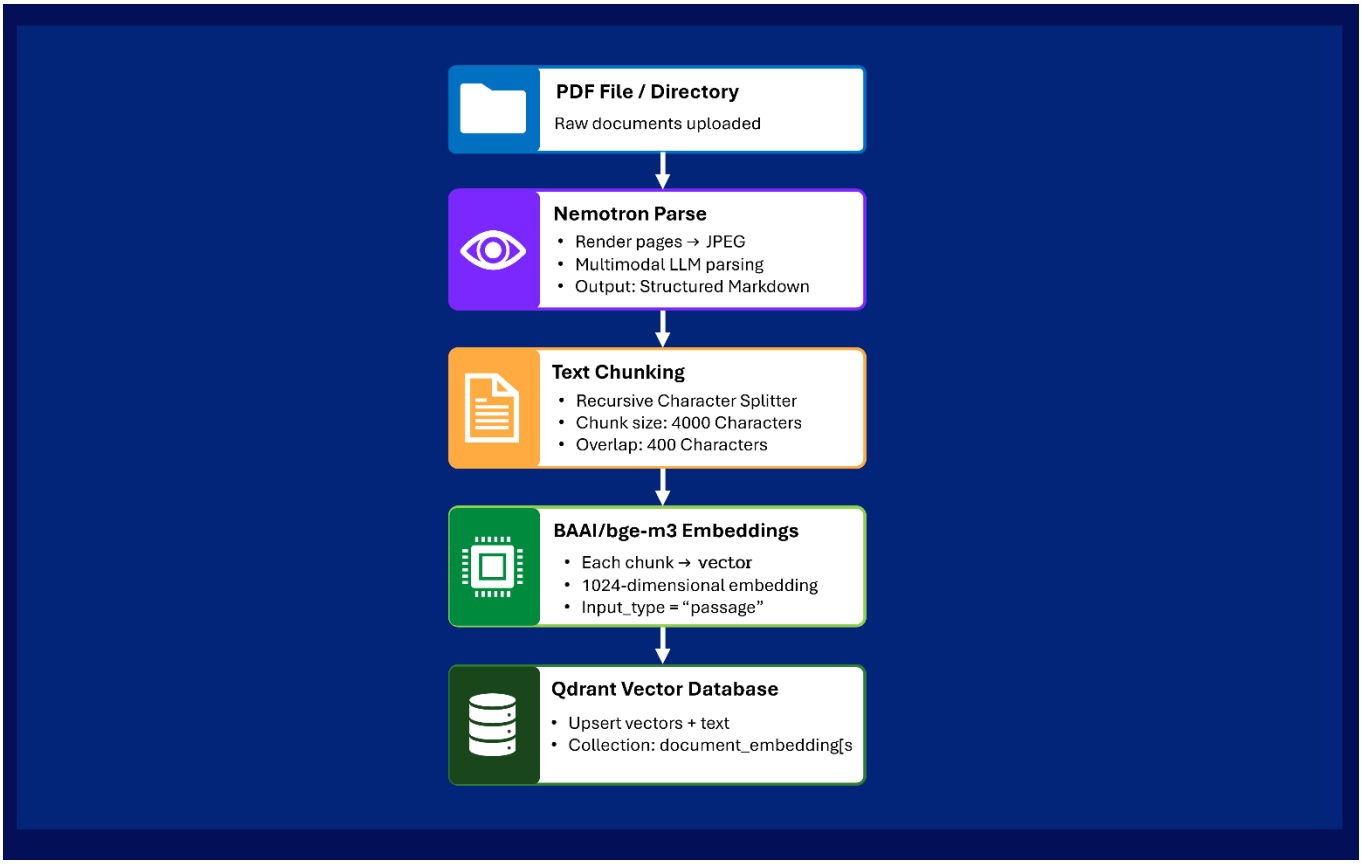


Figure 12 - Document Ingestion Pipeline for Qdrant Vector Indexing

Document Synchronization Workflow

Enterprise documents stored in NetApp ONTAP FlexCache are synchronized to the GPU server using one of the following methods:

- **Direct Mount:** FlexCache volume mounted on the GPU server via NFS.
- **rsync Synchronization:** Periodic synchronization from a FlexCache mount point to local storage.
- **Application-Level Ingestion:** Documents accessed directly from FlexCache by the RAG application.

The synchronized documents are then processed by the document parser, embedded using the embedding model, and indexed in the Qdrant vector database for semantic retrieval.

