AI-Q NVIDIA Research Agent Blueprint for Enterprise RA

Deployment, Scale and Sizing Guide

Version 01

Table of Contents

Abstract

The current landscape of Enterprise AI is rapidly evolving, shifting from just having a dialogue with a large language model (LLM) to actually having sophisticated, goal-oriented *Agentic Workflows*. This shift is driven by the need to not just generate responses from an LLM but to also to have an AI do tasks based on the information retrieved. Most Enterprises have a vast amount of data in PDFs or other documents that can be used to ask complex questions, Retrieval Augmented Generation (RAG) can help answer questions within Enterprise data, however if there are more complex questions that require private as well as Public Internet data, we need an agent that can fetch data from both sources, reflect on it and provide detailed reports. In this paper, we look at the NVIDIA AI-Q Research Agent blueprint, an agentic system that can generate detailed reports based on both internal and external data. We walk through how to deploy, how to scale and provide sizing guidance.

Introduction

The current enterprise landscape is characterized by a rapidly escalating volume of proprietary, internal data—often siloed within numerous systems like document repositories, knowledge bases, and collaborative platforms. Simultaneously, the demand for sophisticated, evidence-based decision-making is driving enterprises toward deploying advanced AI systems. However, traditional Large Language Model (LLM) deployments face significant hurdles:

- **Data Siloing & Limited RAG:** Internal knowledge is untapped because LLMs use public data. Basic RAG lacks multi-step reasoning across diverse, proprietary sources.
- **Lack of Agency & Complex Reasoning:** LLMs struggle with goal-oriented tasks needing planning and iterative refinement. Users need a " Research Agent" for synthesizing reports from internal and external data.
- **Deployment Complexity & Scalability:** Deploying powerful agentic AI requires significant GPUs and complex orchestration. Lack of sizing guidance leads to high costs and bottlenecks.

NVIDIA Blueprints simplify the deployment and management of complex AI systems offering pre-validated architecture that automates dependency management between

various components, and uses Helm for easy deployment and scaling. The **NVIDIA AI-Q Research Agent** blueprint directly addresses these challenges by providing a robust, scalable, and agentic framework. AI-Q uses RAG for retrieval, can consume multiple data sources, reason between different datasources, search the web for any additional research and provide a detailed report on a given subject. AI-Q provides the open blueprint and reference architecture for building next-generation agents, while NeMo Agent Toolkit is the underlying control layer, that integrates with other frameworks like LangChain, LlamaIndex, etc., and connects, profiles, and optimizes AI agents across frameworks and infrastructure..

Scope

This paper covers how to scale, size and optimize AI-Q Research Agent and covers the following components on Enterprise Reference Architecture.

Out of scope: AI-Q allows a human in the loop feature, this feature was not used when conducting performance tests for AI-Q.

Table1. Scope of NVIDIA Blueprints, NVIDIA NIM versions along with GPUs tested

NVIDIA Components	NVIDIA GPUs
AI-Q Blueprint v.1.2.0 w/ Meta Llama 70B NIM v latest	RTX PRO 6000 BSE
RAG Blueprint v.2.3.0 With Nemotron Super 49B 1.5 NIM v 1.14.0	RTX PRO 6000 BSE
NeMo Agent Toolkit v.1.2.0	RTX PRO 6000 BSE

Target Audience

This guide is meant to help NVIDIA partners architecting agentic solutions for deep research based on existing enterprise data. This guide helps determine Infrastructure and sizing requirements for cluster-level deployments. The guide can be used for both new and existing deployments to determine the capacity and scale needed based on Deep Research workload. It can also be used by the following Personas as they architect their Enterprise solutions.

Enterprise Architects: Enterprise Architects tasked with designing and defining servers, GPU's and Networking gear to determine what Infrastructure resources will be needed to support Deep Research agents.

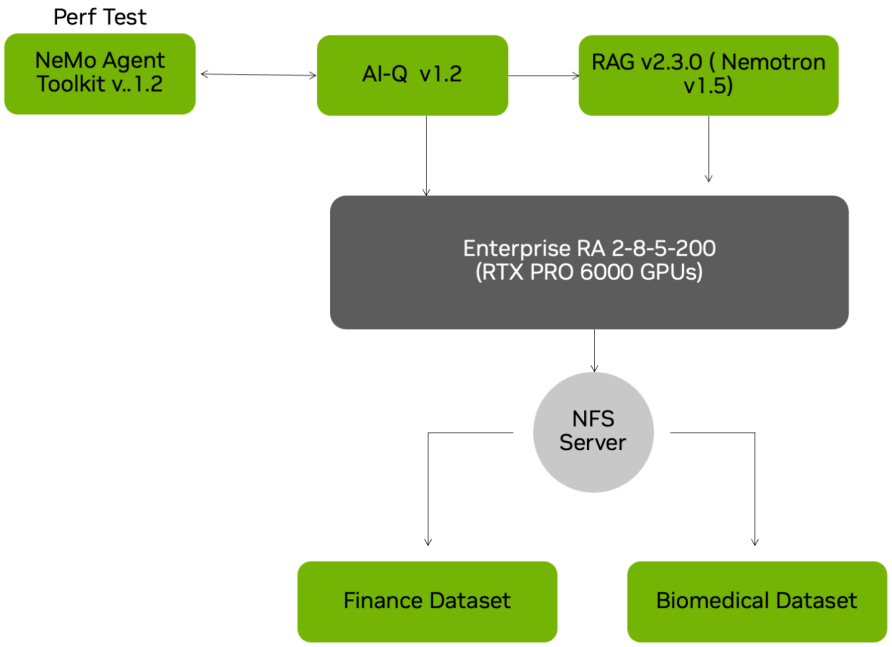
MLOps Engineer: MLOps Engineers can use this to define Infrastructure requirements as they talk to Infrastructure/Cloud teams to carve out resources to run Agentic workflows for Deep Research.

Platform Engineer: Platform Engineers can use this guide to determine how to design their Container Environment around Kubernetes, and also what kind of resources will be needed by the cluster to support Deep research agents.

System Overview and Architecture

This paper was validated on E**nterprise Reference Architecture** in the **2-8-5-200** configuration using **RTX PRO 6000 Blackwell Server Edition GPUs** , with **Cluster Software Reference stack version 25.09** and AI-Q, and RAG deployed on top of it.

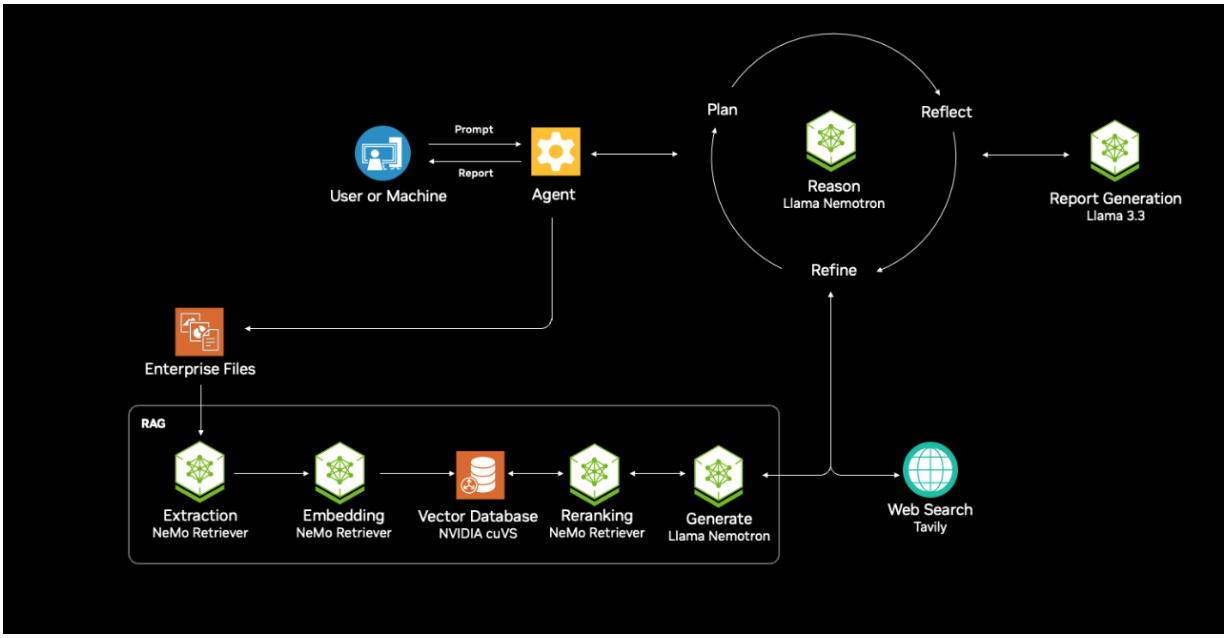
Figure 1. System Overview



AI-Q NVIDIA Research Agent

AI-Q is an advanced agentic framework designed for deep, comprehensive research. It functions by employing a **Retrieval-Augmented Generation (RAG)** system to extract and contextualize information from existing enterprise documents. It leverages **Tavily** internet search agent for external information gathering. This combination allows AI-Q to correlate and reason about topics from disparate internal files, integrate external data, and ultimately synthesize all fetched information into a complete report.

Figure 2: AI-Q Blueprint Architecture



Here are the key components of AI-Q

Data Ingestion and Preparation Pipeline

This component handles enterprise data onboarding and indexing. **NeMo Retriever extraction** is used to chunk and embed multi-modal data from multiple sources, the embeddings are then stored in a Vector Database like **Milvus**. This pipeline is part of the **NVIDIA RAG Blueprint**. The tests conducted for performance used two different datasets , these are uploaded and part of the blueprint on Github.

