

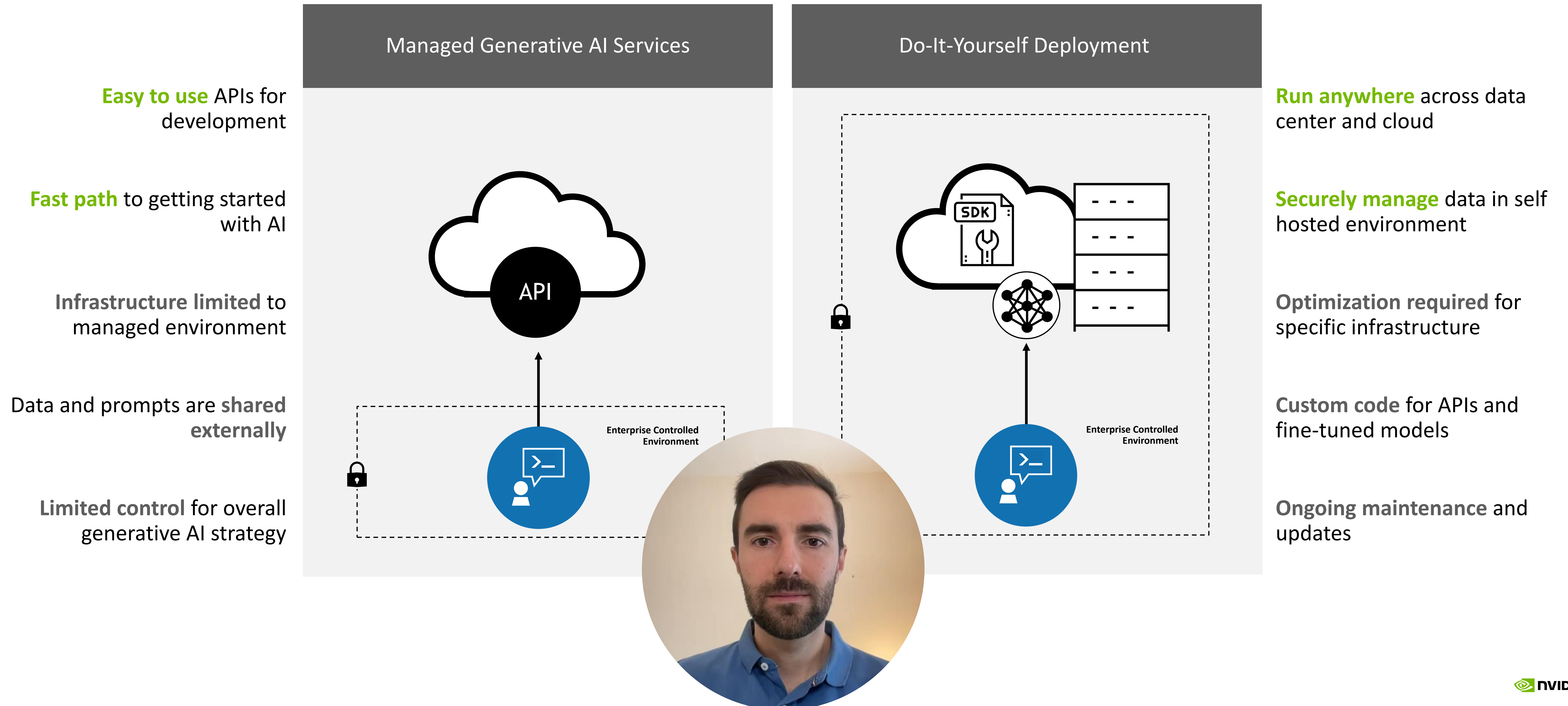
Sizing LLM Inference Systems

First Contact with NIMs



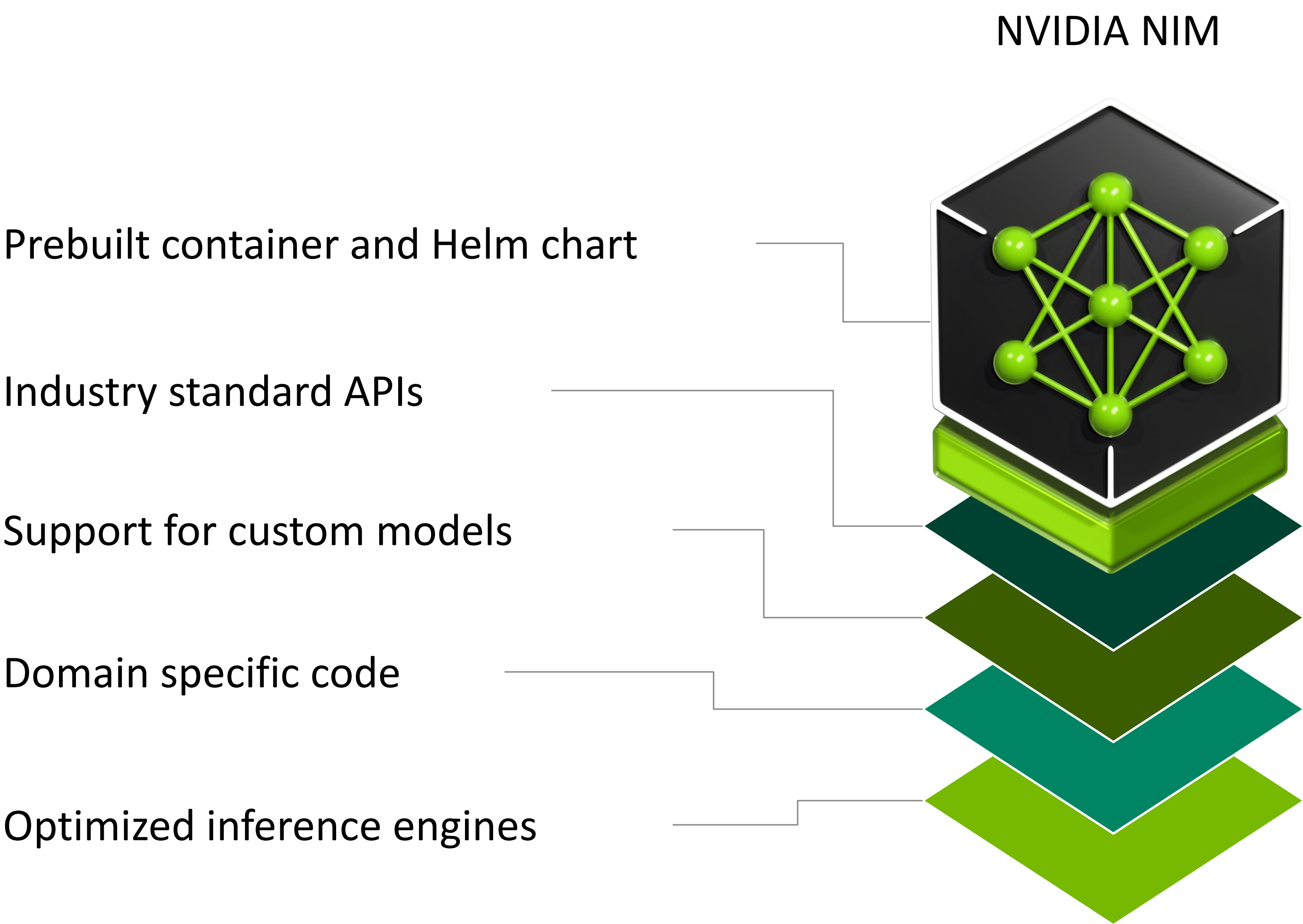
Enterprises Face Challenges Experimenting with Generative AI

Organizations must choose between ease of use and control



NVIDIA NIM Optimized Inference Microservices

Accelerated runtime for generative AI



Deploy anywhere with security and control of AI applications and data

Speed time to market with prebuilt, continuously maintained microservices

Empower developers with the latest AI models, standard APIs and enterprise tools

Optimize throughput and latency to maximize token generation and responsiveness

Boost accuracy by tuning custom models from proprietary data sources

Deploy in production with API stability, security patching, quality assurance and enterprise support




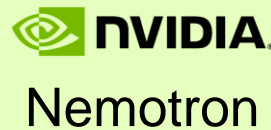


DGX &
DGX Cloud



NVIDIA NIM is the Fastest Path to AI Inference

Reduces engineering resources required to deploy optimized, accelerated models

	NVIDIA NIM	Do It Yourself
Deployment Time	5 minutes	1 week +
API Standardization	Industry standard protocol OpenAI for LLMs, Google Translate for Speech	Implement the API layer for each domain and model family according to industry standard specifications
Optimized Engines	Pre-built engines for NVIDIA and community models    	Build your own engine and manually customize for workload and hardware specific requirements
Pre and Post Processing Pipelines	Pre-built with optimized pipeline engines to handle pre/post processing (tokenization)	Implement custom logic
Model Server Deployment	Automated	Manual setup and configuration
Customization	LoRA is supported, more planned	Create custom logic
Container Validation	Extensive workload specific QA support matrix validation	No validation
Enterprise Support	Delivered with NVIDIA AI Enterprise Security and CVE scanning/patching and tech support	Self supported



