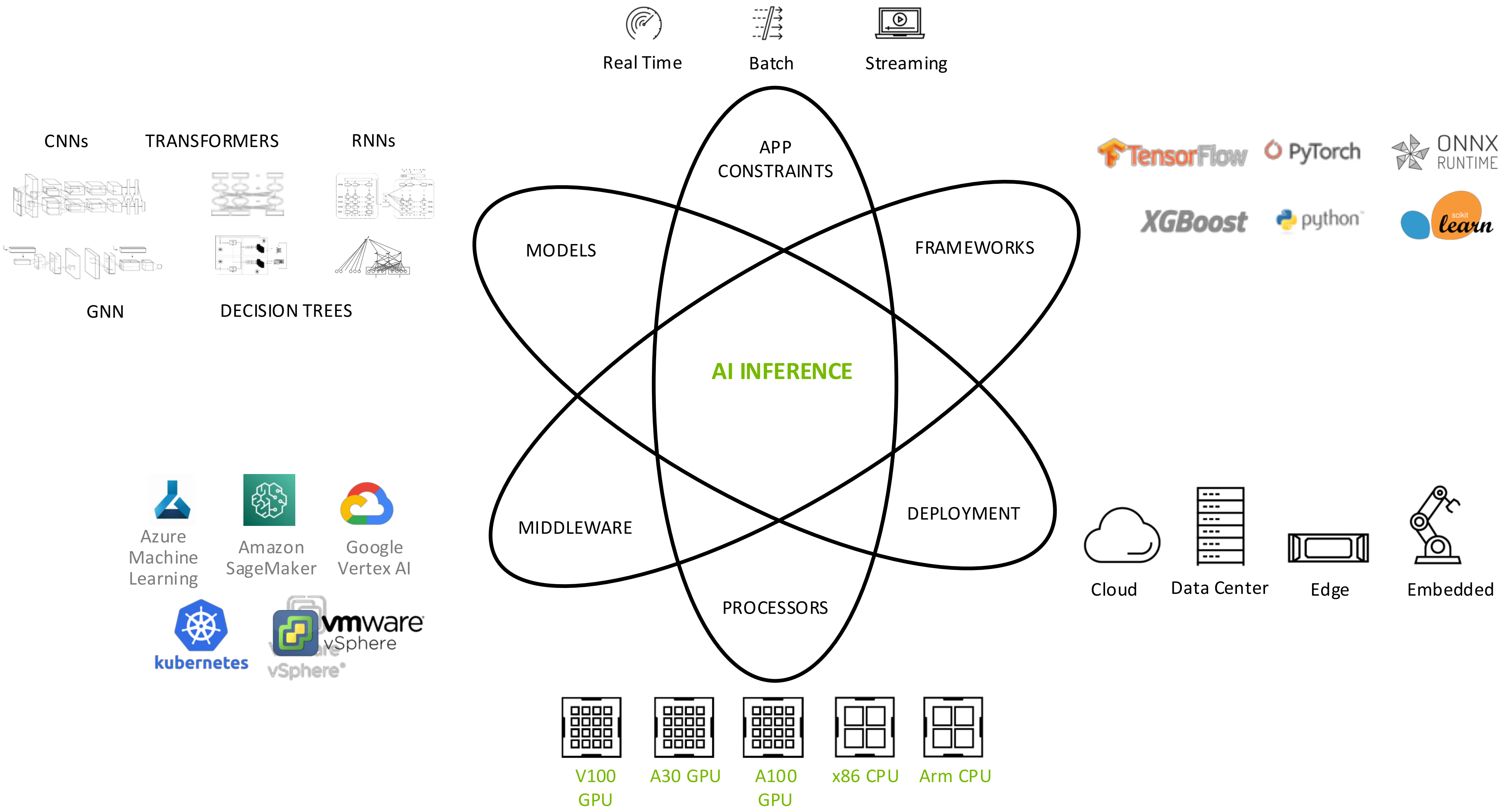
ML Inference on GPUs

Ziv Ilan - Solution Architect, NVIDIA
Sergio Perez - Solution Architect, NVIDIA
Harshita Seth - Solution Architect, NVIDIA