Parse Documents using Nemotron-Parse

Each page of the PDF is rendered to a JPEG image and sent to Nemotron-Parse. The model returns structured markdown preserving tables, headings, and layout.

```
# From indexer.py - parse_document()
doc = fitz.open(file_path)
for page_num in range(len(doc)):
    page = doc.load_page(page_num)
    pix  = page.get_pixmap()
    image_bytes = pix.tobytes("jpeg")
    b64_data    = base64.b64encode(image_bytes).decode("utf-8")

    response = client.chat.completions.create(
        model="nvidia/NVIDIA-Nemotron-Parse-v1.1",
        messages=[{
            "role": "user",
            "content": [
                {"type": "text",      "text":
"</s><s><predict_bbox><predict_classes><output_markdown>"},
                {"type": "image_url", "image_url": {"url": f"{b64_data}"}}
            ]
        }],
        max_tokens=4000,
        temperature=0.0
    )
    extracted_text += response.choices[0].message.content + "\n\n"
```

Verify Document Parser Endpoint

```
curl -s http://<MODEL_HOST_IP>:8003/v1/models
```

```
curl -s http://192.0.2.101:8003/v1/models

#Expected Output:
{
  "object": "list",
  "data": [{
    "id": "nvidia/NVIDIA-Nemotron-Parse-v1.1",
    "object": "model",
    "created": 1772000000,
    "owned_by": "vllm",
    ...
   }
 ]
}
```

Expected Output:

- JSON response listing nvidia/NVIDIA-Nemotron-Parse-v1.1

Text Chunking

Extracted markdown is split into overlapping chunks to preserve semantic continuity across boundaries.

```
# From indexer.py - process_and_index()
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=4000,    # ~1000-1500 tokens per chunk
    chunk_overlap=400,  # ~100 tokens of overlap between chunks
    length_function=len
)
chunks = text_splitter.split_text(parsed_text)
```

Generate Embeddings

Each text chunk is converted into a **1024-dimensional vector representation** using the embedding model.

```
# From indexer.py - get_embedding()
client   = OpenAI(base_url="http://<MODEL_HOST_IP>:8001/v1", api_key="sk-dummy")
response = client.embeddings.create(
    model="BAAI/bge-m3",
    input=chunk
)
embedding = response.data[0].embedding  # List[float], length 1024
```

```
# From indexer.py - get_embedding()
client   = OpenAI(base_url="http://192.0.2.101:8001/v1", api_key="sk-dummy")
response = client.embeddings.create(
    model="BAAI/bge-m3",
    input=chunk
)
embedding = response.data[0].embedding  # List[float], length 1024
```

Verify Embeddings Endpoint

```
curl -s -X POST http://<MODEL_HOST_IP>:8001/v1/embeddings \
  -H "Content-Type: application/json" \
  -d '{"input": ["test sentence"], "model": "BAAI/bge-m3"}'
```

```
#Expected Output:
{
  "object": "list",
  "data": [
    {
      "object": "embedding",
      "index": 0,
      "embedding": [0.0234, -0.0148, 0.0089, ...]  // 1024 floats
    }
  ],
  "model": "BAAI/bge-m3",
  "usage": { "prompt_tokens": 3, "total_tokens": 3 }
}
```

Expected Output:

- JSON response containing **embedding** vector (1024-dimensional array)

Vector Upsert with Metadata into Qdrant

Each embedding is stored along with metadata such as filename, text content, and chunk index.

```
# From indexer.py - process_and_index()
point = PointStruct(
    id=str(uuid.uuid4()),
    vector=embedding,
    payload={
        "filename":    filename,
        "file_path":   file_path,
        "text":        chunk,
        "chunk_index": chunk_idx
    }
)
qdrant.upsert(collection_name="document_embeddings", points=[point])
```

Verify Qdrant Container Status (In Qdrant host)

```
root@qdrant-compute:~# docker ps
CONTAINER
ID   IMAGE           COMMAND             CREATED         STATUS         PORTS
                                        NAMES
98f84d8bfbdc   qdrant/qdrant   "./entrypoint.sh"   7 seconds ago   Up 7 seconds   0.0.0.0:6333-
6334->6333-6334/tcp, [::]:6333-6334->6333-6334/tcp   qdrant
```

Expected Output:

- Image Name = qdrant
- STATUS = Up
- PORTS= 6333-6334->6333-6334/tcp

If Qdrant container is not running, then configure and start the Qdrant service with required storage and network port settings.

```
# Start Qdrant (run once on the DB node)
docker run -d \
  --name qdrant \
  -p 6333:6333 \
  -p 6334:6334 \
  -v /mnt/qdrant_prod/storage:/qdrant/storage \
  qdrant/qdrant
```

```
Host:  192.0.2.22
Port:   6333 (REST) . 6334 (gRPC)
Container:  qdrant/qdrant (Docker)
Storage:    /mnt/qdrant_prod/storage -> /qdrant/storage (mounted)
```

Verify Qdrant Collections

```
curl -s http://<QDRANT_HOST_IP>:6333/collections
```

```
curl -s http://192.0.2.22:6333/collection
```

```
#Expected Output:
{
  "result": {
  "collections": [
     {"name": "document_embeddings"}
    ]
  },
  "status": "ok",
  "time": 0.000041
}
```

Expected Output:

- JSON response listing **document_embeddings** collection.

