Lenovo Validated Design Hybrid AI 221 Canonical Single-node-RAG

Last update: **18 May 2026**

Version 1.0

Production-Ready Multimodal RAG

Optimized Architecture with NVIDIA Blackwell and NVIDIA AI Enterprise

Enterprise Open-Source Stack with Canonical Ubuntu Linux

Validated Scalability and Performance

Hapsara Sukasdadi
Pierce Beary
Leonardo Queirolo
David Markanich

Table of Contents

1 Introduction

The Lenovo Hybrid AI 221 Canonical Single-node-RAG is a high-performance, compact AI solution designed for enterprises requiring localized, secure, and multimodal Retrieval-Augmented Generation (RAG). The solution leverages the Lenovo Hybrid AI 221 with Canonical platform. Lenovo Hybrid AI 221 with Canonical is a platform that enables enterprises of all sizes to quickly deploy smaller scale but scalable AI infrastructure. Lenovo Hybrid AI 221 supports Enterprise AI use cases as either a new, development, greenfield environment or an extension of their existing IT infrastructure. The high-level architecture is shown in the figure below.

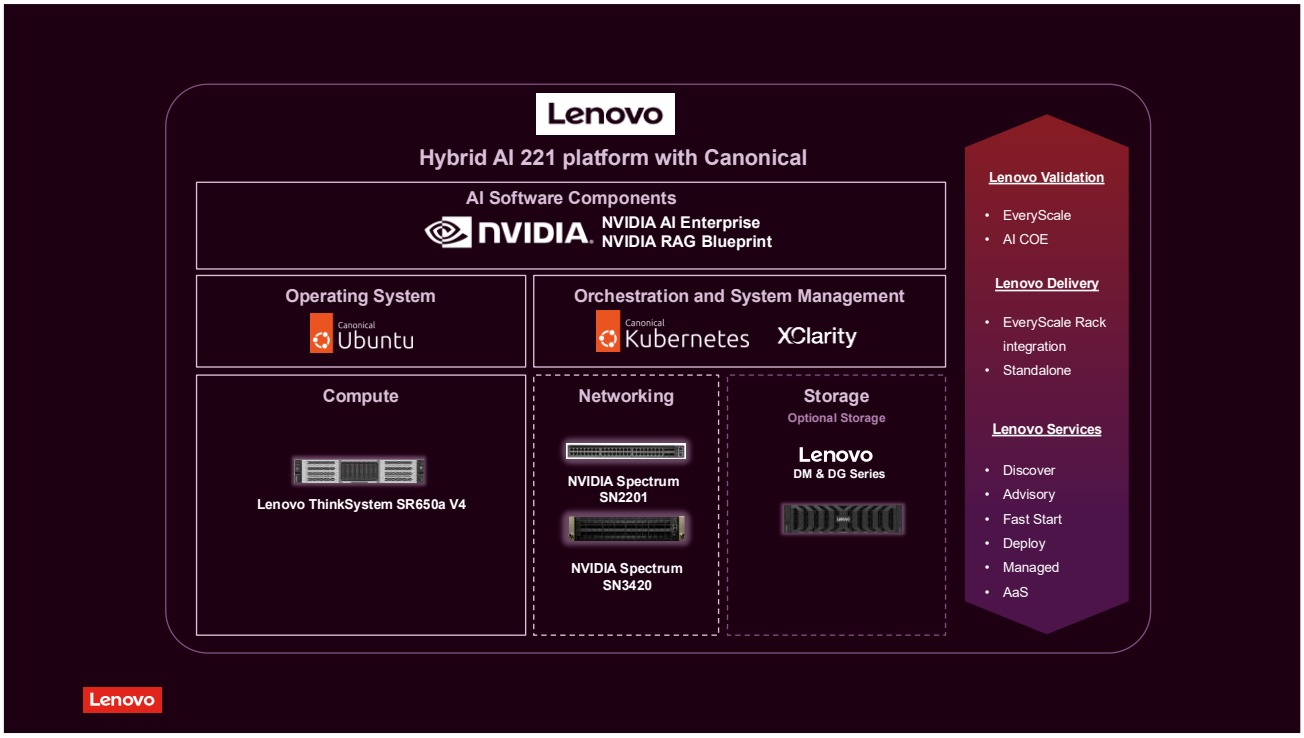


Figure 1: Lenovo Hybrid AI 221 Canonical

The Hybrid AI 221 Canonical platform brings Lenovo ThinkSystem SR650a V4 with NVIDIA RTX PRO 6000 Blackwell Server Edition, Canonical Ubuntu stack with Canonical Kubernetes, and NVDIA AI Enterprise. For the Single-node-RAG solution we are using the NVIDA RAG Blueprint available as part of the NVIDIA AI Enterprise. In summary, this architecture consolidates complex AI pipelines—including document ingestion, vector search, and Large Language Model (LLM) reasoning—into a single physical server.

This LVD provides an in-depth description of the solution architecture including configurations of the components, deployment steps, application user interface, and performance benchmarking.

In this LVD, Networking and Storage are optional and not part of the validation scope.

2 Solution Components

2.1 NVIDIA AI Enterprise (NVAIE)

NVIDIA AI Enterprise is a comprehensive suite of artificial intelligence and data analytics software designed for optimized development and deployment in enterprise settings. This section outlines some of the tools present in NVIDIA AI Enterprise.

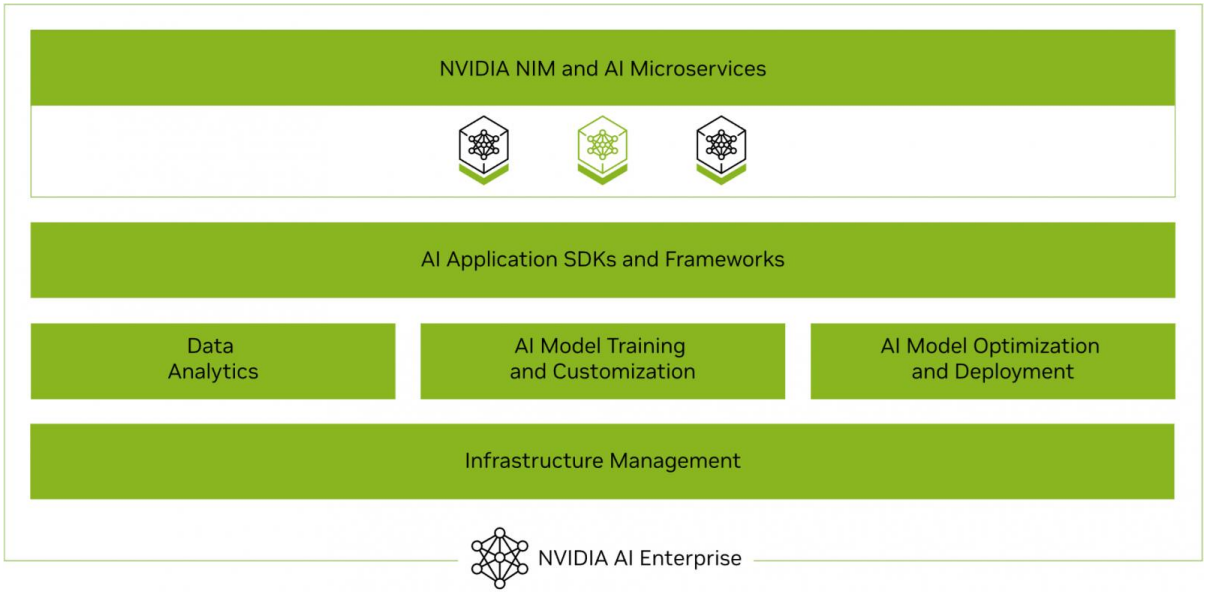


Figure 2: NVIDIA AI Enterprise

NVIDIA AI Enterprise provides access to ready-to-use open-sourced containers and frameworks from NVIDIA like NVIDIA NeMo, NVIDIA RAPIDS, NVIDIA TAO Toolkit, NVIDIA TensorRT and NVIDIA Triton Inference Server.

It also provides full access to the NVIDIA NGC catalogue, a collection of tested enterprise software, services and tools supporting end-to-end AI and digital twin workflows and can be integrated with MLOps platforms such as ClearML, Domino Data Lab, Run:ai, UbiOps, and Weights & Biases. An NVIDIA AI Enterprise License provides access to the fully secured, vetted, and tested software artifacts that are supported by NVIDIA.

2.1.1 NVIDIA Inference Microservice

NVIDIA AI Enterprise introduced NVIDIA Inference Microservices (NIM), a set of performance-optimized, portable microservices designed to accelerate and simplify the deployment of AI models. Those containerized GPU-accelerated pretrained, fine-tuned, and customized models are ideally suited to be self-hosted and deployed on the Lenovo Hybrid AI platforms.

The ever-growing catalog of NIM microservices contains models for a wide range of AI use cases, from chatbot assistants to computer vision models for video processing. The image below shows some of the NIM microservices, organized by use case.

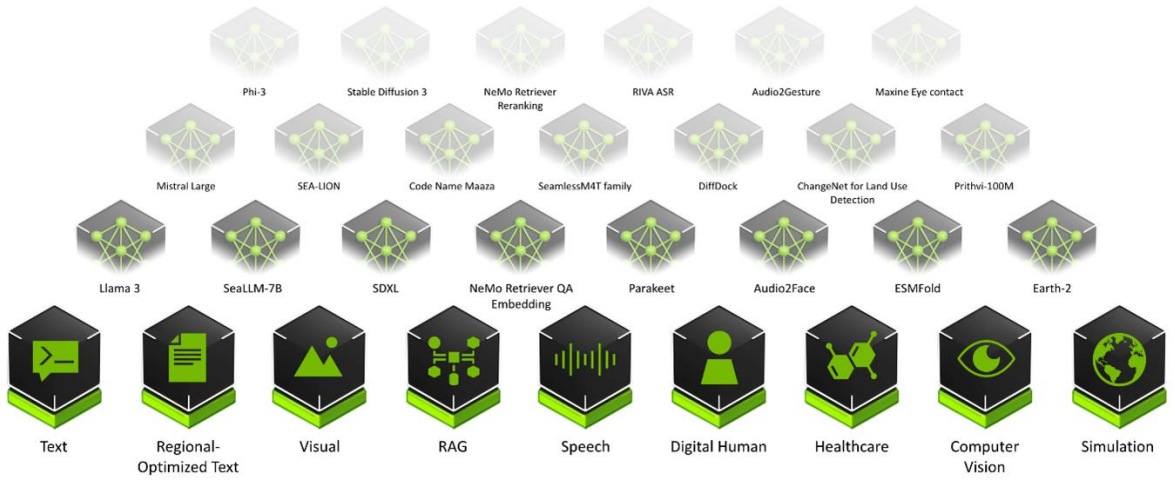


Figure 3: Examples of NIM Catalogue

For an in-depth guide to the NVIDIA Software portfolio with Lenovo Part Numbers, please reference the NVIDIA Software Product Guide on Lenovo Press.