NVIDIA NIM for LLM

Deploy an Optimized NIM with a Single Command

- **NVIDIA NIM for LLM offers per-model containers**
- When a model container is launched, it:
 - Detects the hardware it is running on
 - Mounts the cache for model and asset data
 - Selects the most optimal model given the hardware
 - Downloads the optimized model file from NGC
 - Loads the model file and starts serving

Model-specific Config Image
(llama3-8b-instruct)

Base Container Image
(NIM for LLMs)

```
export NGC_API_KEY=<value>
echo "$NGC_API_KEY"
docker login nvcr.io --username '$oauthtoken' --password-stdin

# Choose a container name for bookkeeping
export CONTAINER_NAME=llama3-8b-instruct

# Choose a LLM NIM Image from NGC
export IMG_NAME="nvcr.io/nim/meta/${CONTAINER_NAME}:1.0.0"

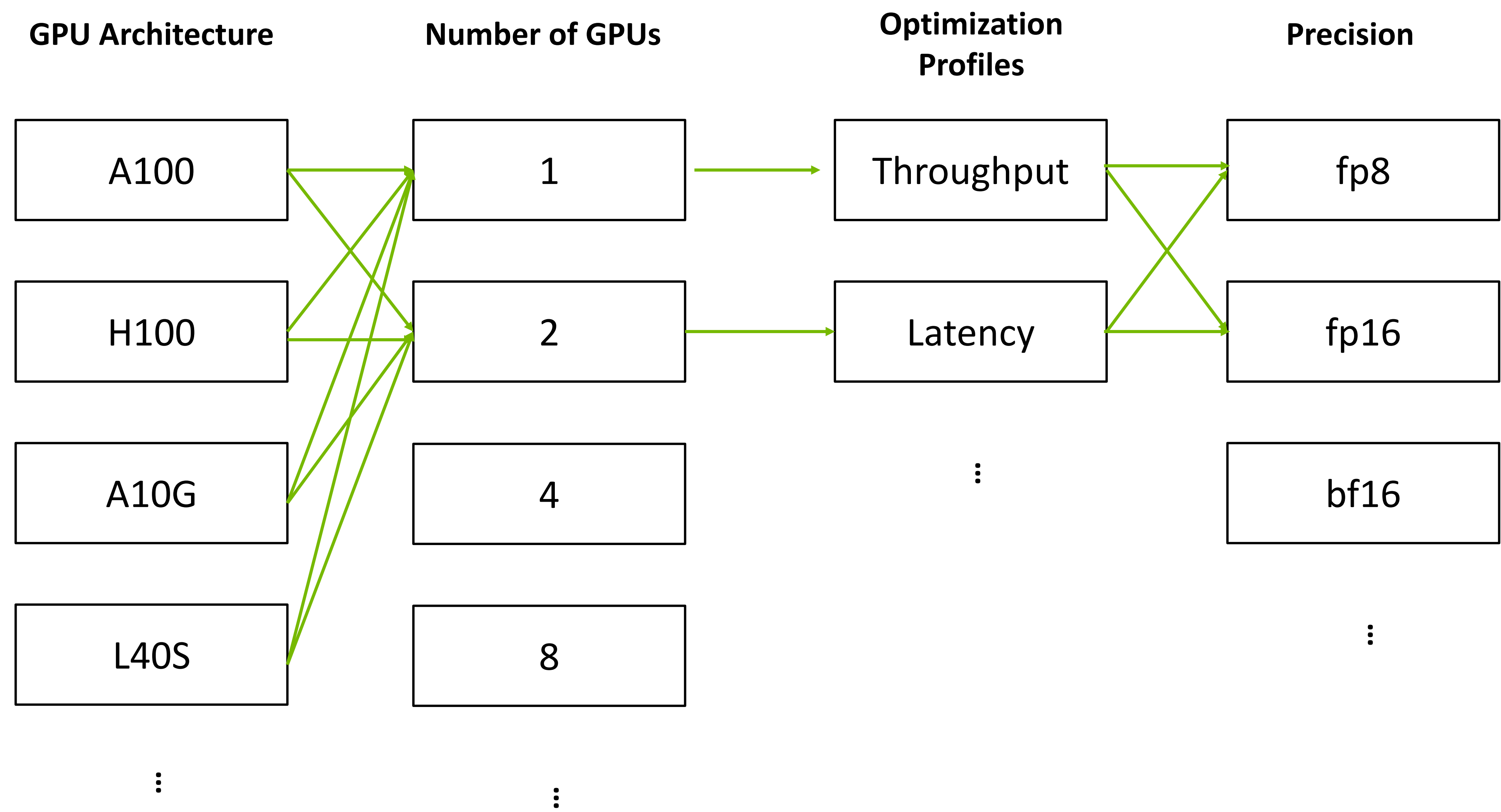
# Choose a path on your system to cache the downloaded models
export LOCAL_NIM_CACHE=~/.cache/nim
mkdir -p "$LOCAL_NIM_CACHE"

# Start the LLM NIM
docker run -it --rm --name=$CONTAINER_NAME \
  --runtime=nvidia \
  --gpus all \
  --shm-size=16GB \
  -e NGC_API_KEY \
  -v "$LOCAL_NIM_CACHE:/opt/nim/.cache" \
  -u $(id -u) \
  -p 8000:8000 \
  $IMG_NAME
```