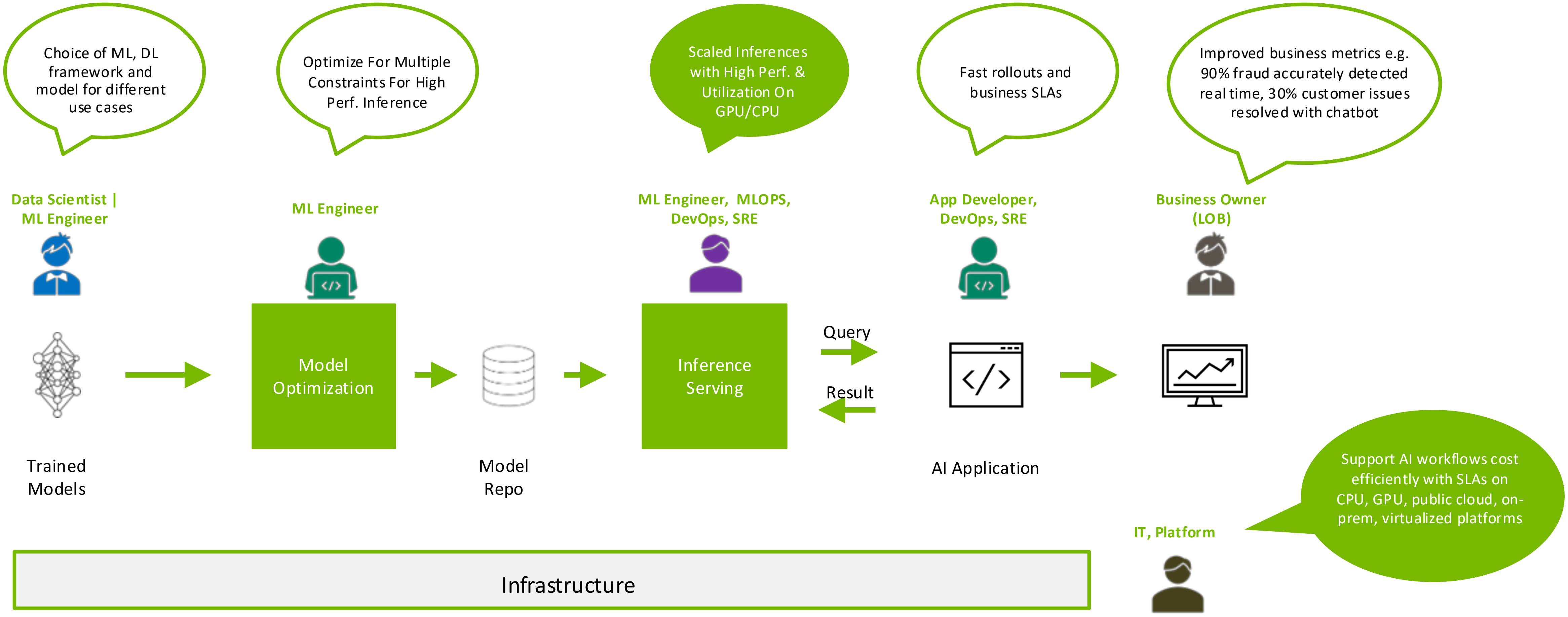
Agenda of ML for inference

- The inference workflow
- Inference optimization with TensorRT
- Inference server with Triton
- NIM to simplify inference