Table 1: NVIDIA AI Enterprise License

Part number	Description
7S02CTO1WW	NVIDIA Software
S6Z3	NVIDIA Enterprise (NVIDIA AI Enterprise and NVIDIA Omniverse Enterprise) Subscription per GPU, 3 Year

2.1.2 NVIDIA RAG Blueprint

NVIDIA RAG Blueprint is a production-ready, modular reference architecture designed to build high-accuracy, high-performance RAG systems that power enterprise search, knowledge assistants, copilots, and agentic workflows at scale. NVIDIA RAG Blueprint transforms raw, multimodal enterprise data into AI-ready knowledge.

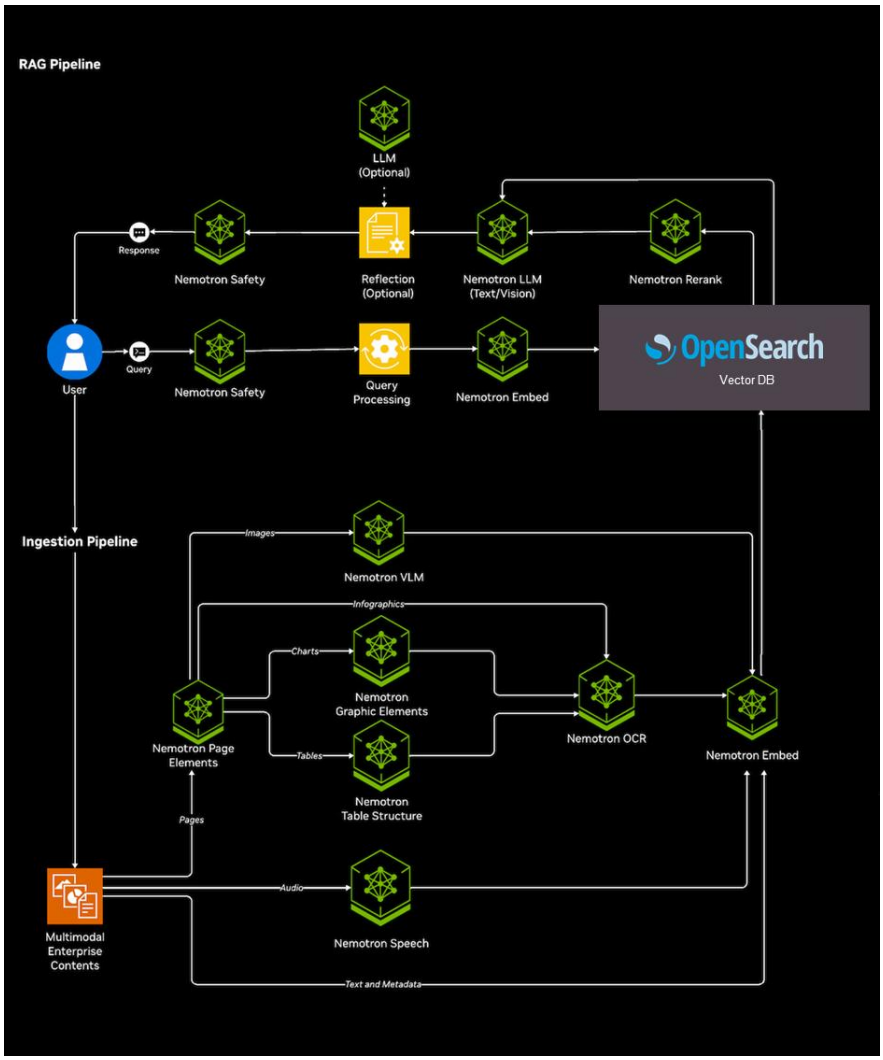


Figure 4: NVIDIA RAG Blueprint with OpenSearch

The RAG Blueprint core capabilities include:

- **Multimodal Ingestion:** It features a robust pipeline that extracts text, tables, charts, images, infographics, and audio/video content from complex enterprise data, which can be enriched with custom metadata to improve downstream retrieval.
- **Advanced Retrieval:** The architecture supports hybrid search (combining dense and sparse retrieval), multi-collection search, GPU-accelerated indexing, and result reranking to improve accuracy. It also features pluggable vector database support, natively integrating with databases like OpenSearch.
- **Reasoning and Generation:** It includes capabilities for shallow and deep document summarization, reasoning-budget configurability, query decomposition, and dynamic metadata filtering, which help AI agents efficiently narrow search spaces and select trusted sources.
- **Vision-Language Support:** The blueprint integrates Vision-Language Models (VLMs) for advanced tasks like image understanding, captioning, and image-aware answer generation, along with optional reflection to further improve answer quality

Software Stack and Integrations The system leverages a suite of **NVIDIA NIM (Inference Microservices)**, utilising models such as Llama-3.3-Nemotron-Super-49B for reasoning and chat, alongside specialised NIMs for query embedding, reranking, OCR, page element detection, and table structure recognition. Furthermore, the sources note that it integrates seamlessly with third-party software like LangChain and Redis Cache. To support modern agent ecosystems, it provides native Python libraries, OpenAI-compatible APIs, MCP server support, and a built-in data catalogue.

Enterprise-Grade Features

- **Scalable Deployment:** The blueprint is highly flexible, fully decomposable, and can be deployed via Docker or Kubernetes with an included user interface. It also supports efficient hardware utilization through GPU sharing via the NIM Operator.
- **Governance:** It offers optional, programmable **NeMo Guardrails** to improve content safety and enforce topic control.
- **Observability and Evaluation:** It includes built-in observability with OpenTelemetry integration and utilizes the **RAGAS (Retrieval Augmented Generation Assessment) framework** evaluation scripts to help development teams measure accuracy, latency, and response groundedness.

In this LVD we are using NVIDIA RAG Blueprint 2.5.0.

2.2 Canonical Software Stack

2.2.1 Ubuntu Linux Operating System

For open source stacks, the AI Compute nodes are typically deployed with Ubuntu Server, which is Canonical's leading open-source Linux operating system optimized for server and infrastructure workloads. It provides a lean, secure, and stable platform with enterprise-grade long-term support (LTS), automated security updates, and strong hardware compatibility. Ubuntu Server 24.04 LTS is used for this LVD.

2.2.2 Canonical Kubernetes

Kubernetes is the open-source platform for automating container deployment, scaling, and management. In on-prem computing environments, Kubernetes offers a flexible and robust foundation for organizing application workloads, allowing organizations to deploy, scale, and manage containers across their own infrastructure. With built-in support for High Availability, Kubernetes enables resilient deployments that minimize downtime. Its self-healing capabilities automatically detect and respond to failures, replacing or restarting containers as needed to ensure applications remain available and performant. This approach empowers IT teams with full control over cluster configuration and operations, enabling customization to meet specific operational or security needs while reducing reliance on vendor-specific solutions.

Canonical Kubernetes is Canonical's distribution of upstream CNCF-conformant Kubernetes. It is delivered as a self-contained snap package that includes the core Kubernetes components plus additional services required for a functional cluster. Canonical Kubernetes can be deployed in multiple ways: as a standalone snap for single-node or clustered operation; via Juju charms with automated, model-driven management; and through Cluster API. In this LVD we are using Canonical Kubernetes with the snap-based deployment.

2.2.3 Ubuntu Pro Subscription

Ubuntu Pro is Canonical's subscription service that enhances Ubuntu with extended security maintenance and enterprise capabilities. It delivers up to 15 years of security updates for the full archive (Main and Universe repositories), kernel livepatching for zero-downtime updates, FIPS-certified cryptography, automated compliance tooling and optional weekday or 24/7 support. For this LVD built on Canonical's open source stack, Ubuntu Pro provides a secure, long-term supported foundation that keeps the operating system, Canonical Kubernetes, and application layer protected and compliant.

Table 2: Ubuntu Pro License

Part number	Description
7S1BCTO1WW	Ubuntu Pro
SCZ3	Canonical Ubuntu Pro 3Yr w/Canonical weekday Support

2.3 Hybrid AI 221 with ThinkSystem SR650a V4

For hardware the Hybrid AI 221 Canonical Single-node RAG solution utilizes ThinkSystem SR650a V4. The Lenovo ThinkSystem SR650a V4 is an ideal 2-socket 2U rack server for customers want to maximize GPU compute power while still retaining the traditional 2U rack form factor. With two Intel Xeon 6700-series or 6500-series processors, plus two RTX PRO 6000 Blackwell Server Edition GPUs. The server supports 2-2-1 and 2-2-3 configuration in a single node. For scaling up, horizontal scaling by adding additional nodes is required.

The Lenovo ThinkSystem SR650a V4 is designed to accelerate GPU-intensive workloads like AI, deep learning, and HPC. Powered by Intel® Xeon® 6 processors, it delivers exceptional GPU density, advanced storage, and PCIe Gen5 connectivity for peak performance. It supports up to 4x double-width or 8x single-width GPUs and offers up to 8x E3.S NVMe drives for fast, low-latency data access.

Figure 5: Lenovo ThinkSystem SR650a V4

The following figure shows the locations of key components inside the server.