Bulk Document Ingestion with indexer.py

The **indexer.py** script recursively scans a directory and ingests all supported documents.

```
# Edit FLEXCACHE_DIR in indexer.py to point documents folder
# FLEXCACHE_DIR = "/mnt/Flexcache_Site2"
```

```
root@nvidia-gpu-compute:~/dynamo_rag_pipeline# source venv/bin/activate
(venv) root@nvidia-gpu-compute:~/dynamo_rag_pipeline# python3 indexer.py
Connecting to Qdrant at 192.0.2.22:6333...
--- Processing: /mnt/Flexcache_Site2/ai-generated-images-with-stable-cascade-and-vultr-cloud-
gpu.pdf ---
Successfully indexed chunk 1 for 'ai-generated-images-with-stable-cascade-and-vultr-cloud-
gpu.pdf'.
Successfully indexed chunk 2 for 'ai-generated-images-with-stable-cascade-and-vultr-cloud-
gpu.pdf'.
Successfully indexed chunk 3 for 'ai-generated-images-with-stable-cascade-and-vultr-cloud-
gpu.pdf'.
Successfully indexed chunk 4 for 'ai-generated-images-with-stable-cascade-and-vultr-cloud-
gpu.pdf'.
Successfully indexed chunk 5 for 'ai-generated-images-with-stable-cascade-and-vultr-cloud-
gpu.pdf'.
Successfully indexed chunk 6 for 'ai-generated-images-with-stable-cascade-and-vultr-cloud-
gpu.pdf'.
Successfully indexed chunk 7 for 'ai-generated-images-with-stable-cascade-and-vultr-cloud-
gpu.pdf'.

--- Processing: /mnt/Flexcache_Site2/solution-brief-qdrant.pdf ---
Successfully indexed chunk 1 for 'solution-brief-qdrant.pdf'.
Successfully indexed chunk 2 for 'solution-brief-qdrant.pdf'.
Successfully indexed chunk 3 for 'solution-brief-qdrant.pdf'.
Indexing finished. Proceeding with export...
Exporting 'document_embeddings' to /mnt/qdrant_prod/qdrant...
Snapshot created successfully. Initiating download...
Downloading to /mnt/qdrant_prod/qdrant/document_embeddings_document_embeddings-7737734939993800-
2026-03-05-07-40-43.snapshot
Export complete.
```

Expected Output:

- Documents are parsed, chunked, embedded, and stored in Qdrant.

Note:

- Re-ingestion is not required after service restarts
- Qdrant persists data at /mnt/qdrant_prod/storage
- Run ingestion only when adding new documents

RAG Pipeline API

The RAG pipeline integrates document ingestion and query workflows using Dynamo inference, Qdrant vector storage, and a Streamlit-based user interface. The FastAPI backend orchestrates parsing, embedding, reranking, and LLM inference to enable end-to-end retrieval-augmented generation (RAG).

Architecture Overview

The architecture integrates LLM serving, embedding, reranking, and vector storage to enable end-to-end retrieval-augmented generation (RAG).