Challenges of AI Inference



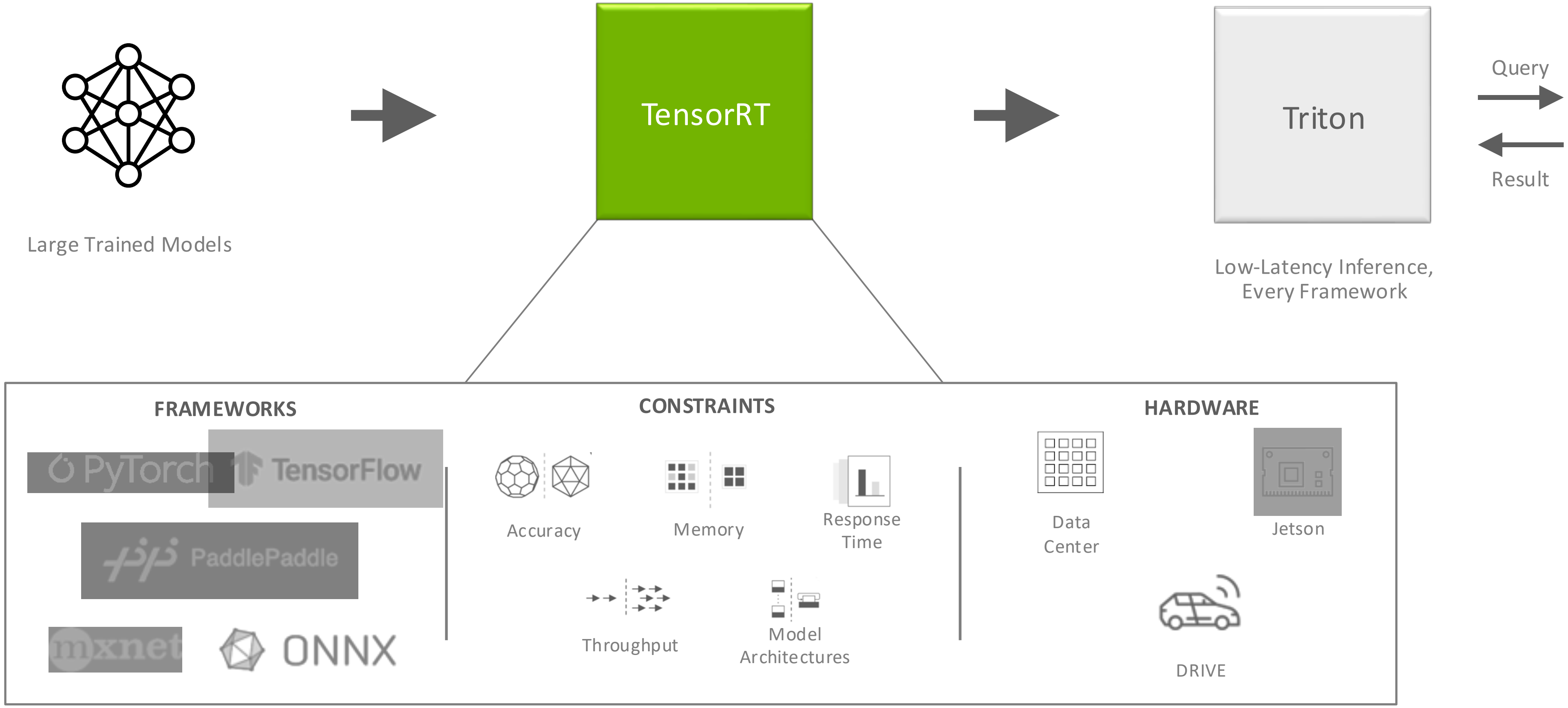
AI Inference Workflow



TensorRT and TensorRT-LLM

Inference is Complex

Real-Time | Competing Constraints | Rapid Updates



NVIDIA TensorRT

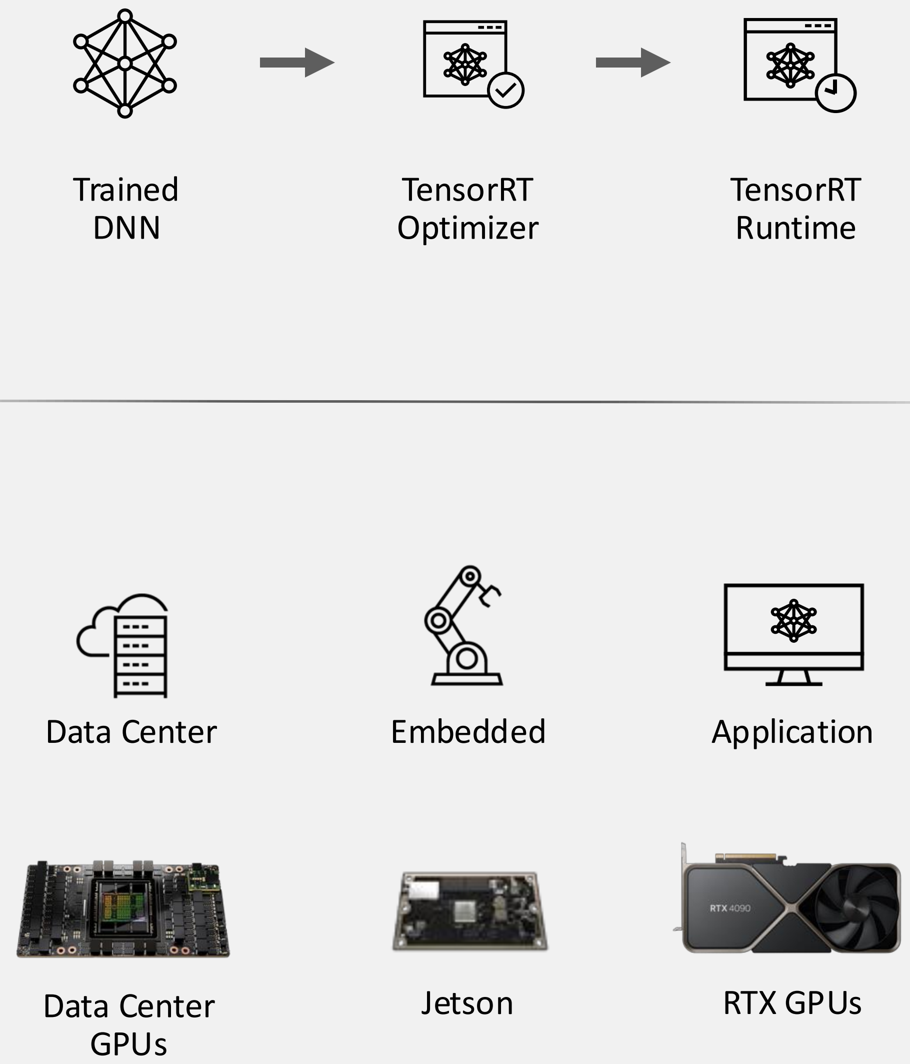
SDK for High-Performance Deep Learning Inference