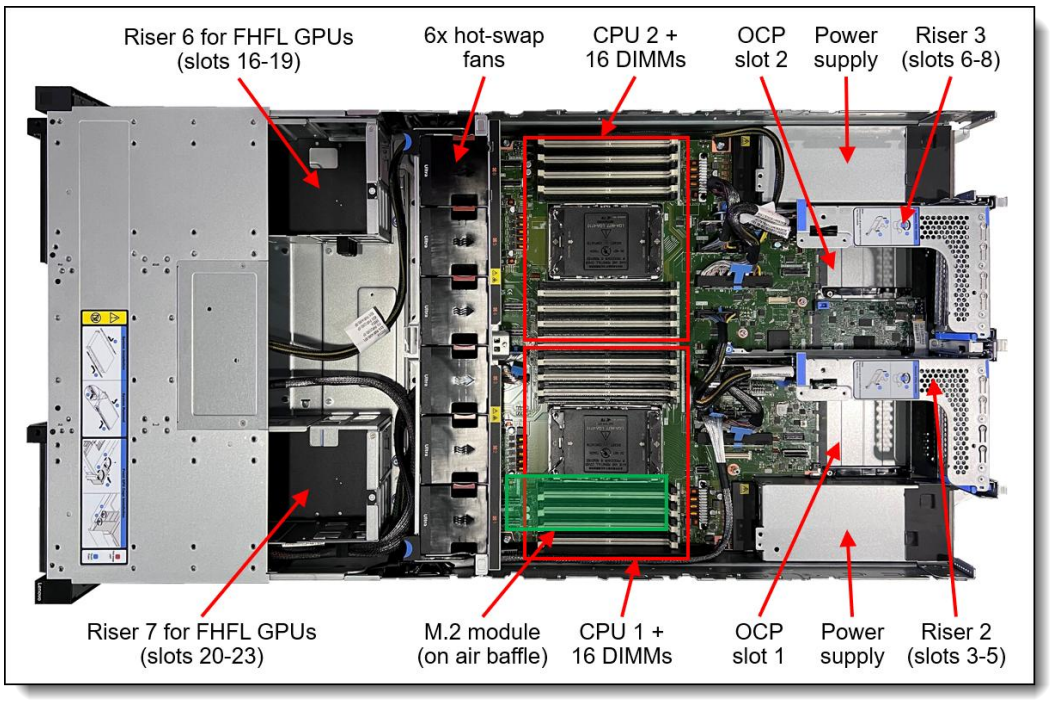


Figure 6: Internal view of the ThinkSystem SR650a V4

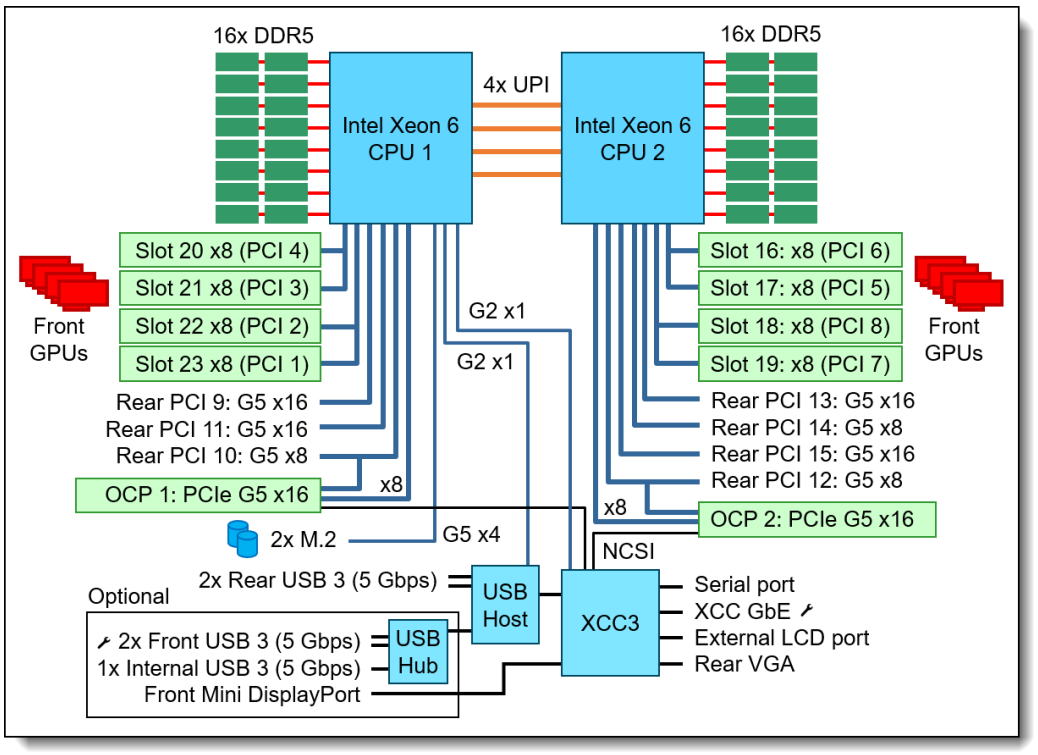


Figure 7: SR650a V4 system architectural block diagram

The AI Compute node is configured with two Intel Xeon 6767P 64C 350W 2.4GHz processors. With 8 Memory Channels per processor socket the Intel based server provides superior Memory bandwidth ensuring highest performance. Leveraging 64GB 6400MHz Memory DIMMs for a total of 1TB main memory providing 192GB memory per GPU or a minimum of 1.5X the H200 NVL GPU. For 2 GPUs the minimum requirement is 576 GB

of main memory.

The Hybrid AI 221 Canonical Single-node RAG utilizes NVIDA RTX PRO 6000 Blackwell Server Edition GPUs. The GPU is built on the groundbreaking NVIDIA Blackwell architecture; the NVIDIA RTX PRO 6000 Blackwell Server Edition delivers a powerful combination of AI and visual computing capabilities to accelerate enterprise data center workloads. Equipped with 96GB of ultra- fast GDDR7 memory, the NVIDIA RTX PRO 6000 Blackwell provides unparalleled performance and flexibility to accelerate a broad range of use cases- from agentic AI, physical AI, and scientific computing to rendering, 3D graphics, and video.

For the detailed Bill-of-Material for hardware and software Please refer to Lenovo Hybrid AI 221 Platform Guide.

3 Solution Architecture

The Hybrid AI 221 Canonical Single-node-RAG is based on the NVIDIA RAG Blueprint customized to run on Lenovo ThinkSystem SR650a V4 with 2x NVIDIA RTX PRO 6000 Blackwell GPUs, deployed on top of Ubuntu OS and Canonical Kubernetes.

The solution replaces the default Milvus vector database with OpenSearch (snap), assigns workloads across a custom MIG layout, and adds a standalone Triton Inference Server for general-purpose ML inference.

3.1 Ubuntu Linux and Canonical Kubernetes Configuration

The Hybrid AI 221 Canonical Single-node-RAG solution is built upon the foundation of **Ubuntu Linux 24.04**. Chosen for its enterprise OS stability, Ubuntu provides the reliable host environment necessary for executing high-density AI workloads. It serves as the critical base layer where the NVIDIA open kernel drivers and CUDA Toolkit are installed, and it natively hosts the bare-metal OpenSearch vector database as an isolated snap package for maximum storage performance

Container orchestration for the AI application layer is handled by **Canonical Kubernetes**, deployed as a streamlined, secure **snap package running version 1.32**. Within this architecture, Canonical Kubernetes is configured as a highly optimized single-node cluster tailored specifically for the single-box footprint. The deployment automatically provisions essential cluster services, including internal DNS, local-storage management, and a MetalLB load balancer operating in L2 mode, which is used to expose the RAG application's frontend to end users

3.2 Software Stack

The NVIDIA RAG Blueprint provides a complete foundation for ingestion, retrieval, reasoning, and generation across multimodal enterprise data. The components are intelligently distributed across the CPU, bare metal, and specific GPU partitions:

- **CPU Workloads:** The CPU handles query orchestration via the RAG Server, document ingestion via the Ingestor Server and nv-ingest/ NeMo Retriever Extraction pipeline (utilizing Ray), and message brokering via Redis.
- **Bare Metal Workloads:** OpenSearch runs as a bare-metal snap (outside of Kubernetes) to provide the vector store (k-NN) and BM25 full-text search capabilities. A Kubernetes Service and Endpoints bridge connects the internal cluster to the host-based OpenSearch instance.
- **GPU Workloads:** Complex model inference tasks are processed by the NVIDIA GPUs, which include chat generation, query and document embedding, result re-ranking, layout detection, and OCR.

The complete component mapping is shown in the table below.

Table 3: Software Component Mappings

Component	Role	Runs On
NIM LLM (Llama-3.3-Nemotron-Super-49B-v1.5)	Chat completion, answer generation	GPU 0 (full, non-MIG)

NIM Embedding (llama-nemotron-embed-1b-v2)	Query and document embedding	GPU 1 MIG 1g.24 GB
NIM Reranking (llama-nemotron-rerank-1b-v2)	Result re-ranking	GPU 1 MIG 1g.24 GB
NIM OCR (nemoretriever-ocr-v1)	Text extraction from scanned pages	GPU 1 MIG 2g.48 GB
NIM Page Elements (nemotron-page-elements-v3)	Layout detection (tables, charts, text)	GPU 1 MIG 2g.48 GB
NIM Graphic Elements (nemotron-graphic-elements-v1)	Chart and graphic detection	GPU 1 MIG 1g.24 GB
NIM Table Structure (nemotron-table-structure-v1)	Table cell/row recognition	GPU 1 MIG 1g.24 GB
RAG Server	Query orchestration (embed, search, rerank, LLM)	CPU
Ingestor Server	Document ingestion API	CPU
nv-ingest/NeMo Retriever Extraction	Document processing pipeline (Ray, CPU)	CPU
OpenSearch	Vector store (k-NN) + BM25 full-text search	Bare metal (snap)
Redis	Message broker for nv-ingest	CPU
Triton Inference Server	General ML inference (ONNX, TensorRT, etc.)	GPU 1 MIG 2g.48 GB

3.2.1 OpenSearch Vector Database

In the Canonical Single-node-RAG solution, OpenSearch replaces the default Milvus vector database found in the upstream NVIDIA RAG Blueprint. Rather than running inside a container, OpenSearch is deployed natively as a baremetal snap package directly on the Ubuntu host, which avoids Kubernetes overhead and maximizes storage performance for optimized vector and hybrid search operations.

The database is configured securely with TLS, an admin password, and the k-NN plugin enabled to support advanced vector storage. To ensure high availability and persistence, the OpenSearch service is also configured to automatically survive system reboots.

To connect this bare-metal database with the containerised AI workloads, a Kubernetes Service and Endpoints bridge is created during the infrastructure setup. This bridge makes the host-based database securely reachable at opensearch:9200 from within the internal Kubernetes cluster, allowing the application layers to communicate with it using self-signed certificates.

At the application layer, the integration leverages the NVIDIA "Define Your Own Vector Database" pattern. A custom OpenSearch operator and VectorClient implementation were developed in Python (opensearch_vdb.py) to manage critical database functions. This operator handles index lifecycles, document upserts and deletions, k-NN vector searches, and BM25 full-text searches for robust hybrid retrieval. Custom query builders map these operations to OpenSearch syntax to support complex metadata filtering.

To ensure the system utilizes these capabilities, this custom operator code is dynamically injected into customized container images (lenovo/rag-server and lenovo/ingestor-server) during the application deployment phase, seamlessly activating OpenSearch when VDB_TYPE=opensearch is defined in the system configuration.

3.2.2 Redis Message Broker

Within the Canonical Single-node-RAG architecture, Redis is deployed as a critical third-party infrastructure component of the underlying NVIDIA RAG Blueprint software stack. It functions to ensure optimal resource distribution and preserve valuable VRAM for AI model inference, Redis is allocated entirely to the host server's CPU. It runs alongside other CPU-bound workloads such as the RAG Server, the Ingestor Server API, and the Ray orchestration engine.

Its primary function in this deployment is to act as the message broker for the nv-ingest document processing pipeline. When users upload complex multimodal enterprise data (such as PDFs containing dense text, tables, charts, and images), the system must orchestrate multiple extraction tasks across various specialized microservices. Redis manages the high-throughput communication queues between the CPU-based ingestor services and the GPU-accelerated document NIMs (such as NeMo Retriever OCR and Nemotron Page Elements). This robust messaging layer ensures that ingestion workflows remain coordinated, stable, and scalable under load.

Additionally, Redis operates as a caching mechanism within the broader application framework. Working in tandem with orchestration tools like LangChain, it helps manage multi-turn conversations and intermediate retrieval states, optimizing the overall responsiveness of the RAG platform.

3.2.3 Triton Inference Server

In the Canonical Single-node-RAG solution, the Triton Inference Server is not actually a core component of the RAG pipeline itself. Instead, it functions as a standalone deployment integrated alongside the RAG operations to handle general-purpose Machine Learning (ML) inference tasks, such as running ONNX or TensorRT models.