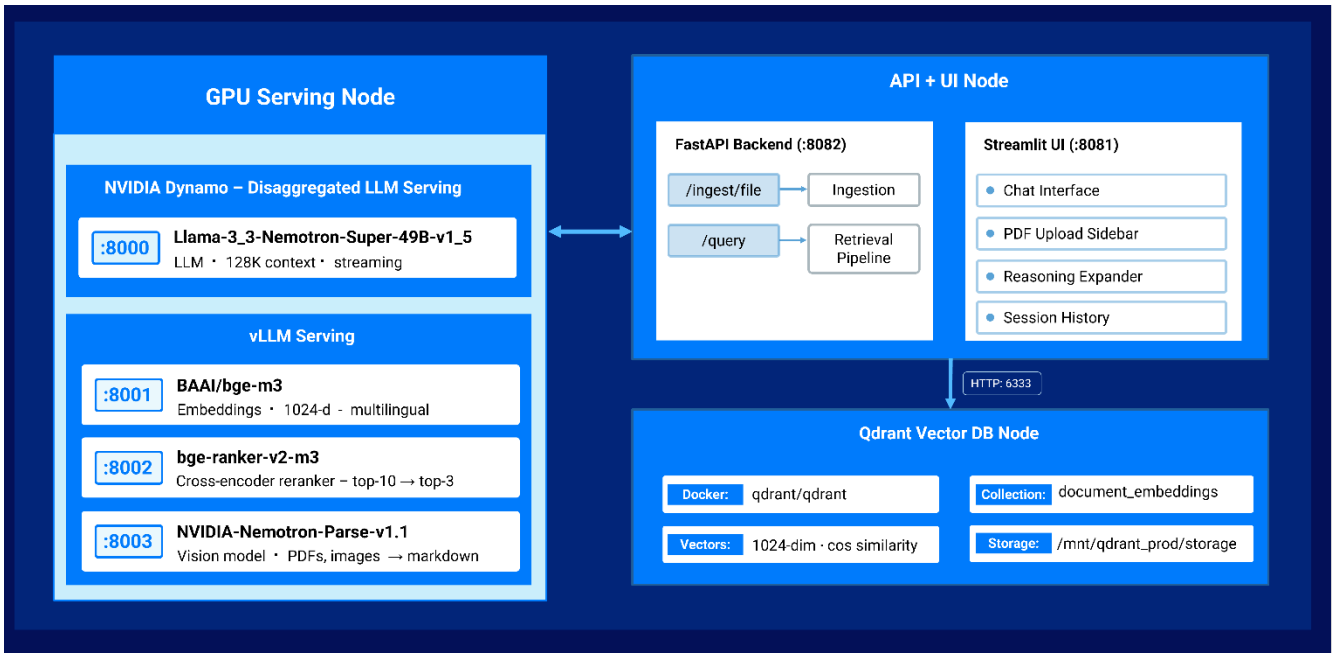


Figure 13 - End-to-End RAG Application Architecture with NVIDIA Dynamo and Qdrant

Models Services and Endpoints

Role	Model	Port	Env Variable	Description
LLM Generation	nvidia/Llama-3_3-Nemotron-Super-49B-v1_5	:8000	DYNAMO_ROUTER_URL	Dynamo disaggregated serving · streaming
Embeddings	BAAI/bge-m3	:8001	EMBED_URL	1024-dimensional dense vectors · multilingual
Reranking	BAAI/bge-reranker-v2-m3	:8002	RERANK_URL	Cross-encoder · top-10 → top-3
PDF Parsing	nvidia/NVIDIA-Nemotron-Parse-v1.1	:8003	PARSE_URL	Vision model · page-by-page JPEG → structured markdown
Vector DB	Qdrant	:6333	QDRANT_URL	Cosine similarity · 1024-dim · persistent storage

Install and Start RAG Services

The FastAPI backend (api.py) uses these environment variables to orchestrate document ingestion and query workflows, exposing REST endpoints on port 8082.

```
source venv/bin/activate
pip install -r requirements.txt
# Configure endpoints
cp .env.example .env
# Edit .env with your actual IPs
```

Sample configuration for *.env*

```
DYNAMO_ROUTER_URL=http://<MODEL_HOST_IP>:8000/v1
EMBED_vLLM_URL=http://<MODEL_HOST_IP>:8001/v1
RERANK_vLLM_URL=http://<MODEL_HOST_IP>:8002
PARSE_vLLM_URL=http://<MODEL_HOST_IP>:8003/v1
QDRANT_URL=http://<QDRANT_HOST_IP>:6333
```

```
LLM_MODEL=nvidia/Llama-3_3-Nemotron-Super-49B-v1_5
EMBED_MODEL=BAAI/bge-m3
RERANK_MODEL=BAAI/bge-reranker-v2-m3
PARSE_MODEL=nvidia/NVIDIA-Nemotron-Parse-v1.1
```

Note: Update .env file with the appropriate IP addresses for all services.

Create and execute the pipeline script “start_pipeline.sh”

```
#start_pipeline.sh
#!/bin/bash
set -e

echo "==> Stopping existing processes..."
pkill -f uvicorn   2>/dev/null || true
pkill -f streamlit 2>/dev/null || true
sleep 1

echo "==> Activating virtual environment..."
source .venv/bin/activate

echo "==> Installing dependencies..."
pip install -q -r requirements.txt

echo "==> Loading environment..."
set -a; source .env; set +a

echo "==> Starting FastAPI backend on :8082..."
nohup uvicorn api:app --host 0.0.0.0 --port 8082 > api.log 2>&1 &
echo $! > api.pid

echo "==> Starting Streamlit UI on :8501..."
nohup streamlit run app/main.py \
  --server.port 8501 \
  --server.address 0.0.0.0 \
  --server.headless true > streamlit.log 2>&1 &
echo $! > streamlit.pid

echo ""
echo "  API:  http://localhost:8082"
echo "  UI:   http://localhost:8501"
echo "  Logs: tail -f api.log | tail -f streamlit.log"
```

```
cd dynamo_rag_pipeline
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
```

```
# Configure endpoints
cp .env.example .env
# Edit .env with your actual IPs

# Start API + Streamlit together
./start_pipeline.sh
```

```
root@nvidia-gpu-compute:~/dynamo_rag_pipeline# ./start_pipeline.sh
Starting FastAPI Backend on port 8082 ...
Starting Streamlit UI on port 8501 ...
```

```
Pipeline is running!
API Backend: http://192.0.2.96:8082
Streamlit UI: http://192.0.2.96:8501
```

```
Check logs with: tail -f api.log or tail -f streamlit.log
```

Expected Output:

- FastAPI backend and Streamlit UI services start successfully

Verify API Availability and OpenAPI Schema

```
curl -s http://<RAG_HOST_IP>:8082/docs

# Or check the OpenAPI schema:

curl -s http://<RAG_HOST_IP>:8082/openapi.json | python3 -m json.tool | head -20
```

Expected Output:

- FastAPI Swagger UI accessible
- JSON response showing API schema

Verify LLM Endpoint

```
import httpx, os, json
from typing import AsyncIterator

LLM_URL   = os.getenv("DYNAMO_ROUTER_URL", "http://192.0.2.101:8000/v1")
LLM_MODEL = os.getenv("LLM_MODEL",         "nvidia/Llama-3_3-Nemotron-Super-49B-v1_5")

SYSTEM_PROMPT = """You are an expert technical assistant with access to a curated knowledge base.

Your job:
- Answer the user's question using ONLY the provided context passages.
- If the context contains the answer, respond with a thorough, well-structured reply using
markdown.
- If the context partially answers the question, share what is available and clearly note what is
missing.
- If the context contains NO relevant information, respond with exactly:
  "I don't have that information in my resources."
- Never fabricate facts or use knowledge outside the provided context."""

async def stream_response(question: str, passages: list[str]) -> AsyncIterator[str]:
    context  = "\n\n---\n\n".join(passages)
    user_msg = f"Context:\n{context}\n\nQuestion: {question}"

    async with httpx.AsyncClient(timeout=120) as client:
        async with client.stream("POST", f"{LLM_URL}/chat/completions", json={
```

```
            "model":       LLM_MODEL,
            "messages": [
                {"role": "system", "content": SYSTEM_PROMPT},
                {"role": "user",   "content": user_msg}
            ],
            "stream":      True,
            "max_tokens":  4096,
            "temperature": 0.1
        }) as resp:
            async for line in resp.aiter_lines():
                if line.startswith("data: ") and "[DONE]" not in line:
                    delta = json.loads(line[6:])
                    token = delta["choices"][0]["delta"].get("content", "")
                    if token:
                        yield token
```

```
curl -s -X POST http://192.0.2.101:8000/v1/chat/completions \
  -H "Content-Type: application/json" \
  -d '{
    "model": "nvidia/Llama-3_3-Nemotron-Super-49B-v1_5",
    "messages": [{"role": "user", "content": "Hello"}],
    "max_tokens": 10
  }'
```

```
# Terminal output:
{
  "id": "chatcmpl-73faeba8-3f82-46b6-9283-4c4580359cad",
  "choices": [
    {
      "index": 0,
      "message": {
        "content": "Hello! It's nice to meet you. How",
        "role": "assistant"
      },
      "finish_reason": "length"
    }
  ],
  "model": "nvidia/Llama-3_3-Nemotron-Super-49B-v1_5",
  "usage": { "prompt_tokens": 16, "completion_tokens": 10, "total_tokens": 26 }
}
```

Endpoint 1 - POST “/ingest/file”

Uploads a PDF and executes the ingestion pipeline (parse → chunk → embed → store), returning the number of indexed chunks.

```
@app.post("/ingest/file")
async def ingest_file(file: UploadFile = File(...), sector: str = "all"):
    contents = await file.read()
    chunks = await parse_pdf(contents, file.filename)
    vectors = await get_embeddings(chunks, input_type="passage")
    count = await upload_chunks(chunks, vectors, sector, file.filename)
```

```
    return {"status": "success", "file": file.filename, "chunks_processed": count}
```

```
curl -X POST "http://<RAG_HOST_IP>:8082/ingest/file?sector=all" \
  -F file=@vultr_gpu_guide.pdf
#Terminal Output:
{
   "status": "success",
   "file":  "Vultr_gpu_guide.pdf",
   "chunks_processed": 38
}
```

API log output during ingestion:

```
INFO:dynamo-rag-api:Ingesting file: vultr_gpu_guide.pdf into sector: all
INFO:httpx:HTTP Request: POST http://192.0.2.101:8003/v1/chat/completions "HTTP/1.1 200 OK"
INFO:httpx:HTTP Request: POST http://192.0.2.101:8003/v1/chat/completions "HTTP/1.1 200 OK"
INFO:dynamo-rag-api:Successfully chunked file into 38 segments.
INFO:httpx:HTTP Request: POST http://192.0.2.101:8001/v1/embeddings "HTTP/1.1 200 OK"
INFO:httpx:HTTP Request: POST http://192.0.2.22:6333/collections/document_embeddings/points
"HTTP/1.1 200 OK"
```

Endpoint 2 - POST "/query"

```
@app.post("/query")
async def query(payload: dict):
    question = payload["question"]
    sector = payload.get("sector", "all")

    async def event_stream():
        query_vec = await get_embeddings([question], input_type="query")
        candidates = await search_chunks(query_vec, sector, top_k=10)
        top_passages = await rerank(question, candidates, top_n=3)
        async for chunk in stream_response(question, top_passages):
            yield f"data: {json.dumps({'content': chunk})}\n\n"
        yield 'data: {"type": "done"}\n\n'

    return StreamingResponse(event_stream(), media_type="text/event-stream")
```

Executes the retrieval pipeline (embed → vector search → rerank → LLM streaming) and returns a streaming response.