The datasets used were:

- **Biomedical_Dataset**: Scientific journals on the Cystic Fibrosis CFTR gene from 2021-2024
- **Financial_Dataset**: Financial reports from Apple, Facebook, Google, Meta from 2020-2024

RAG (Retrieval-Augmented Generation) Framework

The RAG layer provides grounded context retrieval for LLM inference, this executes semantic and hybrid search across our datasets. Applies metadata filtering, reranking, returns high-relevance context to minimize hallucination. The RAG blueprint used here uses NVIDIA **Nemotron 49B Super v1.5** as the NIM LLM to perform the reasoning for a given research. We call this the **Reasoning NIM** for reference hench forth in the document configs

AI-Q Instruct LLM and Toolchain

This is part of the blueprint responsible for coming up with the initial plan based on the users instruction, the Instruct LLM used in the blueprint is **Meta Llama 70B,** it judges the response from the RAG and Reasoning LLM , determines if additional internet search is needed, if needed, it will kick off Tavily to perform a search. The Instruct LLM, then also synthesizes the final response by combining retrieved context with the user's instruction, while enforcing format, tone, or policy constraints

Performance and Observability

AI-Q helm charts can optionally also deploy Phoenix. Phoenix is an open source AI observability platform designed for experimentation, evaluation, and troubleshooting. Phoenix can help evaluate where the toolkit spends time, show traces of entire agentic workflows, records time series based metrics over open telemetry like tokens generated, latency etc.

Phoenix is a great tool to visualize how the agentic workflow is behaving, where are the bottlenecks if any.

Enterprise Reference Architecture Overview (RA)

This guide is part of NVIDIA Enterprise Reference Architecture, which covers certified hardware, software stack, and sizing recommendations to design, build, and scale an end-to-end accelerated computing cluster deployment with balanced CPU to GPU to NIC patterns. The Enterprise Reference architecture provides guided and detailed hardware

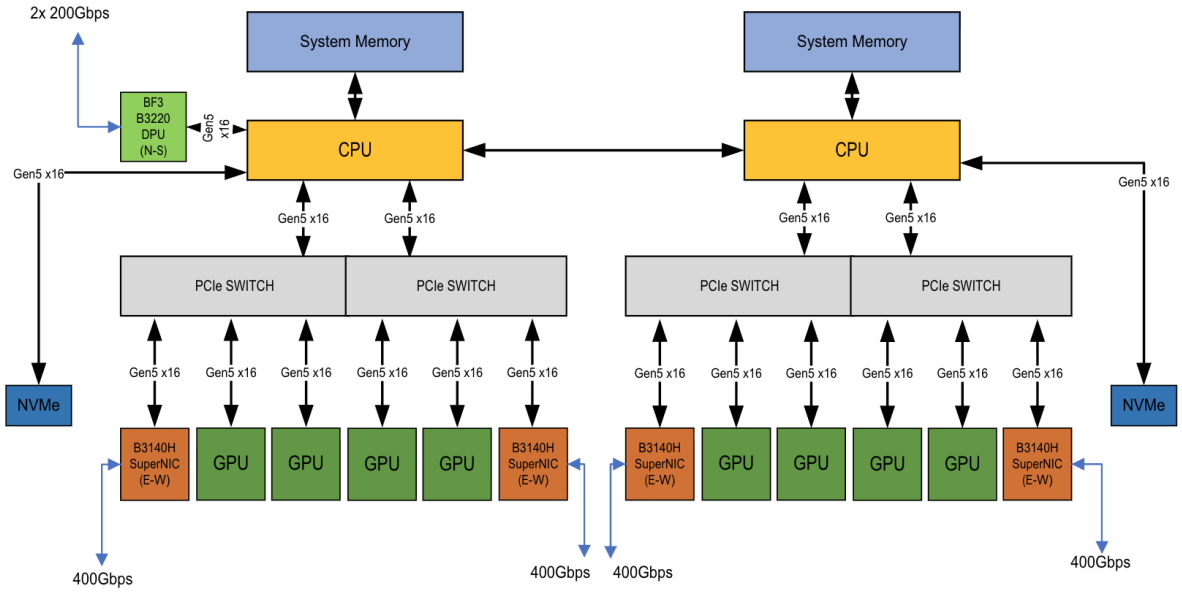
and software architecture recommended by NVIDIA for optimal server, cluster, and network configuration needed to build and scale AI factories.

NVIDIA Enterprise Reference Architecture includes hardware design recommendations, generic software stack configurations, and scalability.

Hardware Enterprise Reference Architecture

For this version of the document, we used NVIDIA's 2-8-5-200 Enterprise RA reference configuration for the overall stack. The PCIe-Optimized 2-8-5-200 (CPU-GPU-NIC-Bandwidth) reference configuration is for NVIDIA-Certified compute nodes using PCIe GPUs, allowing you to deploy up to 8 GPUs with up to 5 NICs balanced with 2 CPUs. This pattern can scale from 4 to up to 32 nodes in a cluster. The Enterprise RA design recommends using Spectrum-X Ethernet Networking Platform - Combining Spectrum-4 Ethernet switches and NVIDIA Bluefield-3 SuperNICs for optimized networking.

Figure 3. System architecture of Enterprise RA 2-8-5-200 reference configuration



Note: For the detailed hardware design, refer to NVIDIA 2-8-5-200 with RTX PRO 6000 BSE NVIDIA Spectrum Platforms Enterprise Reference Architecture: NVOnline :1125114

Table 2. Specification of Individual Components in the server with NVIDIA RTX PRO 6000 Blackwell Server Edition GPUs

Component	Specification
CPUs	AMD EPYC 9555 64-Core Processor 64 cores 3200 MT/s
GPUs	8 x NVIDIA RTX PRO 6000 BSE
Networking – E/W	4 x NVIDIA BlueField-3, B3140H
Networking – N/S	1 x NVIDIA BlueField-3, B3220
Host Memory	32 x 64 GB DRAM (2048)
Host Boot Drive	2 x 896 GB (~1.8 TB)
Host Storage	2 x 3.84TB Storage (~7.6 TB)

The systems are connected with NVIDIA SN5600 switches. For this particular guide, we are using NFS Storage that has been deployed on the BCM head node to provide Persistent Volumes for uploading embedded Enterprise documents. Enterprise customers can use their storage systems to provide storage volumes needed by Kubernetes and NIM inference microservices.

For this test we used a pool of 9 servers with 8 NVIDIA RTX PRO 6000 BSE GPUs per node. The complete cluster has 72 GPUs in total**.**

Software Stack

The software stack used for this environment leverages the bare-metal servers with Kubernetes as the cluster orchestration tool. BCM is part of the NVIDIA AI Enterprise software suite, and it provides all the tools you need to deploy and manage an AI datacenter. It also helps in deploying a Kubernetes cluster on top of the bare-metal servers to create a pool of GPU resources. BCM is then used to deploy all the operators like the GPU Operator, Network Operator, NIM Operator, etc. to run the GPUs and Network cards effectively. We then deploy a NIM service, picking the model for which the Inference service needs to be tested and scaled. We installed Run:ai version 2.2 using the SaaS format.

Figure 5. Overview of the overall deployment stack

This test environment uses NVIDIA AI Enterprise software to install and configure the necessary software and tools required to efficiently deploy and operate an AI factory. Once servers are racked and networked, NVIDIA BCM is used to image individual servers, deploying Ubuntu 24.04 as the operating system, installing NVIDIA GPU and network drivers, and setting up Kubernetes clusters.

Beyond these core capabilities, the deployment also aligns the software dependencies for various components required for Kubernetes cluster operations, such as a **Container Network Interface (CNI)** for managing container networks, **NGINX Ingress Controller** for handling cluster ingress traffic, and **MetalLB** for load balancing services, Prometheus and Grafana to visualize overall stack metrics and also provides firmware to configure and optimize OS like Cumulus Linux for the NVIDIA Spectrum switches, firmware for the Bluefield Super NICs etc.

	Note: For detailed Software reference design, refer to NVIDIA Software Reference Stack and automation for Enterprise RA - vanilla Kubernetes: NVOnline : 1141332

System Configuration

This section provides a step by step instruction on how the blueprints are deployed once the Cluster Software Reference stack has been deployed and configured.

Assumptions

- The environment has egress to the internet enabled

Pre-requisites for installing RAG and AI-Q

- A **Kubernetes** Cluster with a supported version installed with NVIDIA GPU Operator and NVIDIA Network Operator installed, which should already be in place if the NVIDIA Software Enterprise RA is followed.
- Active Subscription to NVAIE and Access to the **NGC** Enterprise Catalog. Please generate and download your nvcr.io access token in NGC.
- Account in **Tavily** and an Access token to Tavily API, this will be needed for AI-Q to run internet search agent
- Install **Helm** in the Kubernetes Cluster and download the helm CLI. This should have been installed if the Software Enterprise RA is followed.
- Access to Kubernetes Clusters config file and `kubectl` CLI installed, this file is in the Enterprise RA cluster BCMe head node under /<user>/.kube/config
- A default Storage Class is defined to create Persistent Volume Claims by Kubernetes, this can be an NFS or Block-based storage class, the storage class name is default.
- Ability to create two namespaces in the Kubernetes cluster. rag, aiq

Deploy and Configure RAG Blueprint

NVIDIA RAG Blueprint can be deployed in multiple ways, for the purpose of this document and respect to the Enterprise Software Reference Architecture we are using Helm as the deployment method. For detailed information on Helm and how to deploy RAG using Helm, refer to the NVIDIA Github Repository here.