Its primary purpose is to add versatility to the server, allowing the hardware to support other non-RAG ML workloads simultaneously.

Key details about its implementation include:

- Resource Sharing: It is deployed in its own triton namespace and shares a 2g.48 GB Multi-Instance GPU (MIG) slice on GPU 1. It uses CUDA time-slicing to share compute cycles with the NIM OCR and NIM Page Elements microservices.
- Dynamic Model Loading: While it comes pre-loaded with a densenet_onnx quickstart model for immediate testing, users can deploy additional custom models at any time by copying them into the

triton-model-repo Persistent Volume Claim (PVC). Triton operates in a polling mode and will automatically detect and load the new models

3.2.4 Software Stack Matrix

The Software stack matrix is shown in the table below.

Table 4: Software Stack Matrix

Component	Version	Provider
Ubuntu	24.04 LTS (Ubuntu Pro)	Canonical
Canonical K8s (snap)	1.32	Canonical
OpenSearch (snap)	2/stable	Canonical
NVIDIA Drivers (open kernel)	595.58.03	NVIDIA
CUDA Toolkit	13	NVIDIA
GPU Operator	v26.3.0	NVIDIA
NVIDIA RAG Blueprint	2.5.0	NVIDIA
NIM Operator	3.1.0	NVIDIA
NIM Containers	2.0.x	NVIDIA
Triton Inference Server	26.03	NVIDIA

3.3 GPU Optimization and Allocation Strategy

To successfully run a complete multimodal RAG pipeline on the two GPUs, the solution utilizes a customized hybrid MIG and time-slicing layout.

- **GPU 0:**
  - **Allocation:** Full GPU (96 GB VRAM) with no MIG and no time-slicing.
  - **Workload:** NIM LLM (Llama-3.3-Nemotron-Super-49B-v1.5).
  - **Rationale:** Large Language Models demand maximum VRAM and compute resources to ensure low-latency chat completions.
- **GPU 1**: Runs in MIG mode, divided into three hardware-isolated slices, providing memory and streaming multiprocessor (SM) isolation. Within each slice, multiple Kubernetes pods share the hardware via CUDA time-slicing by context-switching.
  - **Slice 1 (1g.24 GB):** Allocated for NIM Embedding (llama-nemotron-embed-1b-v2) and NIM Reranking (llama-nemotron-rerank-1b-v2).
  - **Slice 2 (1g.24 GB):** Allocated for NIM Graphic Elements (nemotron-graphic-elements-v1) and NIM Table Structure (nemotron-table-structure-v1).

- **Slice 3 (2g.48 GB):** Allocated for heavier workloads including NIM OCR (nemoretriever-ocr-v1), NIM Page Elements (nemotron-page-elements-v3), and the Triton Inference Server.

The GPU configuration details are shown below:

```
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 595.58.03              Driver Version: 595.58.03      CUDA Version: 13.2     |
+-----------------------------------------+------------------------+----------------------+
| 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 RTX PRO 6000 Blac...    On  |   00000000:6C:00.0 Off |                    0 |
| N/A   39C    P0             86W /  600W |   93205MiB /  97887MiB |      0%      Default |
|                                         |                        |             Disabled |
+-----------------------------------------+------------------------+----------------------+
|   1  NVIDIA RTX PRO 6000 Blac...    On  |   00000000:D8:00.0 Off |                   On |
| N/A   39C    P0             89W /  600W |   47105MiB /  97887MiB |     N/A      Default |
|                                         |                        |             Enabled |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| MIG devices:                                                                            |
+------------------+--------------------------------+-----------+-----------------------+
| GPU  GI  CI  MIG |              Shared Memory-Usage |        Vol|        Shared         |
|      ID  ID  Dev |                Shared BAR1-Usage | SM     Unc| CE ENC  DEC  OFA  JPG |
|                  |                                 |        ECC|                       |
|==================+================================+===========+=======================|
|  1    1   0   0  |           28451MiB / 48512MiB    | 94      0 |  2   2    2    0    2 |
|                  |               0MiB / 16654MiB    |           |                       |
+------------------+--------------------------------+-----------+-----------------------+
|  1    5   0   1  |            9245MiB / 24192MiB    | 46      0 |  1   1    1    0    1 |
|                  |               0MiB /  8327MiB    |           |                       |
+------------------+--------------------------------+-----------+-----------------------+
|  1    6   0   2  |            9411MiB / 24192MiB    | 46      0 |  1   1    1    0    1 |
|                  |               0MiB /  8327MiB    |           |                       |
+------------------+--------------------------------+-----------+-----------------------+
```

3.4 Canonical Single-node-RAG Pipeline

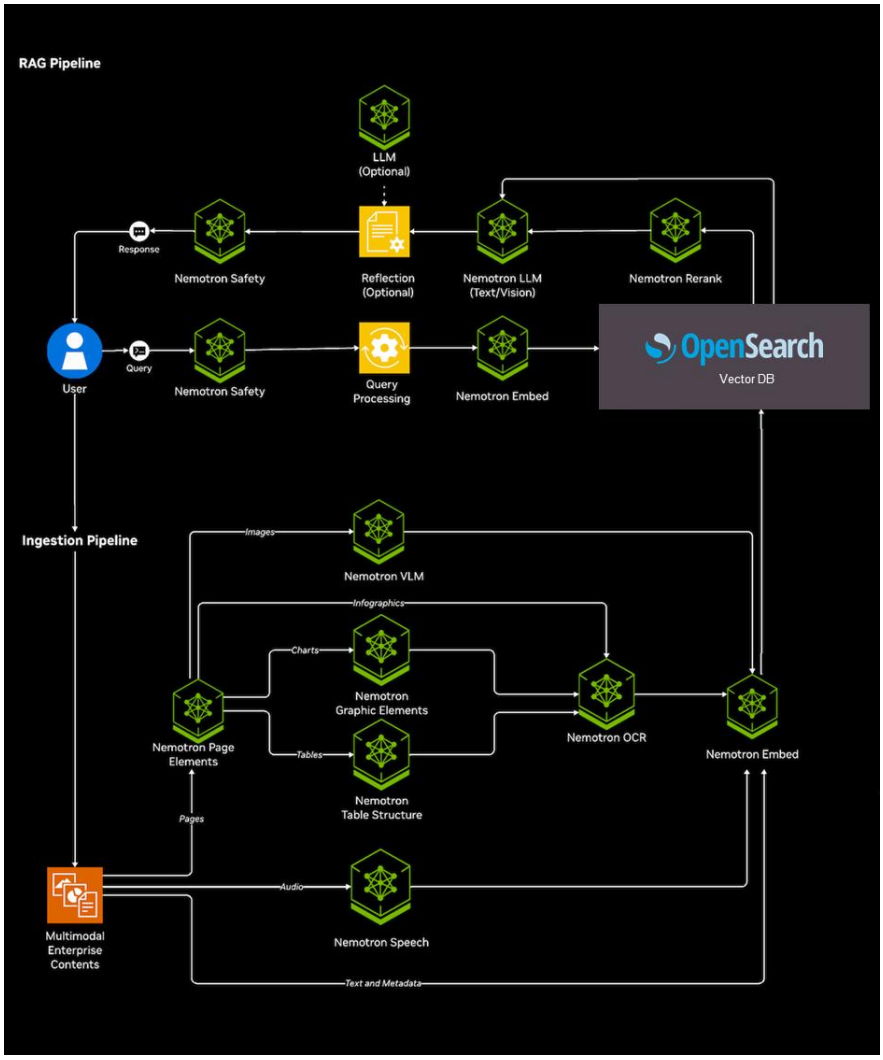


Figure 8: NVIDIA RAG Blueprint with OpenSearch

The high level diagram of the pipeline is shown in Figure 8. The Canonical Single-node-RAG pipeline leverages the NVIDIA RAG Blueprint 2.5.0 to handle end-to-end multimodal document ingestion, semantic search, and AI-driven generation. The pipeline strategically distributes orchestration tasks to the CPU while accelerating intensive AI tasks via targeted NVIDIA Inference Microservices (NIMs).

3.4.1 Multimodal Data Ingestion

The ingestion phase is managed by the CPU-bound Ingestor Server and nv-ingest/NeMo Retriever Extraction document processing pipeline. Orchestrated by Ray and utilizing Redis as a high-throughput message broker, this pipeline is designed to dissect and process complex enterprise data formats, including dense PDFs, audio, and video files.

During ingestion, the pipeline is capable of extracting rich content—such as text, tables, charts, images, and infographics—and enriching it with custom metadata to improve downstream retrieval. To achieve this, nv-ingest routes specific parsing tasks to a suite of specialized vision and layout NIMs running on hardware-

isolated Multi-Instance GPU (MIG) slices on GPU 1.

- **NIM OCR** (nemoretriever-ocr-v1) extracts text from scanned pages.
- **NIM Page Elements** (nemotron-page-elements-v3) detects overarching layouts like tables, charts, and text blocks.
- **NIM Graphic Elements** (nemotron-graphic-elements-v1) detects specific charts and graphics.
- **NIM Table Structure** (nemotron-table-structure-v1) recognizes intricate table cells and rows.

3.4.2 Hybrid Retrieval and Reranking

When a user submits a question, the query workflow is orchestrated by the **RAG Server**, which runs on the host CPU.

1. **Embedding:** The query is first sent to the **NIM Embedding** service (llama-nemotron-embed-1b-v2), running on a 1g.24 GB MIG slice on GPU 1, to be converted into a dense vector representation.
2. **Hybrid Search:** The vectorized query is executed against the bare-metal **OpenSearch** database. To maximize context accuracy, the solution employs **hybrid search**, meaning it executes both a k-NN dense vector search and a BM25 sparse full-text search simultaneously. The retrieval system also utilizes advanced features like multi-collection searching, query decomposition, and dynamic metadata filtering to aggressively narrow the search space to the most relevant sources.
3. **Reranking:** Because hybrid search can return a wide array of document chunks, the results are passed to the **NIM Reranking** service (llama-nemotron-rerank-1b-v2). This model re-evaluates and re-orders the retrieved context, pushing the most highly relevant chunks to the top to improve the final answer's groundedness.
4. **Generation:** Finally, the optimally retrieved and reranked context is packaged alongside the user's original prompt and sent to the **NIM LLM** (Llama-3.3-Nemotron-Super-49B-v1.5). Operating on the dedicated, unpartitioned GPU 0, this large language model performs the final reasoning and chat completion to generate the trusted response

3.5 Automation Scripts

Here are the list of files and scripts used to automate the solution deployment:

```
lenovo_nvidia_rag_blueprint/
  config_infra.env           # Shared configuration (hardware, MIG, OpenSearch)
  setup_infra.sh             # Phase 1: Infrastructure (drivers, K8s, GPU Operator, OpenSearch)
  deploy_apps.sh             # Phase 2: Applications  (images, Helm chart, Triton)
  cleanup.sh                 # Full uninstall (snaps, drivers, K8s, credentials)
  rag_blueprint_lenovo/      # Customized RAG Blueprint (Helm chart, VDB code, Dockerfiles)
  triton/
    triton-server.yaml       # Standalone Triton deployment (K8s manifests)=
```

- **setup_infra.sh**: This script is used to install drivers, Canonical Kubernetes, GPU Operator, and OpenSearch database. It is a two-stage script, that needs a system restarts in the middle of the script run.