Query Pipeline

The query pipeline uses a two-stage retrieval approach to balance performance and accuracy. Vector search provides fast candidate retrieval, while reranking refines results using cross-attention for improved relevance.

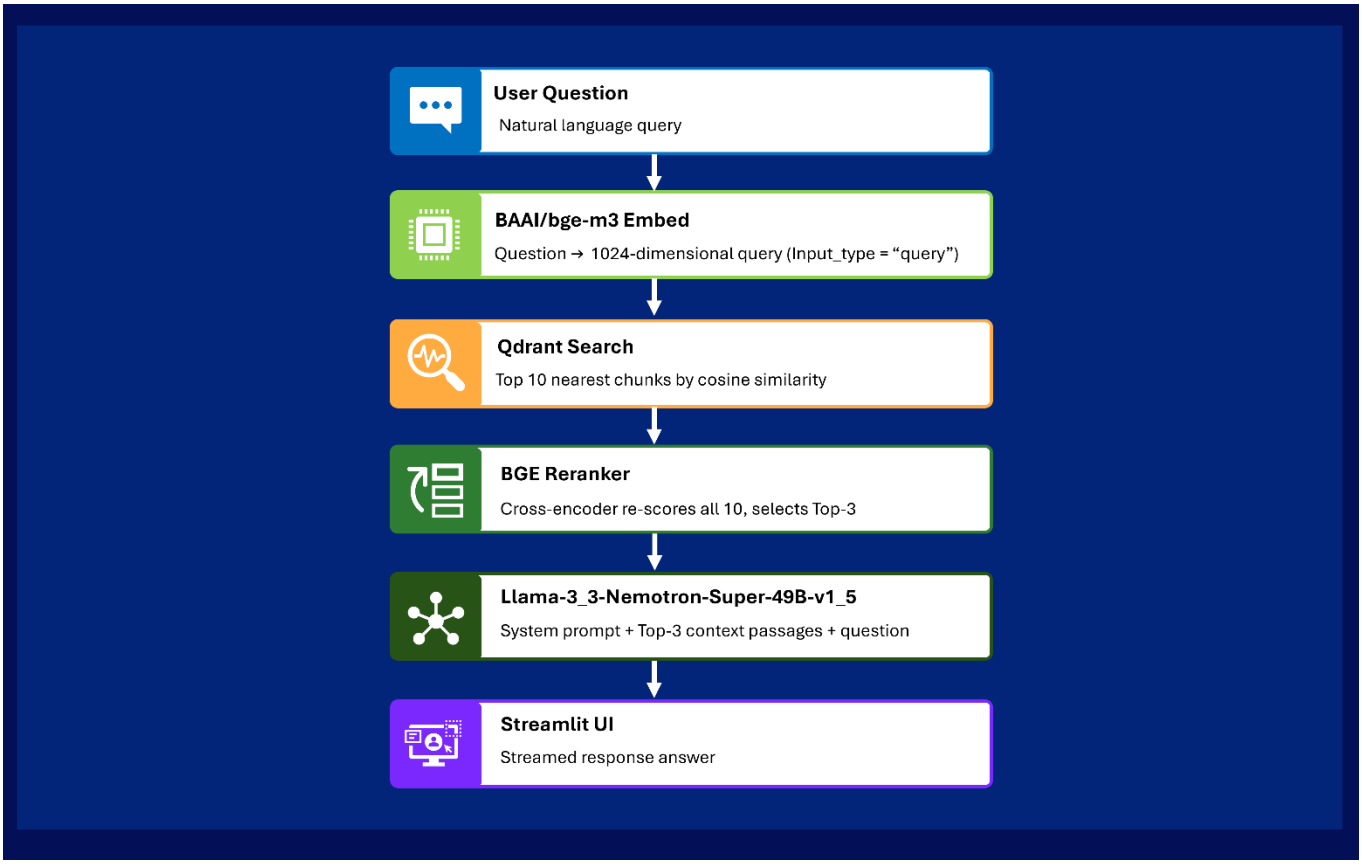


Figure 14 - RAG Query Processing Pipeline with Retrieval, Reranking, and LLM Generation

Stage 1 - Query Embedding

```
import httpx, os

EMBED_URL   = os.getenv("EMBED_URL",   "http://localhost:8001/v1")
EMBED_MODEL = os.getenv("EMBED_MODEL", "BAAI/bge-m3")

async def get_embeddings(texts: list[str], input_type: str = "passage") -> list[list[float]]:
    async with httpx.AsyncClient(timeout=30) as client:
        resp = await client.post(f"{EMBED_URL}/embeddings", json={
            "input":      texts,
            "model":      EMBED_MODEL,
            "input_type": input_type
        })
    data = resp.json()["data"]
    return [item["embedding"] for item in sorted(data, key=lambda x: x["index"])]
```