From the system that has access to the Kubernetes API and has Helm deployed, run the following on the command prompt

Export the NGC API KEY

```
export NGC_API_KEY="nvapi-<redacted>"
```

Create a directory for RAG, CD to it and fetch the RAG repo

```
mkdir rag
cd rag
Git clone https://github.com/NVIDIA-AI-Blueprints/rag.git
```

Add the appropriate Helm Repos to the cluster

```
helm repo add nvidia-nim https://helm.ngc.nvidia.com/nim/nvidia/ --username='$oauthtoken' --
password=$NGC_API_KEY
helm repo add nim https://helm.ngc.nvidia.com/nim/ --username='$oauthtoken' --
password=$NGC_API_KEY
helm repo add nemo-microservices https://helm.ngc.nvidia.com/nvidia/nemo-microservices --
username='$oauthtoken' --password=$NGC_API_KEY
helm repo add baidu-nim https://helm.ngc.nvidia.com/nim/baidu --username='$oauthtoken' --
password=$NGC_API_KEY
helm repo add bitnami https://charts.bitnami.com/bitnami
helm repo add otel https://open-telemetry.github.io/opentelemetry-helm-charts
helm repo add zipkin https://zipkin.io/zipkin-helm
helm repo add prometheus https://prometheus-community.github.io/helm-charts
```

Change the directory

```
cd rag/deploy/helm
```

create a namespace for RAG

```
kubectl create namespace rag
```

Update the default values from RAG Helm chart to work with RTX PRO 6000 BSE GPUs, copy and paste the default values file to a new file name in the rag/deploy/helm folder called values-2gpu-rtxpro6000.yaml. The detailed YAML file for this config is provided in Appendix A, RAG Config, below are key things to update in the file

Ensure under the NIM section, the model is updated with Nemotron 49B v1.5 with the right NIM profile for RTX PRO with fp8 precision

```
nim-llm:
  enabled: true
  image:
    repository: nvcr.io/nim/nvidia/llama-3.3-nemotron-super-49b-v1.5
    tag: "1.14.1"
  resources:
    limits:
      nvidia.com/gpu: 2
      memory: 128Gi  # Increased from 96Gi
    requests:
      nvidia.com/gpu: 2
      memory: 128Gi  # Increased from 96Gi
  model:
    name: "nvidia/llama-3.3-nemotron-super-49b-v1.5"
  env:
    - name: NIM_MODEL_NAME
      value: "nvidia/llama-3.3-nemotron-super-49b-v1.5"
    - name: NIM_MODEL_PROFILE
      value: "610f006b15f3adbdb072da0b4155d8a772332cf1768fb7389ef92a83c31c26dc"
```

Also, if needed you can provide a specific node in the cluster to deploy rest of the RAG components

Install RAG using Helm

```
helm upgrade --install rag -n rag https://helm.ngc.nvidia.com/nvidia/blueprint/charts/nvidia-blueprint-rag-v2.2.0.tgz -f values-2gpu-rtxpro6000.yaml \
--username '$oauthtoken' \
--password "${NGC_API_KEY}" \
--set imagePullSecret.
password=$NGC_API_KEY \
--set ngcApiSecret.password=$NGC_API_KEY
```

Update Helm Dependencies

```
helm dependency update nvidia-blueprint-rag
```

At this point RAG has been deployed and configured, the pods for RAG nim-llm-0 will be be deployed, these pods will take a few minutes to go into “running” state along with other components and also the RAG server and nim-llm will have services created in Kubernetes with cluster IP. We need to make sure that the RAG server and the NIM LLM have IP addresses that can be reached by the benchmarking RAG tool. For this exercise we used External Loadbalancing IPs for both these services , you can do the same

Edit the NIM-LLM service, RAG server to use Load Balancing IP

```
kubectl patch service nim-llm -n rag \
 -p '{"spec":{"type":"LoadBalancer","ports":[{"port":8000,"targetPort":8000}]}}'
```

```
kubectl patch service rag-server -n rag \
 -p '{"spec":{"type":"LoadBalancer","ports":[{"port":8081,"targetPort":8081}]}}'
```

At this point the setup should have the RAG pipeline deployed with all its components in Milvus in a standalone fashion.

Deploy and Configure AI-Q Blueprint

We are deploying the NVIDIA AI-Q blueprint from Github Repo here.

Clone the Git Repo into a folder called aiq

```
git clone https://github.com/NVIDIA-AI-Blueprints/aiq-research-assistant
```

Set environment variables, on the system from which you are installing AI-Q

```
export NGC_API_KEY=”nvapi-xxx” # your API key
export TAVILY_API_KEY=”yyy” # your Tavily API key, optional for web search
```

Create a namespace:

```
kubectl create namespace aira
cd aiq-research-assistant/
```

Add the AI-Q helm chart

```
helm repo add nvidia-nim https://helm.ngc.nvidia.com/nim/nvidia/ --username='$oauthtoken' --
password=$NGC_API_KEY
helm repo add nim https://helm.ngc.nvidia.com/nim/ --username='$oauthtoken' --
password=$NGC_API_KEY
```

```
helm repo add nvidia-nim https://helm.ngc.nvidia.com/nim \
 --username='$oauthtoken' \
 --password=$NGC_API_KEY
```

We need to change the default Helm Values file for our system, copy the values.yaml file in the folder under deploy/helm/aiq-aira to a file called values-rtx-pro-6k.yaml in the deploy/helm folder. We will be editing the file to make some key changes.

Update the nim-llm section to point to the version of the latest version of NIM with the correct profile for RTX PRO 6000 BSE GPUs

```
nim-llm:
 enabled: true
 service:
   name: "nim-llm"
 image:
    repository: nvcr.io/nim/meta/llama-3.3-70b-instruct
    pullPolicy: IfNotPresent
    tag: "1.12.0"
 resources:
  limits:
    nvidia.com/gpu: 1
  requests:
    nvidia.com/gpu: 1
 model:
  name: "meta/llama-3.3-70b-instruct"
 env:
  - name: NIM_MODEL_NAME
    value: "meta/llama-3.3-70b-instruct"
  - name: NIM_MODEL_PROFILE
    value: "257a3035dbddb2a2cd48f5763ee64f972af4ba0dd5ef92ade1dbb478f6cf5dd3"
```

Detailed config YAML for AI-Q is provided in Appendix B of this document

Deploy the AI-Q Helm Chart

```
helm upgrade --install aira -n aiq deploy/helm/aiq-aira -f deploy/helm/aiq-aira/values-rtx-pro-6k.yaml \
```

```
--set imagePullSecret.password=$NGC_API_KEY \
--set ngcApiSecret.password=$NGC_API_KEY \
--set tavilyApiSecret.password=$TAVILY_API_KEY
```

Once AI-Q has been deployed, make sure all the pods are up and running.

We then need to make the frontend and the NIM-llm pod available with an IP so we can reach it from outside the cluster. We also need to expose the Phoenix pod to trace the benchmarking tests as well as the instruct-llm in AI-Q

```
kubectl patch svc aira-aira-frontend -n aiq -p '{"spec": {"type": "LoadBalancer", "ports": [{"name": "http", "port": 3001, "NodePort": 30001}]}}'

kubectl patch service instruct-llm -n aiq \
  -p '{"spec":{"type":"LoadBalancer","ports":[{"port":8000,"targetPort":8000}]}}'


kubectl patch service aiq-aira-phoenix -n aiq \
  -p '{"spec":{"type":"LoadBalancer","ports":[{"port":60006,"targetPort":6006}]}}'
```

By now we should have AI-Q deployed and all the services loadbalanced. We are now going to ingest two datasets within the system using NeMo Retriever extraction.

Ingesting Enterprise Data for Research

RAG and AI-Q blueprint have been configured and deployed, we now need to ingest Enterprise files, data that can be used by AI-Q to conduct deep research.This will be done using NeMo Retriever extraction, an open source library that is part of the RAG pipeline. Individual files can be added to RAG in a single Collection, or multiple collections can be created as well.

The AI research assistant demo web application requires two default collections. One collection supports a biomedical research prompt and contains reports on Cystic Fibrosis. The second supports a financial research prompt and contains public financial documents from Alphabet, Meta, and Amazon.

To load these default collections, apply the standalone Kubernetes job:

Set the RAG_INGEST_URL environment variable based on your RAG deployment. First we need to port forward the rag ingestor service

```
kubectl port-forward -n rag service/ingestor-server 8082:8082
export RAG_INGEST_URL="http://localhost:8082"
```

On the same node where RAG_INGEST_URL was set above, create a Python environment with the correct dependencies:

```
uv python install 3.12
uv venv --python 3.12 --python-preference managed
uv run pip install -r data/requirements.txt
```

Copy the Enterprise zip files intended for user research into the current directory for upload. These files will be ingested into the RAG system, enabling AI-Q users to conduct in-depth research.

Note: This folder already has the default Financial and Bio Medical Collection needed for the benchmarking of this folder

```
cd data
cp files/* .
```

Run the ingest:

```
uv run python zip_to_collection.py
```

By now the RAG and AI-Q have been installed and the data needed for deep research is already embedded and ingested into Milvus.

Benchmarking and Scale Methodology

NVIDIA AI-Q is an agentic solution based on the NVIDIA NeMo™ Agent Toolkit . NVIDIA NeMo Agent Toolkit is an open-source framework for building, profiling, and optimizing agents and tools for agentic workflows. NeMo Agent Toolkit also has a profiling and benchmarking tool that can be used to determine how an agentic workflow is performing on a given architecture.

For this guide, we used the NeMo Agent Toolkit Sizing and profiling function to test simultaneous concurrent users requesting deep research reports on various kinds of topics. The profiler in the NeMo Agent Toolkit, collects usage statistics in real time. These stats include the time each session/user took to get the detailed report back from AI-Q, what the LLM Latency was compared to the overall Workflow latency.

NeMo Agent Toolkit provides the following metrics that tells how a agentic workflow is performing at a given concurrency level (concurrent users utilizing the workflow)

- The **P95 LLM Latency** (95th percentile LLM latency) column contains the latency, in seconds, across all LLM invocations. If multiple models are used, the value will trend towards the latency of the model with the highest latency.
- The **P95 WF Runtime** (95th percentile workflow runtime) column contains the response time, in seconds, of the workflow and is computed across all runs at the specified concurrency.
- The **Total Runtime** column contains the total time, in seconds, taken to process the entire dataset at a specified concurrency level.