- **deploy_apps.sh**: This script is used to install all the applications such as images, Helm chart, and Triton inference server.
- **cleanup.sh**: Use this script to initiate full uninstall of the deployment.

Please consult your Lenovo representative to get the automation scripts.

3.6 Customization vs Upstream Blueprint

The upstream NVIDIA RAG Blueprint 2.5.0 ships with Milvus as the vector database and assumes a homogeneous multi-GPU setup. This bundle makes three categories of changes. No upstream Helm templates were modified.

- OpenSearch replaces Milvus as the vector database.
- Custom GPU layout maps all workloads across a 2-GPU MIG configuration.
- Custom container images add the OpenSearch VDB operator (built automatically by `deploy_apps.sh`).
- Triton Inference Server runs as a standalone deployment sharing a MIG slice.

4 Validation Process and Results

4.1 Installation

This section describes the automated installation process validation of Single-node-RAG using the provided scripts. For more details on the script installation process with console logs, please refer to Appendix A.

4.1.1 Prerequisites

1. Ubuntu 24.04 LTS installed on the server
2. Root (sudo) access
3. Internet connectivity (for container image pulls and model downloads)
4. An NGC API key with access to NIM containers on

4.1.2 Step 1: Deploy Infrastructure

The script **setup_infra.sh** script installs NVIDIA drivers, OpenSearch, Canonical K8s, the GPU Operator with MIG configuration, and creates the OpenSearch K8s bridge. The script will prompt for a reboot after installing NVIDIA drivers on the first run. After rebooting, re-run the same command to continue from where it left off. The script is idempotent. The step-by-step installation process of the script is described below.

Table 5: setup_infra.sh installation steps

#	What it does
1	NVIDIA drivers (open kernel 595.58.03) + CUDA 13
2	OpenSearch snap (TLS, admin password, k-NN plugin)
3	Canonical K8s 1.32 (single-node, DNS, local-storage, LB)
4	GPU Operator v26.3.0 + MIG configuration (mig-manager)
5	Load balancer (MetalLB / k8sd, L2 mode)
6	K8s namespaces, OpenSearch Service+Endpoints bridge, secrets

4.1.3 Step 2: Deploy Applications

The **deploy_apps.sh** script builds the custom images, installs the NIM Operator, deploys the RAG Blueprint via Helm, and deploys Triton. The script will prompt for the NGC API key if not already saved. On subsequent runs it detects the existing deployment and offers to upgrade. The step-by-step installation process of the script is described below.

Table 6: deploy_apps.sh installation steps

#	What it does
1	Validate prerequisites (K8s, GPU Operator, OpenSearch bridge)
2	NGC credentials (prompt or load from store)
3	Build custom images with buildah (OpenSearch VDB support)

4	Build Helm chart dependencies
5	Install NIM Operator (CRDs + controller)
6	Helm install/upgrade of the RAG Blueprint
7	Deploy Triton Inference Server into `triton` namespace
8	Post-deploy status and pod readiness check

4.1.4 Wait for NIM images

NIM container images are large (10-49 GB each). After deploy_apps.sh script completes, pods will be in ContainerCreating or Pending state while images are pulled. Monitor the progress by running:

k8s kubectl get pods -n rag -w

k8s kubectl get pods -n triton -w

```
root@tag-1043:/home/lqueirolo# root@tag-1043:/home/lqueirolo# k8s kubectl get pods -n rag -w
NAME                                                READY   STATUS    RESTARTS   AGE
ingestor-server-b8796745f-hr2gn                     1/1     Running   0          2d1h
nemoretriever-graphic-elements-v1-5f5fdfc45b-4hwqm  1/1     Running   0          2d1h
nemoretriever-ocr-v1-68d665474-h8x7k                1/1     Running   0          2d1h
nemoretriever-page-elements-v3-5b8957659d-8qd4h     1/1     Running   0          2d1h
nemoretriever-table-structure-v1-5668d78f68-hh2k8   1/1     Running   0          2d1h
nemotron-embedding-ms-5879b59d8b-92fgd              1/1     Running   0          2d1h
nemotron-ranking-ms-8948b4fd5-dl8g9                 1/1     Running   0          2d1h
nim-llm-86cffbc4b6-s4hp2                            1/1     Running   0          2d1h
rag-frontend-86ccc4f9b9-5cq5t                       1/1     Running   0          2d1h
rag-nv-ingest-79754bdb59-p4kg5                      1/1     Running   0          2d1h
rag-redis-master-0                                  1/1     Running   0          2d1h
rag-redis-replicas-0                                1/1     Running   0          2d1h
rag-server-568ccf68c-4qzw4                          1/1     Running   0          2d1h

root@tag-1043:/home/lqueirolok8s kubectl get pods -n triton -w-w
NAME                                      READY   STATUS    RESTARTS   AGE
triton-inference-server-6f7d96c96-bgmmf   1/1     Running   0          2d1h
^Croot@tag-1043:/home/lqueirolo#
```

4.2 Retrieval-Augmented Generation (RAG) Functional Test

This section described the RAG use case validation, utilizing the default NVIDIA RAG Blueprint's frontend.

4.2.1 Access Front End

The frontend service is exposed via a LoadBalancer. Once all pods are running, get the external IP:

```
k8s kubectl get svc rag-frontend -n rag
```

Open `http://<EXTERNAL-IP>:3000` in a browser to access the Front-End home page

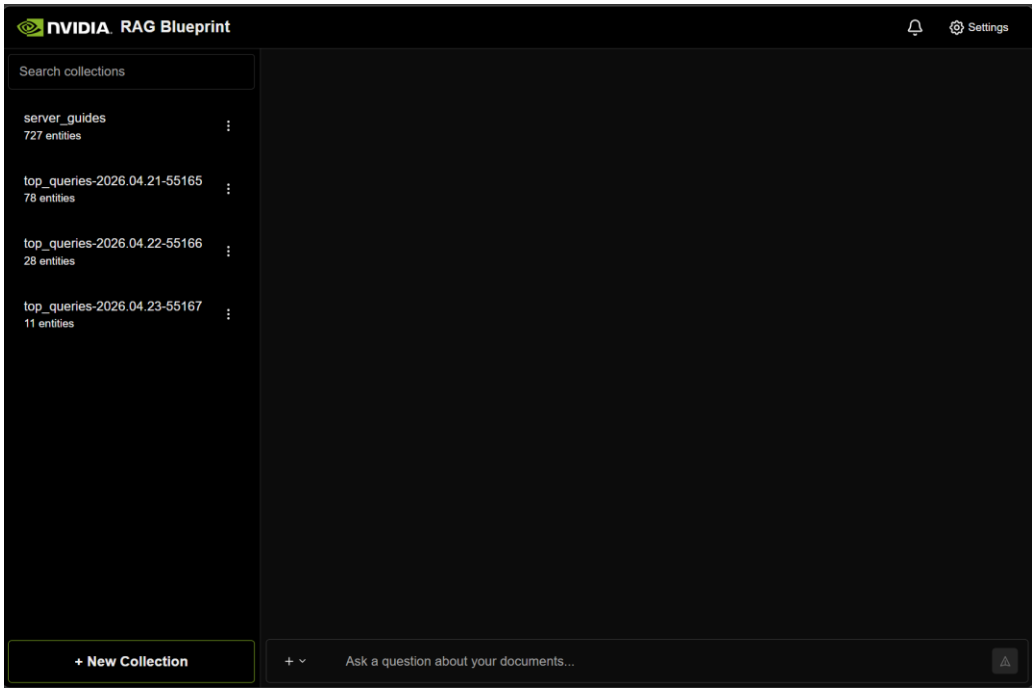


Figure 9: Front End Home Page

4.2.2 Document Ingestion

Document Ingestion is done by clicking on the create “New Collection” from the Front-End home page.

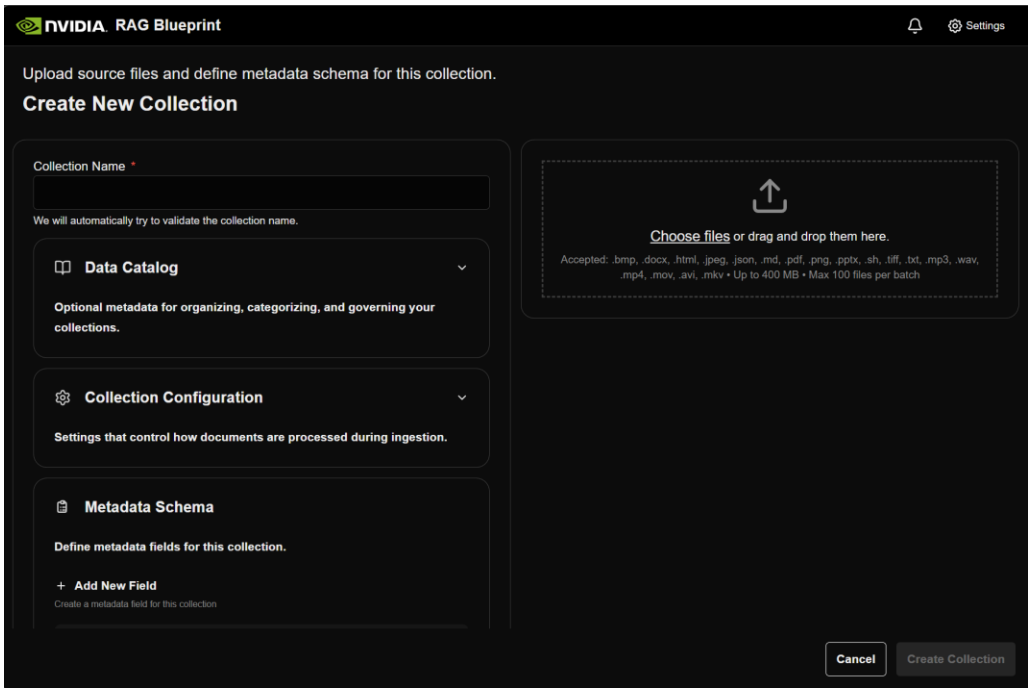


Figure 10: Create New Collection

We uploaded the documents by manually adding or dragging them. Optional metadata can be added to the

collection for organizing, categorizing, and governing purposes.

After clicking “Create Collection” the documents will then be ingested.