Stage 2 - Vector search in Qdrant

```
import httpx, uuid, os

QDRANT_URL = os.getenv("QDRANT_URL", "http://localhost:6333")
COLLECTION = "document_embeddings"

async def search_chunks(query_vector: list[float], sector: str, top_k: int = 10) -> list[str]:
    async with httpx.AsyncClient(timeout=15) as client:
        results = await client.post(
            f"{QDRANT_URL}/collections/{COLLECTION}/points/search",
            json={
                "vector":       query_vector,
                "limit":        top_k,
                "with_payload": True
            }
        )
    return [hit["payload"]["text"] for hit in results.json()["result"]]

async def upload_chunks(chunks, vectors, sector, source) -> int:
    points = [
        {
            "id":      str(uuid.uuid4()),
            "vector":  vec,
            "payload": {"text": chunk, "sector": sector, "source": source}
        }
        for chunk, vec in zip(chunks, vectors)
    ]
    async with httpx.AsyncClient(timeout=30) as client:
        await client.put(
            f"{QDRANT_URL}/collections/{COLLECTION}/points",
            json={"points": points}
        )
    return len(points)
```

Stage 3 - Rerank Results (Top-10 → Top-3)

```
import httpx, os

RERANK_URL   = os.getenv("RERANK_URL",   "http://localhost:8002")
RERANK_MODEL = os.getenv("RERANK_MODEL", "BAAI/bge-reranker-v2-m3")

async def rerank(query: str, candidates: list[str], top_n: int = 3) -> list[str]:
    async with httpx.AsyncClient(timeout=30) as client:
        response = await client.post(f"{RERANK_URL}/rerank", json={
            "model":     RERANK_MODEL,
            "query":     query,
            "documents": candidates
        })
    top_3 = [candidates[hit["index"]] for hit in response.json()["results"][:top_n]]
    return top_3
```

Stage 4 - Generate Response (LLM Streaming)

```
import httpx, os, json
from typing import AsyncIterator

LLM_URL   = os.getenv("DYNAMO_ROUTER_URL", "http://192.0.2.101:8000/v1")
LLM_MODEL = os.getenv("LLM_MODEL",         "nvidia/Llama-3_3-Nemotron-Super-49B-v1_5")

SYSTEM_PROMPT = """You are an expert technical assistant with access to a curated knowledge base.

Your job:
- Answer the user's question using ONLY the provided context passages.
- If the context contains the answer, respond with a thorough, well-structured reply using
markdown.
- If the context partially answers the question, share what is available and clearly note what is
missing.
- If the context contains NO relevant information, respond with exactly:
  "I don't have that information in my resources."
- Never fabricate facts or use knowledge outside the provided context."""

async def stream_response(question: str, passages: list[str]) -> AsyncIterator[str]:
    context  = "\n\n---\n\n".join(passages)
    user_msg = f"Context:\n{context}\n\nQuestion: {question}"

    async with httpx.AsyncClient(timeout=120) as client:
        async with client.stream("POST", f"{LLM_URL}/chat/completions", json={
            "model":       LLM_MODEL,
            "messages": [
                {"role": "system", "content": SYSTEM_PROMPT},
                {"role": "user",   "content": user_msg}
            ],
            "stream":      True,
            "max_tokens":  4096,
            "temperature": 0.1
        }) as resp:
            async for line in resp.aiter_lines():
                if line.startswith("data: ") and "[DONE]" not in line:
                    delta = json.loads(line[6:])
                    token = delta["choices"][0]["delta"].get("content", "")
                    if token:
                        yield token
```


Execute RAG Query via API “/query”

```
curl -X POST "http://localhost:8082/query" -H "Content-Type: application/json" -d '{"question": "What models are used?"}'
```

```
# Terminal output (SSE stream):
data: {"content": "<think>\nLet me check the context for information about table extraction"}
data: {"content": " from images on Vultr GPU instances...\n"}
data: {"content": "I can see passage 1 mentions Nemotron-Parse which handles this.\n</think>\n"}
data: {"content": "To extract tables from images using Vultr Cloud GPU, use **NVIDIA Nemotron-Parse**"}
data: {"content": ", a vision model that accepts images and returns structured markdown"}
data: {"content": " including tables, headings, and layout information.\n\n"}
data: {"content": "**Steps:**\n1. Render the image as JPEG\n2. Base64-encode it\n"}
data: {"content": "3. Send to Nemotron-Parse at `:8003/v1/chat/completions`\n"}
data: {"type": "done"}
```

API Log Output During Query

```
INFO:httpx:HTTP Request: POST http://192.0.2.101:8001/v1/embeddings "HTTP/1.1 200 OK"
INFO:httpx:HTTP Request: POST http://192.0.2.22:6333/collections/document_embeddings/points/search "HTTP/1.1 200 OK"
INFO:httpx:HTTP Request: POST http://192.0.2.101:8002/rerank "HTTP/1.1 200 OK"
INFO:httpx:HTTP Request: POST http://192.0.2.101:8000/v1/chat/completions "HTTP/1.1 200 OK"
```

Monitoring

```
tail -f api.log       # FastAPI backend logs
tail -f streamlit.log # Streamlit logs
cat api.pid           # FastAPI PID
cat streamlit.pid     # Streamlit PID
```

Expected Output:

- Application logs for FastAPI backend and Streamlit UI
- Process IDs for running services

Streamlit UI

The Streamlit UI provides a chat-based interface for querying the knowledge base and uploading documents. It communicates with the FastAPI backend on port **8082**.