NeMo Agent Toolkit sizing tool can be given how many concurrent users to run, here is a sample out put

Note: AI-Q also has man in the middle function, this function helps a user refine the created report further to their liking, the benchmarking tests done here did not test this feature.

Setup NeMo Agent Toolkit to benchmark AI-Q

On a linux system that's outside the Enterprise RA cluster, setup NeMo Agent Toolkit

Clone the NeMo Agent Toolkit repository to the benchmarking system

```
git clone -b main https://github.com/NVIDIA/NeMo-Agent-Toolkit.git nemo-agent-toolkit
```

```
cd nemo-agent-toolkit
```

Initialize, fetch, and update submodules in the Git repository

```
git submodule update --init --recursive
```

Create a Python environment

```
uv venv --python 3.12 --seed .venv
source .venv/bin/activate
```

Install the NeMo Agent Toolkit library. To install the NeMo Agent Toolkit library along with all of the optional dependencies. Including developer tools (--all-groups) and all of the dependencies needed for profiling and plugins (--all-extras) in the source repository, run the following:

```
uv sync --all-groups --all-extras
```

In addition to plugins, there are optional dependencies needed for profiling. To install these dependencies, run the following:

```
uv pip install -e '.[profiling]'
```

Load Python modules needed for NeMo Agent Toolkit to talk to AI-Q Workflows

```
uv pip install --no-deps -e ../aiq-research-assistant/aira
```

Getting Started With Sizing a GPU Cluster

To begin, set the configuration file and output directory.

```
cd <NAT-root-directory>
export CALC_OUTPUT_DIR=.tmp/sizing_calc/
```

```
export CONFIG_FILE=${CALC_OUTPUT_DIR}aiq-test.yml
mkdir -p ${CALC_OUTPUT_DIR}
```

Datasets needed for prompt during sizing

The sizing tool config file, needs a dataset that has a list of prompts that it can use to run concurrent users tests. This is a JSON format file and can have any no. of entries. For benchmarking we used the dataset.json file in Appendix C. This file needs to be copied into the NeMo Agent Toolkit folder under .tmp directory

Configuration File for NeMo Agent Toolkit to run sizing

We need to add a evaluation config file to let NeMo Agent Toolkit know where the workflow is, what are the different LLMs to test, how to access the LLM, and what datasets to use,please add a YAML config file called aiq-test-rtxpro6k.yml in the .tmp/sizing_calc folder in NeMo Agent Toolkit.

This file, provided in Appendix D has the evaluation sections, please update the config file with the right IP/endpoints for NIM-LLM from RAG, the Instruct-LLM from AI-Q and the RAG server URL. Also, if needed update the Phoenix app endpoint to send traces for the benchmarking tests

Run Benchmarking

Activate virtual environment

```
cd ~/tmp/NeMo-Agent-Toolkit
source .venv/bin/activate
uv pip install --no-deps -e ../../aiq-research-assistant/
```

Export Tavily API key

```
export TAVILY_API_KEY="tvly-prod-redacted"
```

Run The Sizing calculator

The example below shows a concurrency run for 1 and 5, this can be changed or more concurrencies added to the command

```
nat sizing calc   --config_file .tmp/sizing_calc/aiq-test-rtxpro6k.yml   --calc_output_dir
.tmp/sizing_calc/concurrency_era_1_5   --concurrencies 1,5
```

Scale Methodology

To establish the benchmarking on RTX PRO 6000 BSE GPUs, we started load testing AI-Q with a 1X scale of all the components, this means on the 9 node cluster, every component deployed between RAG and AI-Q was running at 1X scale. We started NeMo Agent Toolkit sizing to get NIM Latency and Workflow Latency at different incremental concurrency ranges at 1X scale, the concurrency we ran was from 2, 4, 8 etc till 256 users. We would then scale the reasoning NIM to 2X scale, doubling the total no. of pods and hence GPU consumption and fetch the NIM Latency and Workflow latency at the same concurrencies. The goal was to make sure the LLM Latency and the Workflow latency dropped as we scaled the reasoning NIM LLM pods. We did this till we maxed out all the GPUs in the cluster at 32X NIM LLM scale, where all the 72 GPUs we consumed.

Table 3. Scale and Benchmarking runs tested

Overall Scale	Reasoning NIM Scale (Nemotron Super 49B)	Instruct NIM Scale (Meta Lama 70B)	Precision	Other RAG Components Scale	Concurrency
1X	1 Pod (2 GPUs)	1 Pod (1 GPU)	fp8	1X	2,4,8,16,32,64,96,128,256
2X	2 Pod (4 GPUs)	1 Pod (1 GPU)	fp8	1X	2,4,8,16,32,64,96,128,256
4X	4 Pod (8 GPUs)	1 Pod (1 GPU)	fp8	1X	2,4,8,16,32,64,96,128,256
8X	8 Pod (16 GPUs)	1 Pod (1 GPU)	fp8	1X	2,4,8,16,32,64,96,128,256
16X	16 Pod (32 GPUs)	1 Pod (1 GPU)	fp8	1X	2,4,8,16,32,64,96,128,256

32X	32 Pod (64 GPUs)	1 Pod (1 GPU)	fp8	1X	2,4,8,16,32,64,96,128,256
Other Scale Options Tested					
1X	1 Pod (2 GPUs)	2 Pod (2 GPU)	fp8	1X	2,4,8,16,32,64
1X	1 Pod (2 GPUs)	1 Pod (1 GPU)	fp8	Reranker 2X	2,4,8,16,32,64
1X -32X	1 Pod (2 GPUs)-32 Pod (54 GPUs)	1 Pod (1 GPU)	nvfp4	1X	2,4,8,16,32,64

We also ran scaling exercises where we would scale the Instruct LLM instead on the Reasoning NIM to check the overall impact at different concurrencies. We even scaled Nemotron reranker in RAG's pipeline to test how that impacts the overall latency of the workflow. Earlier benchmarking tests were done with bf16 precision as well.

Scale the reasoning NIM LLM from 1 X to 2X, please wait till all the pods are online.

```
kubectl scale statefulset rag-nim-llm --replicas=2 -n rag
```

Check if all the NIM -LLM pods are running

```
Kubectl get pods -n rag
```

Benchmarking and Scale testing Results for AI-Q

Performance tests were conducted using different numbers of concurrent users. The results, shown below, were generated by scaling various components from both the RAG and AI-Q blueprints. A few observations based on the runs

- On an average, each concurrent user call in AI-Q was generating ~17,000 -24,000 output tokens per session, so the overall ISL/OSL for AI-Q was 20/20000, see Figure 6 from the Phoenix server capturing spans and tokens
- AI-Q does at least two rounds of reasoning rounds to check for relevancy of the output tokens generated, see Figure 7 capturing the overall workflow
- We initially used precision of bf16 and moved to fp8 precision for NIM this improved the overall latency by ~20%
- When a large number of requests come to the Nemotron Super 49B NIM pod, the backend service of the pod limits the total number of requests to 4 for latency profile, and to 5 for throughput profile, this is cause of the optimizations done on the NIM for RTX PRO 6000 GPUs, keeping the host framebuffer memory in mind and also the KV cache usage needed to store context. As a result of this, few requests are waiting in the queue to process. The results here capture the p95 latency numbers, however the average latency is almost half the p95 latency. For example, the requests that get in the first batch complete the workflow in 6 minutes, however the requests that are in queue can take ~17-20 minutes.

Figure 6. Phoenix app showcasing the average tokens per session and P50/P95 latencies