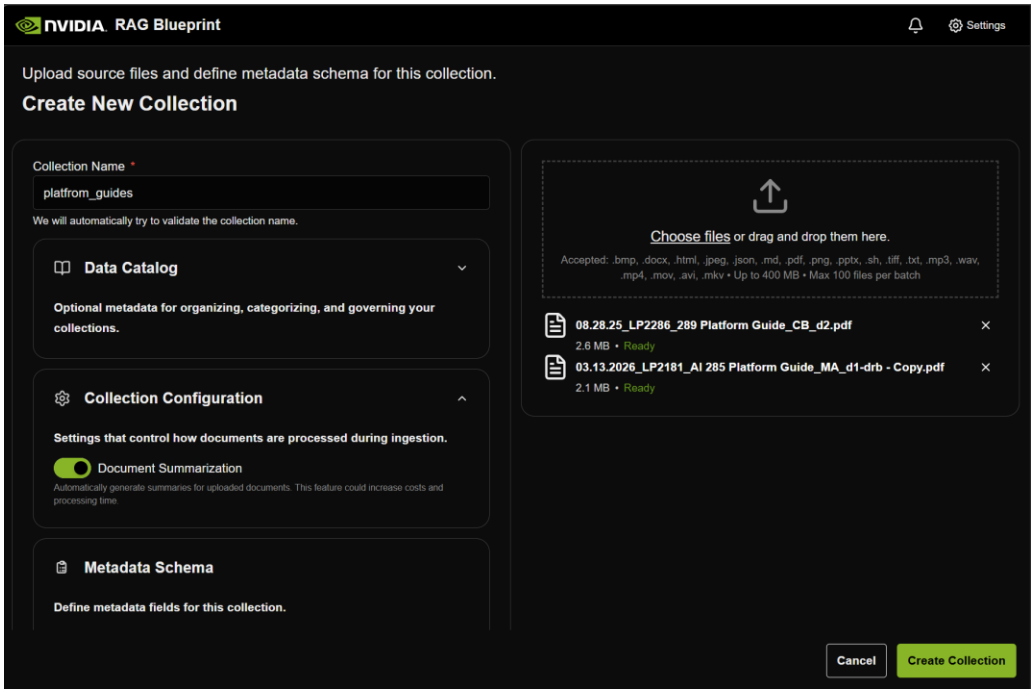
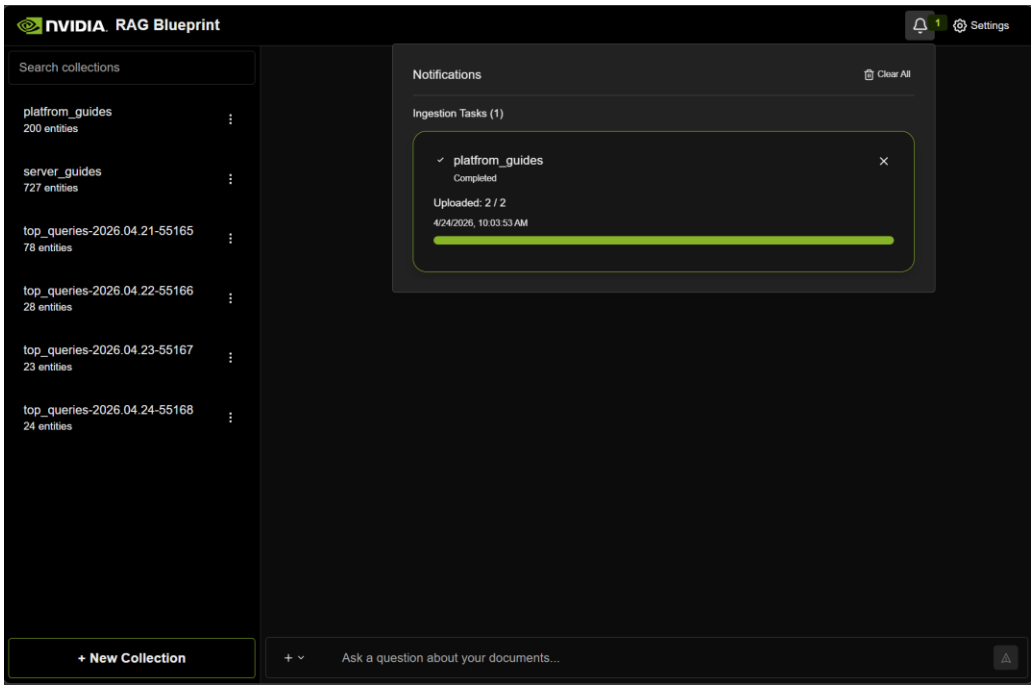


Figure 11: Uploading Documents to a Collection

When the ingestion process is complete, we will get a notification.



4.2.3 Document Retrieval and Query

Without any attached context, the model will return a standard missing information response.

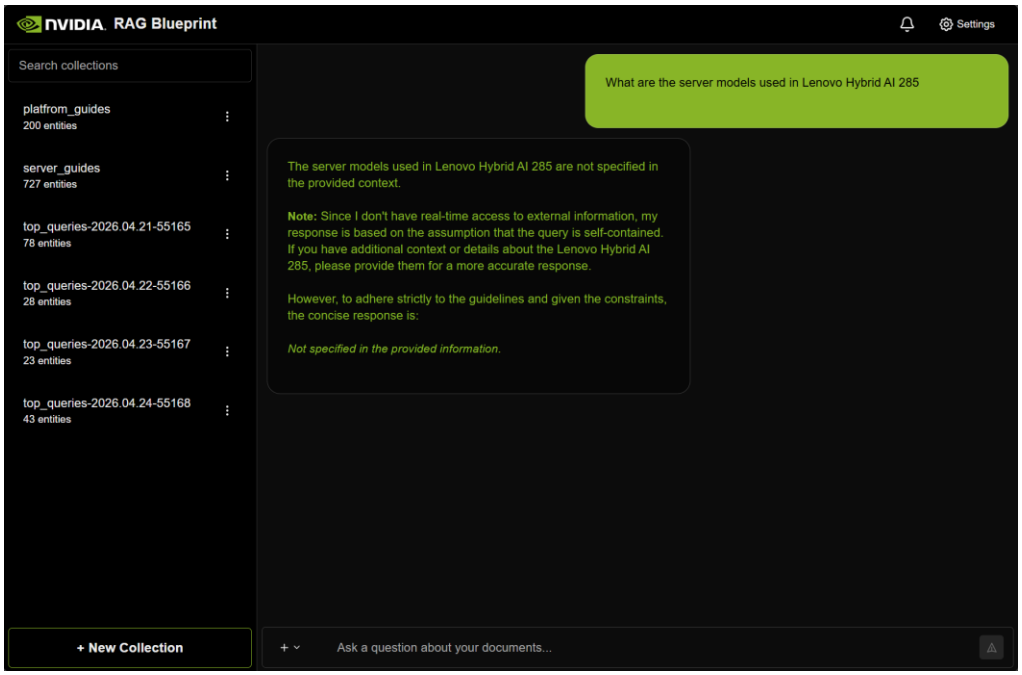


Figure 12: Missing context response

Use the corresponding collection during query by attaching them to the prompt. After attaching the right collection, the model can retrieve the information and provide a correct response.

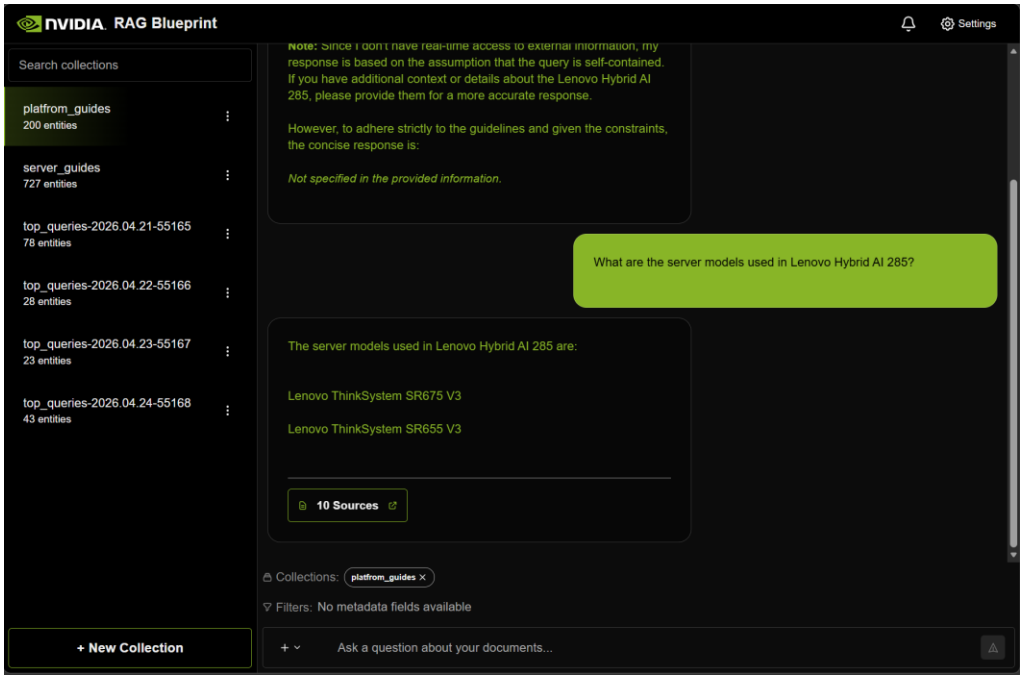


Figure 13: Successful retrieval and response

With the successful response, the RAG pipeline functionality is validated.

5 Performance Validation

Comprehensive benchmarking was conducted to validate the end-to-end performance, retrieval quality, and system throughput of the deployment. The evaluation sequence covers component health, Triton standalone performance, retrieval recall, end-to-end RAG quality, and system load testing.

5.1 RAG Quality Evaluation

Quality benchmarking was performed using the industry-standard **RAGAS** notebooks included in the NVIDIA RAG Blueprint, utilizing the **FinanceBench dataset** (150 questions across 84 PDFs, generating 40,522 OpenSearch chunks). The LLM judge used to evaluate answers was mistralai/Mixtral-8x22B-Instruct-v0.1 via the NVIDIA API Catalog.

- **Answer Accuracy:** Scored **0.6067**, which is within 0.9% of the official NVIDIA reference baseline of 0.612, demonstrating parity with Milvus-backed homogenous multi-GPU setups.
- **Context Relevance:** Achieved an excellent score of **0.9350**, indicating high relevance of the retrieved context to the query.
- **Response Groundedness:** Scored **0.8633**, confirming the LLM's responses were strongly grounded in the retrieved documents without hallucination.
- **Search Latency:** The average retrieval-only latency across the queries was **945 ms**, yielding an average of 6.6 results per query.

5.2 RAG Throughput and Load Testing

Throughput and latency degradation under load were tested using **Locust** against the /v1/generate (full RAG SSE stream), /v1/search (retrieval only), and /v1/health endpoints.

- **Optimal Operating Range:** The system can comfortably sustain **20 to 30 concurrent users**.
- **20 Concurrent Users:** Achieved **5.12 requests/second** with a **0% failure rate**. The average end-to-end generation latency was ~2.8 seconds, with a 95th percentile (p95) latency of 7.2 seconds.
- **50 Concurrent Users:** The system scaled to handle **10.06 requests/second**, though generation latency increased to an average of ~4.1 seconds (p95 of 9.5 seconds) with a minor 0.23% failure rate due to connection saturation.