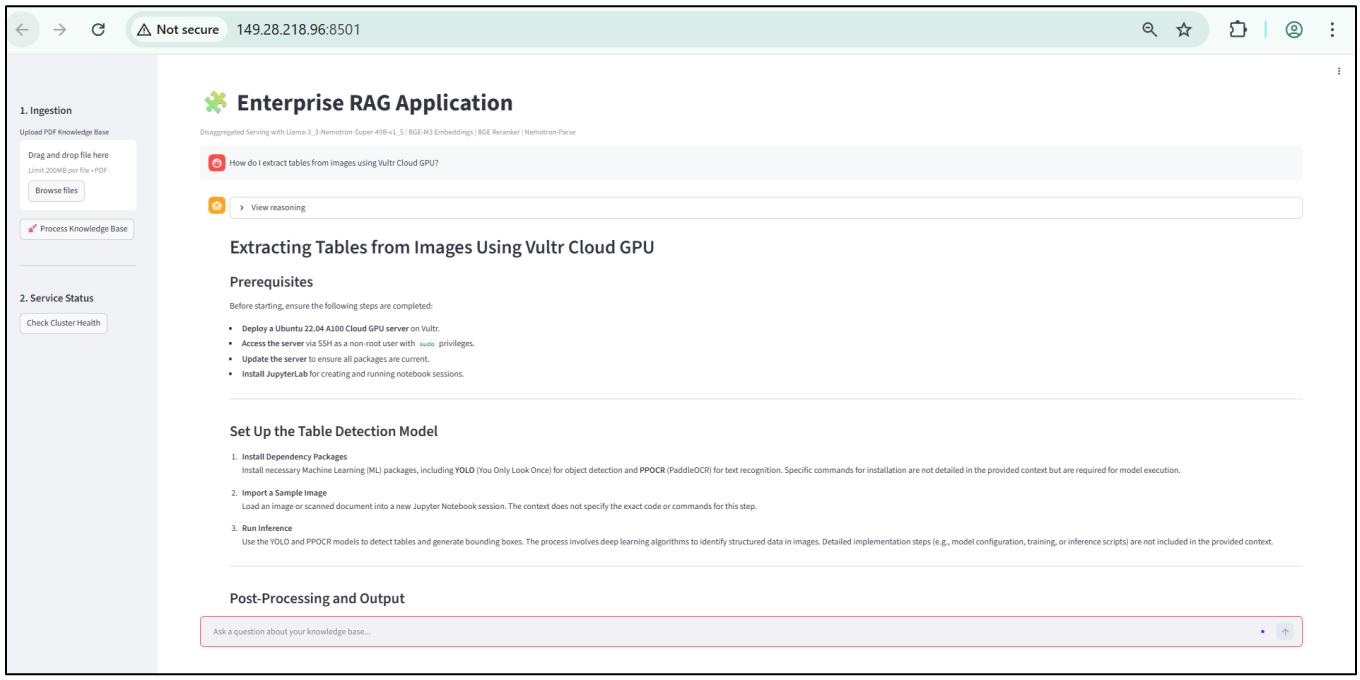
Streamlit UI is already started by the “start_pipeline.sh” script and is running on the port 8501

URL: http://192.0.2.96:8501

Features

Feature	Description
PDF Upload	Upload PDFs from the sidebar - parsed, chunked, embedded, and indexed automatically via /ingest/file
Chat Interface	Ask questions in natural language; responses stream token-by-token in real time
Chat History	Full conversation history is preserved within the session

Application UI



Conclusion

This guide demonstrated an end-to-end workflow for deploying a Retrieval-Augmented Generation (RAG) application using NVIDIA GPUs, NVIDIA Dynamo, and NetApp FlexCache on Vultr Cloud. The deployment enables organizations to build AI-driven applications that interact with enterprise document repositories while leveraging FlexCache for hybrid data access, allowing frequently accessed data to be located closer to GPU resources for accelerated AI workloads.

The solution architecture integrates the following components:

- NetApp ONTAP with FlexCache for governed enterprise document access
- Qdrant for semantic indexing and vector similarity search
- NVIDIA Dynamo for GPU-accelerated inference orchestration and multi-model serving
- vLLM for deploying embedding, reranking, and document parser model services
- NVIDIA Nemotron models for document parsing and large language model inference
- Vultr Cloud providing NVIDIA GPU infrastructure for AI workloads

Together, these technologies provide a scalable architecture for enterprise AI and document intelligence workloads. The platform enables efficient retrieval and generation of knowledge from internal data sources while maintaining existing data governance, storage architecture, and operational controls. The architecture supports both aggregated and disaggregated inference modes, enabling optimization for latency or throughput based on workload requirements.

.