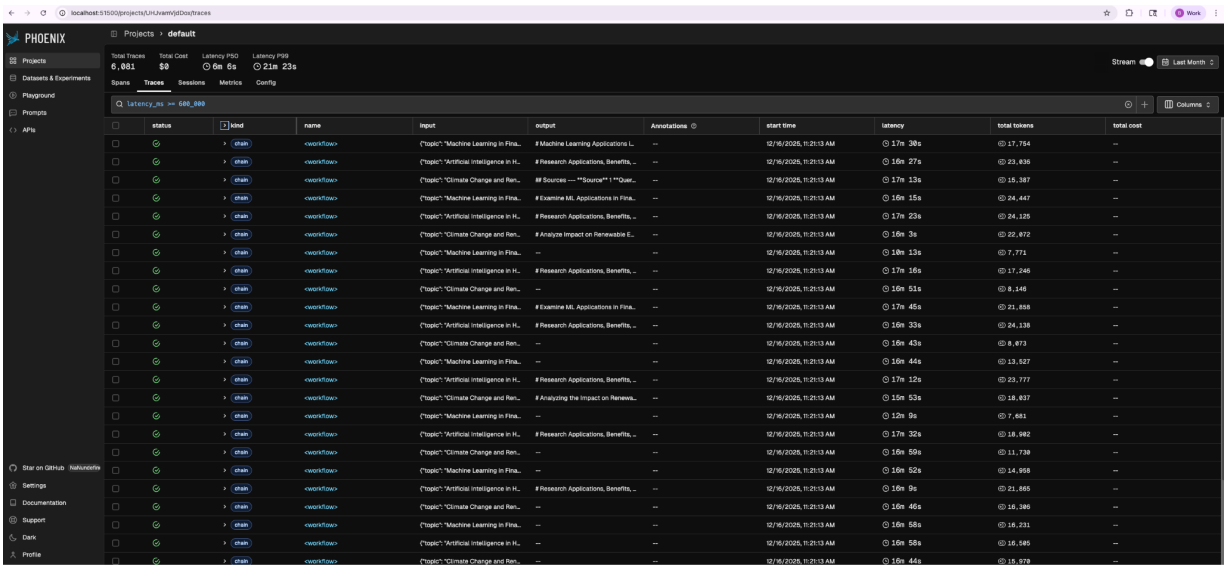
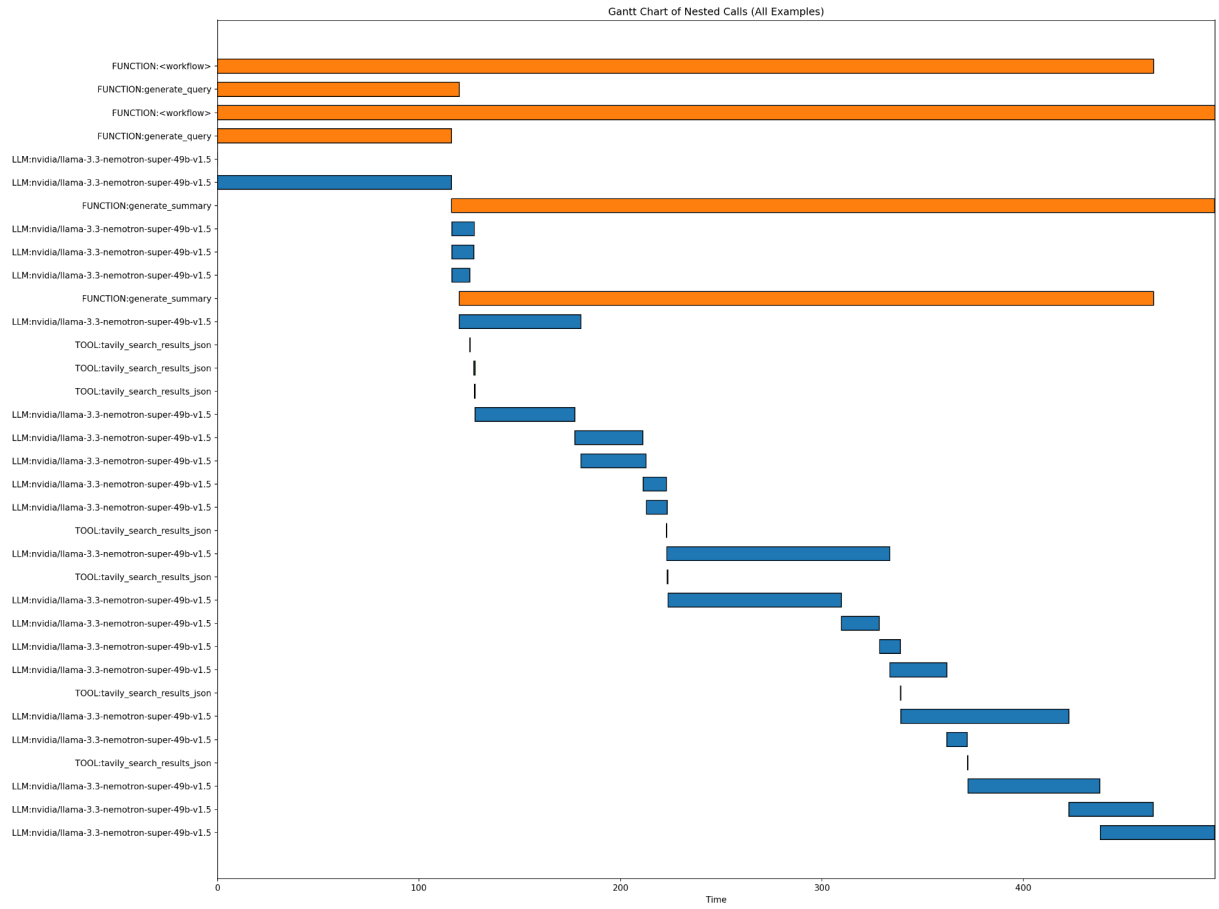


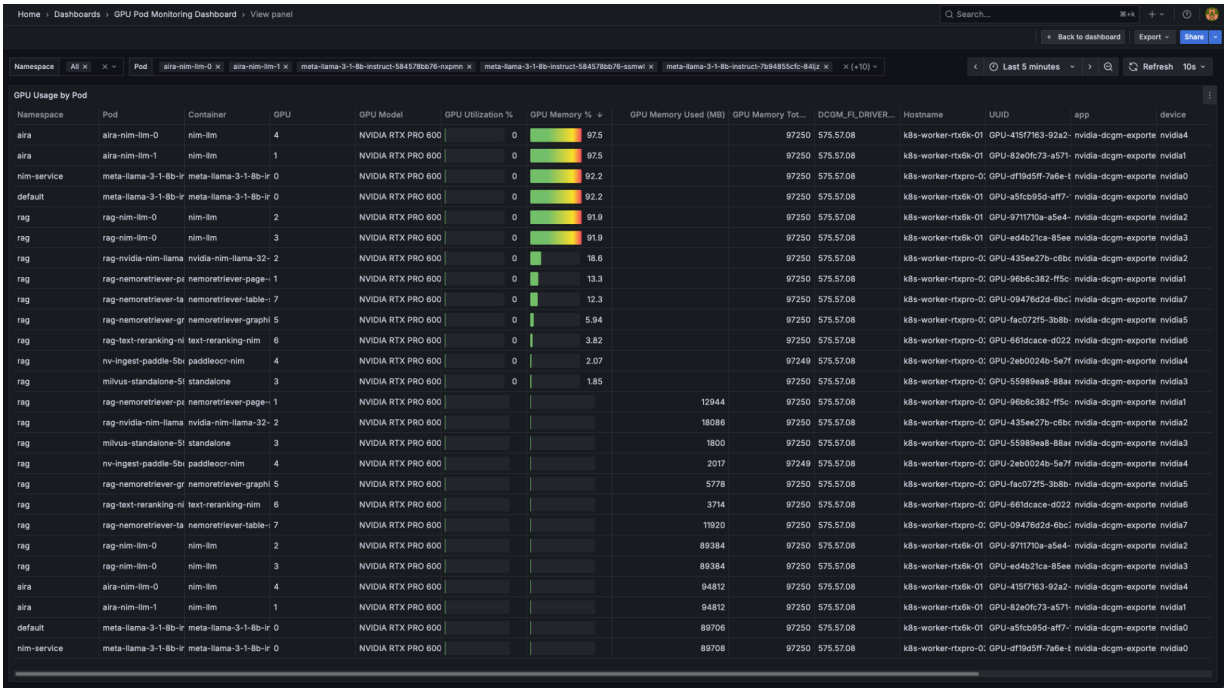
Figure 7. Gantt chart showcasing where the workflow spends time



Latency Impact of Reasoning Model Scale

The Nemotron Super 49B reasoning model significantly influences both overall workflow latency and NIM LLM latency. Scaling the Reasoning Model NIM resulted in a latency reduction of approximately 10-30% for the same number of concurrent users. This is when the precision of the model is already set to fp8. Most other components part of the RAG and AI-Q blueprint did not reach max usage when running benchmarking tests. See Figure 8 for the overall GPU consumption of various components in the system. While AI-Q Nemotron Instruct NIM, was also high in usage, it did not contribute primarily to the overall workflow latency.

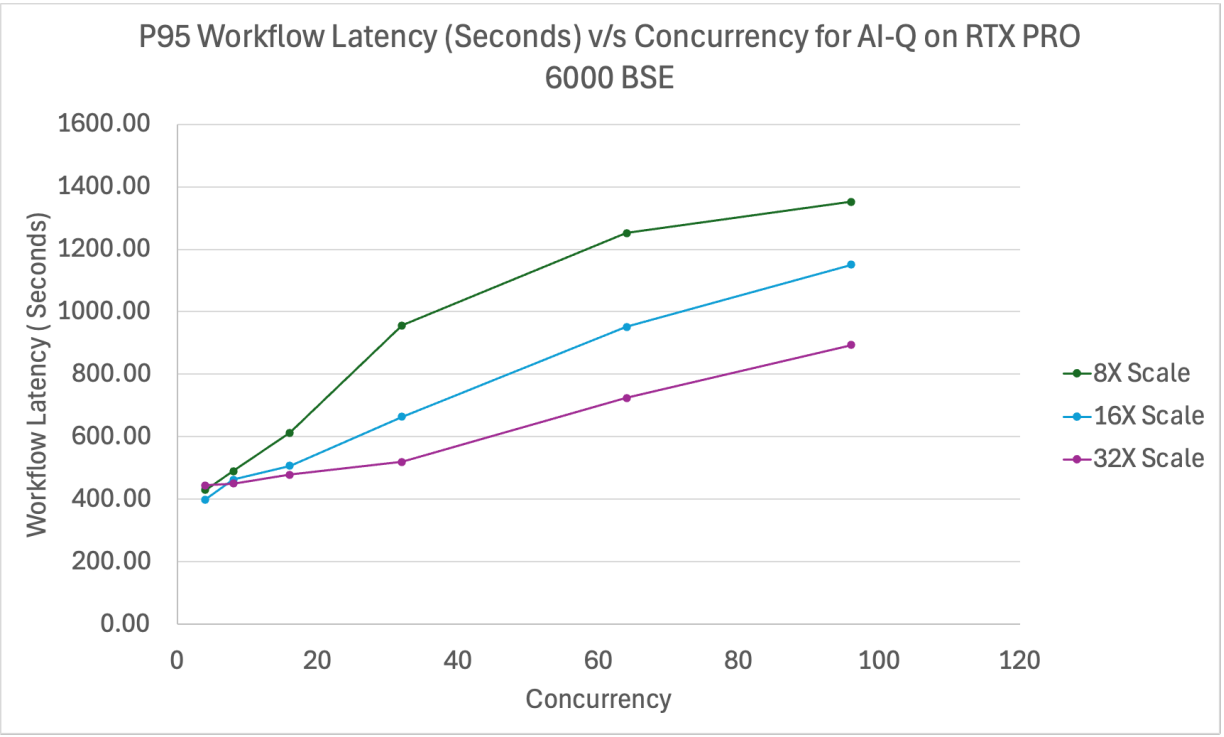
Figure 8. Overall GPU usage of various components



Latency v/s Concurrent Users at Scale

As we scaled the reasoning LLM pods from 1X till 32X, the overall workflow latency kept increasing as more concurrent users were added. Concurrent users at 96 were above the knee for latency. Concurrent users above 96 showed a higher drop in latency. See Figure 9. for reference. Overall, at 32X the workflow latency was lower by approximately ~30 % than at 2X scale.