How NIMs Leverage Optimized Engines

Llama-3-8B-Instruct Optimization for NVIDIA GPUs with LoRA Support Option



Optimized Models

llama3-8b-instruct:0.10.0+cbc614f5-**h100**x1-fp8-**throughput**
llama3-8b-instruct:0.10.0+cbc614f5-**h100**x2-fp8-**latency**
llama3-8b-instruct:0.10.0+cbc614f5-**h100**x1-fp16-**throughput**
llama3-8b-instruct:0.10.0+cbc614f5-**h100**x1-fp16-**lora**

llama3-8b-instruct:0.10.0+cbc614f5-**a100**x1-fp16-**throughput**
llama3-8b-instruct:0.10.0+cbc614f5-**a100**x1-fp16-**lora**
llama3-8b-instruct:0.10.0+cbc614f5-**a100**x2-fp16-**latency**
llama3-8b-instruct:0.10.0+cbc614f5-**h100**x2-fp16-**latency**

llama3-8b-instruct:0.10.0+cbc614f5-**a10g**x2-fp16-**latency**
llama3-8b-instruct:0.10.0+cbc614f5-**a10g**x1-fp16-**throughput**
llama3-8b-instruct:0.10.0+cbc614f5-**a10g**x1-fp16-**lora**
llama3-8b-instruct:0.10.0+cbc614f5-**l40s**x2-fp8-**latency**
llama3-8b-instruct:0.10.0+cbc614f5-**l40s**x2-fp16-**latency**
llama3-8b-instruct:0.10.0+cbc614f5-**l40s**x1-fp8-**throughput**
llama3-8b-instruct:0.10.0+cbc614f5-**l40s**x1-fp16-**throughput**
llama3-8b-instruct:0.10.0+cbc614f5-**l40s**x1-fp16-**lora**

```
Detected 3 compatible profiles.  
Valid profile: 17190a87573ad181e4a55b96d1a84b52f1c82c6542e2404473b25011ebfb403e (tensorrt_llm-A100-bf16-tp1-balanced)  
Valid profile: 4ca08b7e7f1d59d5da5f761969320738d6922f25f26f334ab344c21f2e57348f (tensorrt_llm-A100-fp16-tp1-throughput)  
Valid profile: 15fc17f889b55aedaccb1869bfeadd6cb540ab26a36c4a7056d0f7a983bb114f (vllm)  
Selected profile: 17190a87573ad181e4a55b96d1a84b52f1c82c6542e2404473b25011ebfb403e (tensorrt_llm-A100-bf16-tp1-balanced)  
Profile metadata: llm_engine: tensorrt-llm
```



Notebook Objectives

Notebook 0

1. Get familiar with NIM deployment and usage
2. Call the NIM endpoint with curl and python to generate some text
3. Consider End-to-End Latency (E2E) versus Time-to-First-Token Latency (TTFT)