5.3 Triton Inference Server Benchmarking

The standalone Triton Inference Server, sharing a 2g.48GB MIG slice via CUDA time-slicing, was validated using NVIDIA's perf_analyzer.

- Using the DenseNet-121 ONNX computer vision model, the server achieved a peak throughput of **~245 inferences per second** (via HTTP) and **~238 inferences per second** (via gRPC) at a concurrency level of 2-3.
- Compute time remained highly stable at approximately **4ms**, with a **0%** error rate, proving that the time-sliced MIG partition successfully isolates memory while sharing compute cycles effectively without bottlenecking the inference workload.

6 Appendix A: Method of Procedure (MoP)

6.1 Installation

Start with **setup_infra.sh** script

```
@tag-1043: ~/lenovo_nvidia_rag_blueprint
@tag-1043:~/lenovo_nvidia_rag_blueprint$ sudo ./setup_infra.sh
================================================================================
  Lenovo Infra Setup
================================================================================
  All output is written to: /var/log/lenovo-infra-setup.log
  Monitor progress:         tail -f /var/log/lenovo-infra-setup.log

  Steps:
    1. System preparation & NVIDIA drivers (reboot required)
    2. OpenSearch
    3. Canonical Kubernetes + Helm
    4. NVIDIA GPU Operator + MIG configuration
    5. Load balancer
    6. Namespaces, OpenSearch bridge & secrets

  This will take 15-30 minutes. Do not interrupt.
================================================================================

  Step 1 - System Preparation & NVIDIA Drivers

================================================================================
  Step 1 complete: NVIDIA drivers installed.

  Please reboot the system to load the kernel modules, then
  log back in and re-run this script to continue:

    sudo reboot
    # (after reboot)
    sudo bash /home/lqueirolo/lenovo_nvidia_rag_blueprint/setup_infra.sh

Full log: /var/log/lenovo-infra-setup.log
```

Reboot System

```
@tag-1043: ~/lenovo_nvidia_rag_blueprint
@tag-1043:~/lenovo_nvidia_rag_blueprint$ sudo reboot
Broadcast message from root@tag-1043 on pts/0 (Tue 2026-04-21 20:05:44 UTC):

The system will reboot now!
```

and continue with running setup_infra.sh script after system reboot to complete infra setup.

```
@tag-1043: ~/lenovo_nvidia_rag_blueprint
@tag-1043:~/lenovo_nvidia_rag_blueprint$ sudo ./setup_infra.sh

================================================================================
  Lenovo Infra Setup
================================================================================
  All output is written to: /var/log/lenovo-infra-setup.log
  Monitor progress:         tail -f /var/log/lenovo-infra-setup.log

  Steps:
    1. System preparation & NVIDIA drivers (reboot required)
    2. OpenSearch
    3. Canonical Kubernetes + Helm
    4. NVIDIA GPU Operator + MIG configuration
    5. Load balancer
    6. Namespaces, OpenSearch bridge & secrets

  This will take 15-30 minutes. Do not interrupt.
================================================================================



  Step 1 - System Preparation & NVIDIA Drivers


  Step 2 - OpenSearch Snap Setup

Run configure hook of "opensearch" snap
/
Run configure hook of "opensearch" snap
-
[0m[?25h[K
  Step 3 - Canonical Kubernetes Setup


  Step 4 - NVIDIA GPU Operator (Helm) + MIG Configuration


  Step 5 - Load Balancer Configuration


  Step 6 - Namespace Setup, OpenSearch Bridge & Secrets


  Step INFRASTRUCTURE SETUP COMPLETE


================================================================================
  INFRASTRUCTURE READY
================================================================================
```

```
  Server:         Lenovo ThinkSystem SR650a V4
  OS:             Ubuntu 24.04 LTS (Ubuntu Pro)
  GPUs:           2x NVIDIA RTX PRO 6000 Blackwell (96GB each)

  GPU Layout:
    GPU 0:  Full 96GB, non-MIG, dedicated (LLM only)
    GPU 1:  96GB, MIG 2x 1g.24gb + 1x 2g.48gb
            1g.24gb time-sliced 2x each (Embed+Rerank, GraphicElements+TableStructure)
            2g.48gb time-sliced 3x (OCR + Page Elements)

  K8s allocatable GPU resources:
    nvidia.com/gpu:               1  (GPU 0, dedicated)
    nvidia.com/mig-1g.24gb:  4  (2 physical x 2 replicas)
    nvidia.com/mig-2g.48gb:  3  (1 physical x 3 replicas)

  Infrastructure:
    Kubernetes:    Canonical K8s (snap)
    GPU Operator:  NVIDIA GPU Operator v26.3.0 (MIG mixed strategy)
    Load Balancer: Built-in (k8sd-managed, L2 mode)
    OpenSearch:    Snap (bare metal), TLS-secured, k-NN plugin enabled

  Endpoints:
    OpenSearch:    https://172.28.3.132:9200

  Next Step:
    sudo bash deploy_apps.sh

  Full log: /var/log/lenovo-infra-setup.log
```

Continue with deploying the applications with **deploy_apps.sh** script

```
================================================================================
@tag-1043: ~/lenovo_nvidia_rag_blueprint
@tag-1043:~/lenovo_nvidia_rag_blueprint$ sudo bash deploy_apps.sh

================================================================================
  Lenovo RAG Blueprint Deployment
================================================================================
  Chart:     /home/lqueirolo/lenovo_nvidia_rag_blueprint/rag_blueprint_lenovo/deploy/helm/nvidia-
blueprint-rag
  Release:   rag
  Namespace: rag
  Action:    install
  Log:       /var/log/lenovo-apps-deploy.log
```

```
==========================================================================
  Step 1 - Validate Prerequisites
  GPU resources detected:
  nvidia.com/gpu: 1
  nvidia.com/mig-1g.24gb: 4
  nvidia.com/mig-2g.48gb: 3

  Step 2 - NGC Credentials
  NGC API key is required to pull NIM container images.
  Get one at: https://org.ngc.nvidia.com/setup

  Enter NGC API key:

  Step 3 - Build Custom RAG Server Images (OpenSearch support)
  Using buildah buildah version 1.33.7 (image-spec 1.1.0-rc.5, runtime-spec 1.1.0)
  Logging into nvcr.io for base image access...

  Building rag-server image (this may take 10-20 minutes on first build)...
    Dockerfile:
/home/lqueirolo/lenovo_nvidia_rag_blueprint/rag_blueprint_lenovo/src/nvidia_rag/rag_server/Dockerfile
    Context:    /home/lqueirolo/lenovo_nvidia_rag_blueprint/rag_blueprint_lenovo
  Exporting and importing rag-server into containerd...
    lenovo/rag-server:2.5.0-opensearch — done.

  Building ingestor-server image (this may take 10-20 minutes on first build)...
    Dockerfile:
/home/lqueirolo/lenovo_nvidia_rag_blueprint/rag_blueprint_lenovo/src/nvidia_rag/ingestor_server/Docke
rfile
    Context:    /home/lqueirolo/lenovo_nvidia_rag_blueprint/rag_blueprint_lenovo
  Exporting and importing ingestor-server into containerd...
    lenovo/ingestor-server:2.5.0-opensearch — done.

  Custom images built and imported:
    lenovo/rag-server:2.5.0-opensearch
    lenovo/ingestor-server:2.5.0-opensearch

  Step 4 - Build Helm Dependencies
  Adding Helm repositories...
  Building chart dependencies (this may take a few minutes)...

  Step 5 - NIM Operator
  Installing NIM Operator...
  Waiting for NIM Operator CRDs to register...
  Waiting for NIM Operator pod to be ready...
```

```
  Step 6 - Deploy RAG Blueprint
  Installing rag...

  Step 7 - Deploy Triton Inference Server
  Creating NGC imagePullSecret in 'triton' namespace...
  Deploying Triton Inference Server (26.03)...
    Namespace: triton
    Image:     nvcr.io/nvidia/tritonserver:26.03-py3
    GPU:       nvidia.com/mig-2g.48gb: 1 (time-sliced 3x with OCR + Page Elements)
    Models:    triton-model-repo PVC (50Gi) mounted at /models
  Triton deployed. It will start once the container image is pulled.
  To add models, copy them into the triton-model-repo PVC:
    k8s kubectl cp ./my-model triton/triton-inference-server-<pod>:/models/my-model

  Step 8 - Post-Deploy Status

  Waiting for pods to start (this may take 10-30 minutes for NIM model downloads)...

  Current pod status (rag):
  -------------------
    NAME                                    READY   STATUS              RESTARTS   AGE   IP
NODE       NOMINATED NODE   READINESS GATES
    ingestor-server-b8796745f-hr2gn         1/1     Running             0          18s   10.1.0.62
tag-1043   <none>           <none>
    nemoretriever-graphic-elements-v1-pod   0/1     ContainerCreating   0          18s   <none>
tag-1043   <none>           <none>
    nemoretriever-ocr-v1-pod                0/1     ContainerCreating   0          18s   <none>
tag-1043   <none>           <none>
    nemoretriever-page-elements-v3-pod      0/1     ContainerCreating   0          18s   <none>
tag-1043   <none>           <none>
    nemoretriever-table-structure-v1-pod    0/1     ContainerCreating   0          18s   <none>
tag-1043   <none>           <none>
    nemotron-embedding-ms-cache-pod         0/1     ContainerCreating   0          18s   <none>
tag-1043   <none>           <none>
    nemotron-ranking-ms-cache-pod           0/1     ContainerCreating   0          18s   <none>
tag-1043   <none>           <none>
    nim-llm-cache-pod                       0/1     ContainerCreating   0          18s   <none>
tag-1043   <none>           <none>
    rag-frontend-86ccc4f9b9-5cq5t           1/1     Running             0          18s   10.1.0.231
tag-1043   <none>           <none>
    rag-nv-ingest-79754bdb59-p4kg5          0/1     ContainerCreating   0          18s   <none>
tag-1043   <none>           <none>
```

```
    rag-redis-master-0                      0/1     Running             0          18s   10.1.0.30
tag-1043   <none>           <none>
    rag-redis-replicas-0                    0/1     Running             0          18s   10.1.0.18
tag-1043   <none>           <none>
    rag-server-568ccf68c-4qzw4              1/1     Running             0          18s   10.1.0.154
tag-1043   <none>           <none>

  Current pod status (triton):
  --------------------
    NAME                                    READY   STATUS     RESTARTS   AGE   IP           NODE
NOMINATED NODE   READINESS GATES
    triton-inference-server-6f7d96c96-bgmmf   0/1     Init:0/1   0          13s   10.1.0.173   tag-
1043   <none>           <none>

  NOTE: 1 pod(s) are still Pending.
  This is normal during initial deployment — NIM images are large (10-40GB each).
  Monitor progress:
    k8s kubectl get pods -n rag -w
    k8s kubectl get pods -n triton -w
    k8s kubectl describe pod <pod-name> -n <namespace>

======================================================================
  RAG BLUEPRINT + TRITON DEPLOYED
======================================================================

  Release:    rag
  Namespace:  rag
  Chart:      /home/lqueirolo/lenovo_nvidia_rag_blueprint/rag_blueprint_lenovo/deploy/helm/nvidia-
blueprint-rag

  Access:
    The frontend is exposed via LoadBalancer. Get the external IP:
      k8s kubectl get svc rag-frontend -n rag
    Then open: http://<EXTERNAL-IP>:3000
  Internal services (cluster-only):
    RAG Server:       rag-server:8081
    Ingestor Server:  ingestor-server:8082
    NIM LLM:          nim-llm:8000
    OpenSearch:       opensearch:9200  (bridge to bare-metal snap)
    Triton HTTP:      triton-inference-server:8000  (namespace: triton, KServe V2 / REST)
    Triton gRPC:      triton-inference-server:8001  (namespace: triton)
    Triton Metrics:   triton-inference-server:8002  (namespace: triton)

  GPU Layout:
```

```
    GPU 0: NIM LLM (Llama-3.3-Nemotron-Super-49B-v1.5, NVFP4, TP1) — nvidia.com/gpu: 1
    GPU 1: MIG 2x 1g.24gb + 1x 2g.48gb
      1g.24gb (2x TS): Embed, Rerank, Graphic Elements, Table Structure
      2g.48gb (3x TS): OCR, Page Elements, Triton Inference Server

  Vector Database: OpenSearch (bare-metal snap, via K8s Service+Endpoints bridge)

  Custom images (built with buildah, imported to containerd k8s.io):
    lenovo/rag-server:2.5.0-opensearch
    lenovo/ingestor-server:2.5.0-opensearch

  Triton model repository:
    PVC: triton-model-repo (50Gi) mounted at /models in namespace 'triton'
    Copy models: k8s kubectl cp ./my-model triton/<triton-pod>:/models/my-model

  Monitor pods:
    k8s kubectl get pods -n rag -w

  Check NIM startup logs:
    k8s kubectl logs -f <nim-pod-name> -n rag

  Troubleshoot:
    k8s kubectl describe pod <pod-name> -n rag
    k8s kubectl get events -n rag --sort-by='.lastTimestamp'

  Full deploy log: /var/log/lenovo-apps-deploy.log

======================================================================
```