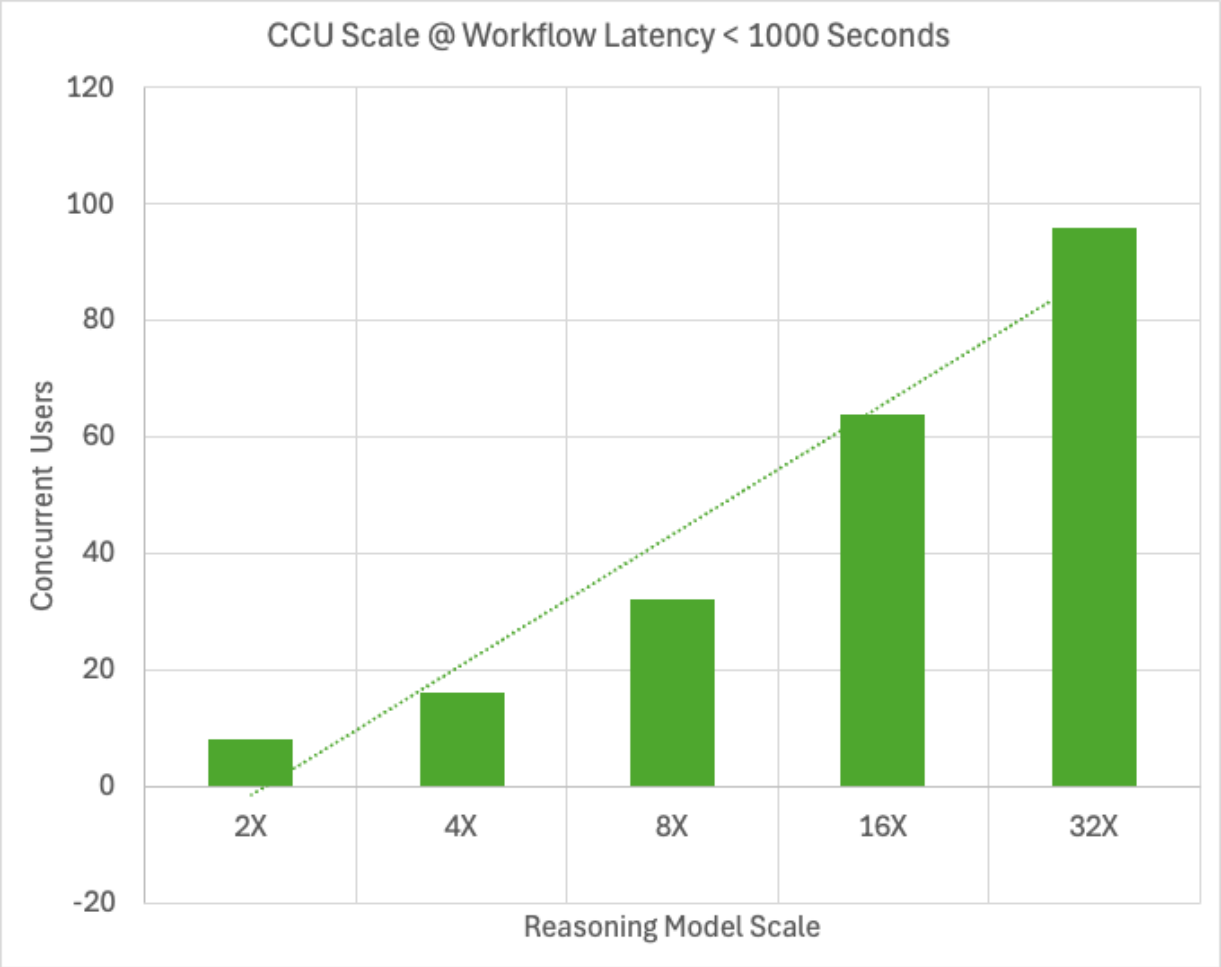
Figure 9 . Workflow Latency v/s Concurrent Users at scale



AI-Q scales Linearly

The AI-Q system demonstrated linear scalability with the Reasoning LLM NIM. Specifically, when the Reasoning LLM NIM was doubled (2X scale), the system could simultaneously handle twice the number of concurrent users—up to eight—while maintaining the workflow latency below 1000 seconds for individual users generating reports. This established that AI-Q's scaling performance is directly proportional to the system's scaling. Refer to Figure 10 for the benchmarked concurrent user capacity achieved under the 1000-second workflow latency constraint.

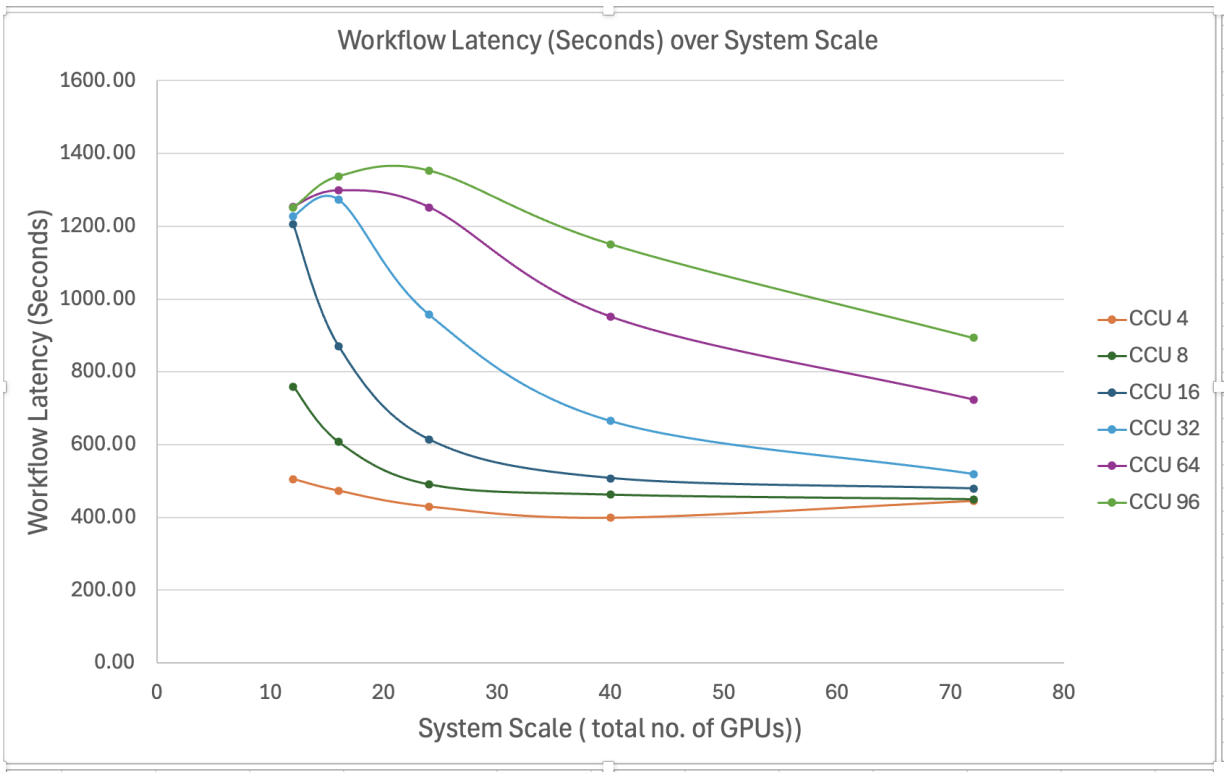
Figure 10. Linear concurrent users with System Scale



Latency Drops as Systems Scale

The performance analysis demonstrated a clear correlation between the scaling of the Reasoning Large Language Model (LLM) and a reduction in critical latency metrics. Specifically, for any defined level of concurrent users, increasing the Reasoning LLM resulted in a measurable drop in both the overall LLM processing latency and the end-to-end Workflow latency.

Figure 11. Workflow Latency dropped for a specific concurrency at scale



Sizing Guideline

Below is the sizing guidance for an Enterprise RA 2-8-5-200 Cluster with RTX PRO 6000 GPUs , a minimum of two nodes will allow for 16 simultaneous users to build reports while keeping total workflow latency below 1000 Seconds. As users grow, scaling up the Reasoning LLM pods is recommended to keep the workflow latency under 1000 seconds. This is based on the Biomedical and Finance datasets, as the datasets grow, certain components like Milvus, Nemotron reranker etc. will have to be scaled as well.

Table 4 . Sizing RTX PRO 6000 BSE for AIQ on Enterprise RA 2-8-5-200 Architecture

Reasoning Model Scale	Nodes (Worker)	RTX PRO 6000 BSE GPUs	Concurrency	Workflow Latency (Seconds)	LLM Latency (Seconds)	Estimated Throughput (Cumulative Tokens)
2X	2	12	8	759.38	112.85	152000
4X	2	16	16	869.99	117.38	304000
8X	3	24	32	956.40	120	608000
16X	5	40	64	951.35	116.60	1216000
32X	9	72	100	1000	133	1900000

Conclusion

AI-Q blueprint is an agentic workflow that leverages an Enterprises existing multi modal data to create deep research reports along with internet search. The agentic workflow leverage the Nemotron Super 49B reasoning model to reflect and think about the results before generating a final report, this creates a very high volume of tokens per user session (Approx Average 19000 tokens). For deep research reports the time it takes to generate tokens is comparatively higher than most LLM chat/summary use cases. RTX PRO 6000 GPUs work for uptill peak usage of 100 users for a latency SLA of 1000 Seconds. Latency can grow for concurrencies above 128 users.

To scale and get the most efficiency out of the System for AI-Q

- Scaling the reasoning model, Nemotron 49B, has the biggest impact on lowering workflow TCO. As more users are added, scale the Nemotorn 49B NIM LLM pod to keep consistent workflow latency
- Select NIM profiles with fp8 precision using TRT-LLM backend and Tensor Parallelism of 2 over vLLM backed profiles.
- Linear scale can be achieved by scaling the reasoning LLM pods

Appendix A

RAG values.YAML for Helm Deployment.

Note: Please replace the node name in the node-selector field to the host name of a node in the cluster. This will allow all non NIM-LLM pods to be scheduled on a single node. Makes it easy to scale the NIM-LLM pod later on.

```
# Custom values for 2 GPU deployment with RTX Pro 6000 GPUs
# This configuration allocates 2 full GPUs across key services on RTX Pro 6000 nodes
# Using Nemotron 49B version 1.5 NIM


# Allocate 2 GPUs to the main LLM NIM for Nemotron 49B v1.5
nim-llm:
 enabled: true
 image:
  repository: nvcr.io/nim/nvidia/llama-3.3-nemotron-super-49b-v1.5
  tag: "1.14.0"
 resources:
  limits:
   nvidia.com/gpu: 2
   memory: 128Gi  # Increased from 96Gi
  requests:
   nvidia.com/gpu: 2
   memory: 128Gi  # Increased from 96Gi
 model:
  name: "nvidia/llama-3.3-nemotron-super-49b-v1.5"
 env:
  - name: NIM_MODEL_NAME
   value: "nvidia/llama-3.3-nemotron-super-49b-v1.5"
  - name: NIM_MODEL_PROFILE
```

```
    value: "610f006b15f3adbdb072da0b4155d8a772332cf1768fb7389ef92a83c31c26dc"
  # - name: NIM_HTTP_MAX_WORKERS
  #   value: "10"


# Keep other services on single GPU
nvidia-nim-llama-32-nv-embedqa-1b-v2:
 enabled: true
 nodeSelector:
   kubernetes.io/hostname: pdx-2852-w01-lr6
 resources:
   limits:
     nvidia.com/gpu: 1
   requests:
     nvidia.com/gpu: 1


text-reranking-nim:
 enabled: true
 nodeSelector:
   kubernetes.io/hostname: pdx-2852-w01-lr6
 resources:
   limits:
     nvidia.com/gpu: 1
   requests:
     nvidia.com/gpu: 1


# Disable VLM if not needed to save GPU resources
nim-vlm:
 enabled: false
```

```
ingestor-server:
 nodeSelector:
  kubernetes.io/hostname: pdx-2852-w01-lr6
 # Keep ingestion services on single GPU each
 nv-ingest:
  nodeSelector:
   kubernetes.io/hostname: pdx-2852-w01-lr6
  paddleocr-nim:
   nodeSelector:
    kubernetes.io/hostname: pdx-2852-w01-lr6
   resources:
    limits:
     nvidia.com/gpu: 1
    requests:
     nvidia.com/gpu: 1

  nemoretriever-graphic-elements-v1:
   nodeSelector:
    kubernetes.io/hostname: pdx-2852-w01-lr6
   resources:
    limits:
     nvidia.com/gpu: 1
    requests:
     nvidia.com/gpu: 1

  nemoretriever-page-elements-v2:
   nodeSelector:
```

```
    kubernetes.io/hostname: pdx-2852-w01-lr6
  resources:
    limits:
      nvidia.com/gpu: 1
    requests:
      nvidia.com/gpu: 1

nemoretriever-table-structure-v1:
  nodeSelector:
    kubernetes.io/hostname: pdx-2852-w01-lr6
  resources:
    limits:
      nvidia.com/gpu: 1
    requests:
      nvidia.com/gpu: 1

milvus:
  standalone:
    nodeSelector:
      kubernetes.io/hostname: pdx-2852-w01-lr6
    resources:
      limits:
        nvidia.com/gpu: 1

redis:
  image:
    repository: redis
    tag: 8.2.1
```

Appendix B

AI-Q Helm values file

```
# ------------------------------------------------------
# The following values are for the AIQ AIRA backend service.
# ------------------------------------------------------

replicaCount: 1

# The name of the image pull secret to use for the AIQ container images.
# Either create the secret manually and update the name here
# or update the imagePullSecret.password with your NGC API key
ngcImagePullSecretName: "ngc-secret"

imagePullSecret:
 create: true
 name: "ngc-secret"
 registry: "nvcr.io"
 username: "$oauthtoken"
 password: "" #UPDATE THIS

# The image repository and tag for the AIQ AIRA backend service.
image:
 repository: nvcr.io/nvidia/blueprint/aira-backend
 tag: v1.1.0
 pullPolicy: IfNotPresent

# The service type and port for the main AIQ AIRA backend service
service:
 port: 3838

# Update each value according to your desired configuration.
config:
 # The instruct_ settings are for the general purpose Q&A LLM
```

```
  instruct_model_name: "meta/llama-3.3-70b-instruct"
  instruct_temperature: "0.0"
  instruct_api_key: "not-needed"
  instruct_base_url: "http://nim-llm.aira.svc.cluster.local:8000/v1"
  # The nemotron_ settings are for the reasoning LLM
  nemotron_api_key: "not-needed" # not needed as we use the nemotron service from the RAG deployment which does not require an API key
  nemotron_model_name: "nvidia/llama-3.3-nemotron-super-49b-v1.5"
  nemotron_temperature: "0.5"
  nemotron_base_url: "http://nim-llm.rag.svc.cluster.local:8000/v1" # provided by the RAG deployment
  nemotron_max_tokens: "5000"
  nemotron_stream: "true"
  # Enter your Tavily API key here to enable web search
  tavily_api_key: "" #UPDATE THIS
  # Enter the IP address of the RAG services
  rag_ingest_url: "http://ingestor-server.rag.svc.cluster.local:8082" # provided by the RAG deployment
  rag_url: "http://rag-server.rag.svc.cluster.local:8081" # provided by the RAG deployment
  rag_api_key: "" #Typically not required
  milvus_host: "milvus.rag.svc.cluster.local" # provided by the RAG deployment
  milvus_port: "19530"

# Do not update this command. It is the default command to launch the AI-Q backend service.
command: "/entrypoint.sh"



# ------------------------------------------------------
# The following values are for the instruct LLM service
# The nemotron llm is assumed to be deployed via the RAG helm chart
# ------------------------------------------------------

ngcApiSecret:
  name: "ngc-api"
  password: "" # UPDATE THIS
  create: true

nim-llm:
  enabled: true
  service:
```

```
  name: "nim-llm"
image:
    repository: nvcr.io/nim/meta/llama-3.3-70b-instruct
    pullPolicy: IfNotPresent
    tag: "1.12.0"
resources:
  limits:
    nvidia.com/gpu: 1
  requests:
    nvidia.com/gpu: 1
model:
  name: "meta/llama-3.3-70b-instruct"
env:
  - name: NIM_MODEL_NAME
    value: "meta/llama-3.3-70b-instruct"
  - name: NIM_MODEL_PROFILE
    value: "257a3035dbddb2a2cd48f5763ee64f972af4ba0dd5ef92ade1dbb478f6cf5dd3"



# ------------------------------------------------------
# The following values are for the nginx proxy that enables the AIQ frontend
# to interact with both the AIQ AIRA backend service and the RAG service
#
# You may need to update the RAG service IP address if you have not deployed RAG via helm on the same cluster
# ------------------------------------------------------

nginx:
  nginxImage:
    ngcImageRegistry: ""
    ngcImageRegistryPath: ""
    name: "nginx"
    tag: "1.27.0"
    pullPolicy: Always

service:
  port: 8051
```

```
nginx_config:
  conf: |-
    worker_processes auto;

    events {
        worker_connections 1024;
    }

    http {
        proxy_ssl_server_name on;

        proxy_cache_path /server_cache_llm levels=1:2 keys_zone=llm_cache:10m max_size=20g inactive=14d
use_temp_path=off;

        proxy_cache_path /server_cache_intel levels=1:2 keys_zone=intel_cache:10m max_size=20g
inactive=14d use_temp_path=off;

        error_log /dev/stdout info;

        log_format upstream_time '$remote_addr - $remote_user [$time_local] '
                                 '"$request" $status $body_bytes_sent '
                                 '"$http_referer" "$http_user_agent"'
                                 'rt=$request_time uct="$upstream_connect_time" uht="$upstream_header_time"
urt="$upstream_response_time"';

        log_format cache_log '[$time_local] ($upstream_cache_status) "$request" $status - $body_bytes_sent
bytes {$remote_addr} "$http_user_agent" $request_time - $connection_requests. Auth:
$http_authorization';

        log_format no_cache_log '[$time_local] (BYPASSED) "$request" $status - $body_bytes_sent bytes
{$remote_addr} "$http_user_agent" $request_time - $connection_requests. Auth: $http_authorization';

        log_format mirror_log '[$time_local] (MIRROR) "$request" $status - $body_bytes_sent bytes
{$remote_addr} "$http_user_agent" $request_time - $connection_requests. Auth: $http_authorization';

        log_format nvai_cache_log '[$time_local] ($upstream_cache_status) "$request" $status -
$body_bytes_sent bytes {$remote_addr} "$http_user_agent" $request_time - $connection_requests. Auth:
$http_authorization. $upstream_addr';

        map $http_cache_control $cache_bypass {
```

```
    no-cache   1;
}

# Log to stdout and a file for searchability
access_log /dev/stdout cache_log;
access_log /var/log/nginx/access.log cache_log;

error_log /dev/stdout info;
error_log /var/log/nginx/error.log info;

server {
 listen 8051;
 server_name _;

 # Common proxy settings
 proxy_set_header X-Real-IP $remote_addr;
 proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
 proxy_set_header X-Forwarded-Proto $scheme;
 proxy_pass_request_headers on;

 # Common buffer settings
 large_client_header_buffers 4 32k;
 client_header_buffer_size 4k;

 # Common timeout settings
 client_body_timeout 900s;
 client_header_timeout 900s;

 # Common settings for document-related endpoints
 proxy_read_timeout 600s;
 proxy_connect_timeout 60s;
 proxy_send_timeout 600s;
 client_max_body_size 100M;
 proxy_max_temp_file_size 0;
 proxy_buffering on;
 proxy_buffer_size 1M;
 proxy_buffers 100 1M;
```

```
        proxy_busy_buffers_size 2M;

        # Original routes
        location ~ ^/v1/(status|documents|collections) {
            proxy_pass http://ingestor-server.rag.svc.cluster.local:8082/$1$is_args$args;
            proxy_set_header Host http://ingestor-server.rag.svc.cluster.local:8082;
        }

        # Protected routes
        location ~ ^/v2/protected/aiq/v1/(status|documents|collections) {
            proxy_pass http://ingestor-server.rag.svc.cluster.local:8082/$1$is_args$args;
            proxy_set_header Host http://ingestor-server.rag.svc.cluster.local:8082;
        }

        # Special case for files route
        location /v2/protected/aiq/v1/files {
            proxy_pass http://ingestor-server.rag.svc.cluster.local:8082/v1/documents;
            proxy_set_header Host http://ingestor-server.rag.svc.cluster.local:8082;
        }

        # Protected routes AIRA v1
        location ~
^/v2/protected/aiq/v1/((generate_query|generate_summary|artifact_qa|default_collections)(/stream)?)$ {
            proxy_pass http://aira-aira-backend.aira.svc.cluster.local:3838/$1$is_args$args;
            proxy_set_header Host http://aira-aira-backend.aira.svc.cluster.local:3838;
        }

        # Health routes
        location /v2/protected/aiq/keepalive {
            default_type text/plain;
            return 200 "OK";
        }

        location /v2/protected/aiq/health {
            default_type text/plain;
            return 200 "OK";
        }
```

```
        location = /health {
            default_type text/plain;
            return 200 "OK";
        }

        location = /keepalive {
            default_type text/plain;
            return 200 "OK";
        }


        # Catch-all for other protected routes
        location /v2/protected/aiq/ {
            rewrite ^/v2/protected/aiq/(.*) /$1 break;
            proxy_pass http://aira-aira-backend.aira.svc.cluster.local:3838;
            proxy_set_header Host $host;
        }

        # Default location for all other routes
        location / {
            proxy_pass http://aira-aira-backend.aira.svc.cluster.local:3838;
            proxy_set_header Host $host;
        }

        error_page 500 502 503 504 /50x.html;
        location = /50x.html {
            root /usr/share/nginx/html;
        }
    }


}

# -------------------------------------------------------
# The following values are for the AIQ AIRA frontend service.
# -------------------------------------------------------
```

```yaml
# The frontend application is a React web app. We recommend a NodePort so the frontend will be accessible at <your-node-ip>:3001
frontend:
 enabled: true
 # Update the value below to the IP address and port of the nginx service
 proxyUrl: http://aira-nginx.aira.svc.cluster.local:8051
 service:
   port: 3001
   targetPort: 3001

 image:
   repository: nvcr.io/nvidia/blueprint/aira-frontend
   tag: v1.1.0
   pullPolicy: IfNotPresent

 replicaCount: 1

# ------------------------------------------------------
# The following values are optional utility services
# ------------------------------------------------------

# Enables the Phoenix tracing service
phoenix:
 enabled: true
 image:
   repository: arizephoenix/phoenix
   tag: latest
   pullPolicy: IfNotPresent
 resources:
   limits:
     cpu: 500m
     memory: 512Mi
   requests:
     cpu: 200m
     memory: 256Mi
```