Wait until all pods are running.

k8s kubectl get pods -n rag -w

k8s kubectl get pods -n triton -w

```
root@tag-1043:/home/lqueirolo# root@tag-1043:/home/lqueirolo# k8s kubectl get pods -n rag -w
NAME                                                READY   STATUS    RESTARTS   AGE
ingestor-server-b8796745f-hr2gn                     1/1     Running   0          2d1h
nemoretriever-graphic-elements-v1-5f5fdfc45b-4hwqm  1/1     Running   0          2d1h
nemoretriever-ocr-v1-68d665474-h8x7k                1/1     Running   0          2d1h
nemoretriever-page-elements-v3-5b8957659d-8qd4h     1/1     Running   0          2d1h
nemoretriever-table-structure-v1-5668d78f68-hh2k8   1/1     Running   0          2d1h
nemotron-embedding-ms-5879b59d8b-92fgd              1/1     Running   0          2d1h
nemotron-ranking-ms-8948b4fd5-dl8g9                 1/1     Running   0          2d1h
nim-llm-86cffbc4b6-s4hp2                            1/1     Running   0          2d1h
rag-frontend-86ccc4f9b9-5cq5t                       1/1     Running   0          2d1h
```

```
rag-nv-ingest-79754bdb59-p4kg5                          1/1     Running   0          2d1h
rag-redis-master-0                                      1/1     Running   0          2d1h
rag-redis-replicas-0                                    1/1     Running   0          2d1h
rag-server-568ccf68c-4qzw4                              1/1     Running   0          2d1h

root@tag-1043:/home/lqueirolok8s kubectl get pods -n triton -w-w
NAME                                      READY   STATUS    RESTARTS   AGE
triton-inference-server-6f7d96c96-bgmmf   1/1     Running   0          2d1h
root@tag-1043:/home/lqueirolo#
```

6.2 Uninstall

Uninstallation of the deployment is done using **cleanup.sh** script.

```
@tag-1043:~/lenovo_nvidia_rag_blueprint$ sudo ./cleanup.sh
================================================================================
  Lenovo RAG Blueprint — Full Cleanup
================================================================================
  Log: /var/log/lenovo-cleanup.log

  WARNING: This will remove ALL components installed by setup_infra.sh
  and deploy_apps.sh, including Kubernetes, OpenSearch, NVIDIA drivers,
  all data, and all credentials.

  Are you sure you want to proceed? Type 'yes' to confirm: yes

========================================
  Phase 1: Stop port-forward processes
========================================
  [SKIP] No rag-frontend port-forward running.

========================================
  Phase 2: Remove Helm releases and Kubernetes resources
========================================
  Uninstalling Helm release 'rag'...
  [OK] Helm release 'rag' uninstalled.
  Uninstalling Helm release 'nim-operator'...
  [OK] Helm release 'nim-operator' uninstalled.
  Deleting Triton resources...
  [OK] Triton resources deleted.
  Uninstalling Helm release 'gpu-operator'...
  [OK] Helm release 'gpu-operator' uninstalled.
  Deleting namespace 'rag'...
  [WARN] Namespace 'rag' deletion timed out.
```

```
  [OK] Namespace 'rag' deleted.
  Deleting namespace 'triton'...
  [OK] Namespace 'triton' deleted.
  Deleting namespace 'nim-operator'...
  [OK] Namespace 'nim-operator' deleted.
  Deleting namespace 'gpu-operator'...
  [OK] Namespace 'gpu-operator' deleted.
  Deleting namespace 'inference'...
  [OK] Namespace 'inference' deleted.
  [OK] Removed image 'docker.io/lenovo/rag-server:2.5.0-opensearch' from containerd.
  [OK] Helm releases and K8s resources cleaned up.

=======================================
  Phase 3: Remove Helm repos and binary
=======================================
  [OK] Helm repos removed.
  [OK] Helm binary removed.

=======================================
  Phase 4: Disable MIG mode
=======================================
  Disabling MIG on GPU 1...
  [WARN] MIG disable failed (may require reboot).
  [OK] MIG disabled on GPU 1 (reboot may be required to take effect).

=======================================
  Phase 5: Remove snap packages
=======================================

  Remove Canonical K8s snap? This destroys the entire cluster and all PVCs. [y/N] y
  Removing Canonical K8s snap...
  [OK] Canonical K8s snap removed.

  Remove OpenSearch snap? This destroys all indexed data. [y/N] y
  Removing OpenSearch snap...
  [OK] OpenSearch snap removed.
  [OK] Removed vm.max_map_count=262144 from /etc/sysctl.conf

=======================================
  Phase 6: Remove NVIDIA drivers and CUDA packages
=======================================

  Remove NVIDIA drivers, CUDA toolkit, and related packages? (Answer 'n' to keep the drivers
installed) [y/N] y
```

```
  Found 72 NVIDIA/CUDA package(s) to remove.
  [OK] NVIDIA driver and CUDA packages removed (72 packages).
  [OK] Removed /etc/modprobe.d/blacklist-nouveau.conf
  [OK] Removed /etc/modprobe.d/nvidia-uvm.conf
  [OK] Initramfs rebuilt.

========================================
  Phase 7: Remove credentials, logs, and temporary files
========================================
  [OK] Removed /root/.credentials/lenovo-rag
  [OK] Credentials, logs, and temporary files removed.

================================================================================
  CLEANUP COMPLETE
================================================================================

  Removed:
    - Helm releases (rag, nim-operator, gpu-operator)
    - K8s namespaces (rag, triton, nim-operator, gpu-operator, inference)
    - Custom container images
    - Snap packages (k8s, opensearch) — if confirmed
    - NVIDIA drivers and CUDA packages — if confirmed (may have been kept)
    - Credentials, logs, and temp files

  NVIDIA kernel module configs (nouveau blacklist, nvidia-uvm) were removed.
  A reboot is recommended to fully unload NVIDIA kernel modules.
  OpenSearch sysctl tuning (vm.max_map_count) was removed.

  Cleanup log: /var/log/lenovo-cleanup.log

================================================================================
```

7 Appendix B: Bill of Material (BoM)

For more information on the Bill of Material (BoM), please refer to Lenovo Hybrid AI 221 Platform Guide.

7.1 Hybrid AI 221 Canonical Solution ID in DCSC

The Lenovo Hybrid AI 221 Solution ID (SID) is now officially available for configuration and ordering within the Data Center Solution Configurator (DCSC). By using a pre-defined Solution ID (SID), we ensure technical accuracy and consistency across our global sales' technical engagements.

For Hybrid AI 221 Canonical configuration, use **CRN Number: SID0001042 Hybrid AI 221 Single Node 650i V4 RTX Pro 6000** under Deployment Ready Solutions – AI – Hybrid AI 221. Customize the SID and add the SW license for Ubuntu PRO.

Resources

Resources	Links
NVIDIA RAG Blueprint Documentation	https://docs.nvidia.com/rag/2.5.0/index.html
NVIDIA RAG Blueprint Github	https://build.nvidia.com/nvidia/build-an-enterprise-rag-pipeline
Lenovo Hybrid AI 221 Platform Guide	https://lenovopress.lenovo.com/lp2313-lenovo-hybrid-ai-221-platform-guide
Lenovo Hybrid AI Software Platform	https://lenovopress.lenovo.com/lp2311-lenovo-hybrid-ai-software-platform

Document history

Version 1.0	May 2026	Lenovo Hybrid AI 221 Canonical Single-node-RAG v1.0

Trademarks and special notices

References in this document to Lenovo products or services do not imply that Lenovo intends to make them available in every country.

Lenovo, the Lenovo logo, ThinkCentre, ThinkVision, ThinkVantage, ThinkPlus and Rescue and Recovery are trademarks of Lenovo.

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Intel Inside (logos), MMX, and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Information is provided "AS IS" without warranty of any kind.

All customer examples described are presented as illustrations of how those customers have used Lenovo products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer.

Information concerning non-Lenovo products was obtained from a supplier of these products, published announcement material, or other publicly available sources and does not constitute an endorsement of such products by Lenovo. Sources for non-Lenovo list prices and performance numbers are taken from publicly available information, including vendor announcements and vendor worldwide homepages. Lenovo has not tested these products and cannot confirm the accuracy of performance, capability, or any other claims related to non-Lenovo products. Questions on the capability of non-Lenovo products should be addressed to the supplier of those products.

All statements regarding Lenovo future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. Contact your local Lenovo office or Lenovo authorized reseller for the full text of the specific Statement of Direction.

Some information addresses anticipated future capabilities. Such information is not intended as a definitive statement of a commitment to specific levels of performance, function or delivery schedules with respect to any future products. Such commitments are only made in Lenovo product announcements. The information is presented here to communicate Lenovo’s current investment and development activities as a good faith effort to help with our customers' future planning.

Performance is based on measurements and projections using standard Lenovo benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

Photographs shown are of engineering prototypes. Changes may be incorporated in production models.

Any references in this information to non-Lenovo websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this Lenovo product and use of those websites is at your own risk.