Appendix C

Evaluation Dataset , research_assistant_dataset.json


```
[
 {
  "id": 1,
  "question": "{\"topic\": \"Artificial Intelligence in Healthcare\", \"report_organization\": \"Research applications, benefits, challenges, and future prospects\", \"search_web\": true, \"rag_collection\": \"biomedical_dataset\", \"num_queries\": 3, \"llm_name\": \"nemotron\"}",
  "answer": "A comprehensive report on AI in healthcare."
 },
 {
  "id": 2,
  "question": "{\"topic\": \"Machine Learning in Finance\", \"report_organization\": \"Examine ML applications in financial services\", \"search_web\": true, \"rag_collection\": \"financial_dataset\", \"num_queries\": 3, \"llm_name\": \"nemotron\"}",
  "answer": "Analysis of ML in finance."
 },
 {
  "id": 3,
  "question": "{\"topic\": \"Climate Change and Renewable Energy\", \"report_organization\": \"Analyze impact on renewable energy adoption\", \"search_web\": true, \"rag_collection\": \"\", \"num_queries\": 3, \"llm_name\": \"nemotron\"}",
  "answer": "Research on climate change and energy."
 }
]
```

Appendix D

File to run NeMo Agent Toolkit sizing against AI-Q, aiq-test-rtxpro6k.yml

```
general:
 use_uvloop: true
 telemetry:
  tracing:
   phoenix:
    _type: phoenix
    endpoint: http://10.184.203.86:6006/v1/traces
    project: default
front_end:
 _type: fastapi
 endpoints:
  - path: /generate_query
    method: POST
    description: Creates the query
    function_name: generate_query
  - path: /generate_summary
    method: POST
    description: Generates the summary
    function_name: generate_summary
  - path: /artifact_qa
    method: POST
    description: Q/A or chat about a previously generated artifact
    function_name: artifact_qa
  - path: /aiqhealth
```

```
    method: GET
    description: Health check for the AIQ AIRA service
    function_name: health_check
  - path: /default_collections
    method: GET
    description: Get the default collections
    function_name: default_collections
  # Add profiler endpoint
  - path: /analyze_performance
    method: POST
    description: Analyze performance metrics using profiler agent
    function_name: profiler_agent



llms:
 instruct_llm:
  _type: openai
  model_name: meta/llama-3.3-70b-instruct
  temperature: 0.0
  base_url: http://10.184.203.82:8000/v1
  api_key: not-needed
  stream: false
  max_retries: 1  # Default is 3
  timeout: 120
 nemotron:
  _type: openai
  model_name: nvidia/llama-3.3-nemotron-super-49b-v1.5
  temperature: 0.0
```

```
  base_url: http://10.184.203.83:8000/v1
  stream: false
  api_key: not-needed
  max_retries: 1  # Default is 3
  timeout: 120

functions:
 generate_query:
  _type: generate_queries
  llm_name: nemotron

 generate_summary:
  _type: generate_summaries
  rag_url: http://10.184.203.84:8081/v1
  timeout: 60

 artifact_qa:
  _type: artifact_qa
  llm_name: instruct_llm
  rag_url: http://10.184.203.84:8081/v1
  timeout: 60

 # Add profiler agent and related tools
 profiler_agent:
  _type: profiler_agent
  llm_name: instruct_llm
  max_iterations: 4
  max_retries: 3
```

```yaml
  tools:
    - px_query
    - flow_chart
    - token_usage
    - response_composer
  output_dir: .tmp/aiq/aira/profiler_agent/  # Add persistent storage

px_query:
  _type: px_query
  phoenix_url: http://10.184.203.86:6006
  time_window_seconds: 600000
  default_project_name: default

flow_chart:
  _type: flow_chart

token_usage:
  _type: token_usage

response_composer:
  _type: response_composer




workflow:
_type: ai_researcher
timeout: 600
```

```yaml
eval:
 general:
  output_dir: .tmp/eval/aiq-test-rtxpro6k
  dataset:
    _type: json
    file_path: .tmp/research_assistant_dataset.json
  profiler:
    base_metrics: true
    token_uniqueness_forecast: false
    workflow_runtime_forecast: false
    compute_llm_metrics: true
    csv_exclude_io_text: true
    prompt_caching_prefixes:
      enable: false
      min_frequency: 0.1
    bottleneck_analysis:
      enable_nested_stack: false
  # # Evaluators for quality assessment during sizing calculations
 # evaluators:
 #   research_quality_evaluator:
 #     _type: tunable_rag_evaluator
 #     llm_name: instruct_llm  # Using your existing LLM
 #     judge_llm_prompt: |
 #       You are an expert evaluator for AI research assistant outputs. Assess the quality of the generated research content based on:
 #       1. Coverage: How comprehensively does it address the research question?
 #       2. Correctness: How accurate and factual is the information?
 #       3. Relevance: How relevant is the content to the specific research domain?
```

```
#     4. Citation Quality: How well are sources cited and referenced?


#     Rate each aspect from 0.0 to 1.0.
#   default_scoring: true
#   default_score_weights:
#     coverage: 0.3
#     correctness: 0.3
#     relevance: 0.2
#     citation_quality: 0.2
```

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation (“NVIDIA”) makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer (“Terms of Sale”). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer’s own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer’s product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, “MATERIALS”) ARE BEING PROVIDED “AS IS.” NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA’s aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

VESA DisplayPort

DisplayPort and DisplayPort Compliance Logo, DisplayPort Compliance Logo for Dual-mode Sources, and DisplayPort Compliance Logo for Active Cables are trademarks owned by the Video Electronics Standards Association in the United States and other countries.

HDMI

HDMI, the HDMI logo, and High-Definition Multimedia Interface are trademarks or registered trademarks of HDMI Licensing LLC.

ARM

ARM, AMBA and ARM Powered are registered trademarks of ARM Limited. Cortex, MPCore and Mali are trademarks of ARM Limited. All other brands or product names are the property of their respective holders. "ARM" is used to represent ARM Holdings plc; its operating company ARM Limited; and the regional subsidiaries ARM Inc.; ARM KK; ARM Korea Limited.; ARM Taiwan Limited; ARM France SAS; ARM Consulting (Shanghai) Co. Ltd.; ARM Germany GmbH; ARM Embedded Technologies Pvt. Ltd.; ARM Norway, AS and ARM Sweden AB.

OpenCL

OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.

Trademarks

NVIDIA, the NVIDIA logo, are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com