Optimize and deploy neural networks in production.

Maximize throughput for latency-critical apps with compiler and runtime.

Optimize every network, including CNNs, RNNs, and Transformers.

1. Reduced mixed precision: FP32, TF32, FP16, and INT8
2. Layer and tensor fusion: Optimizes use of GPU memory bandwidth
3. Kernel auto-tuning: Select best algorithm on target GPU
4. Dynamic tensor memory: Deploy memory-efficient apps
5. Multi-stream execution: Scalable design to process multiple streams.
6. Time fusion: Optimizes RNN over time steps



https://developer.nvidia.com/tensorrt

TensorRT-LLM in the *DL Compiler* Ecosystem

TensorRT-LLM builds on TensorRT Compilation

TensorRT-LLM

LLM specific optimizations:

- KV Caching
- Multi-GPU, Muti-Node
- Custom MHA optimizations
- Paged KV Cache (Attention)
- *etc…*

TensorRT

General Purpose Compiler

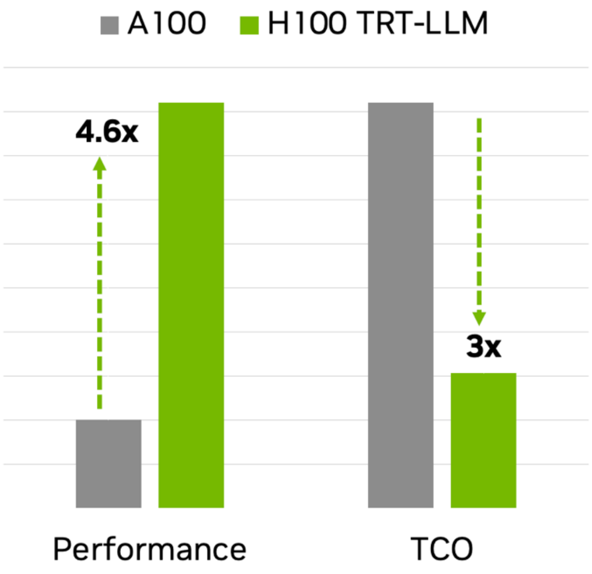
- Optimized GEMMs & general kernels
- Kernel Fusion
- Auto Tuning
- Memory Optimizations
- Multi-stream execution

TensorRT-LLM Optimizing LLM Inference

SoTA Performance for Large Language Models for Production Deployments

SoTA Performance

Leverage TensorRT compilation & kernels from FasterTransformers, CUTLASS, OAI Triton, ++



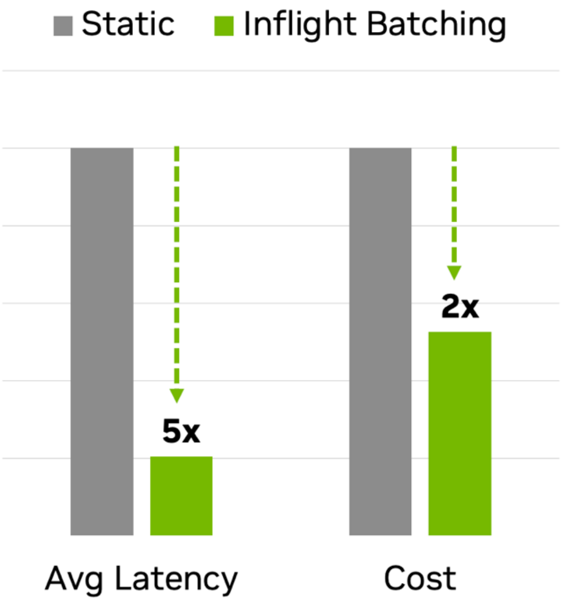
Ease Extension

Add new operators or models in Python to quickly support new LLMs with optimized performance

```
# define a new activation
def silu(input: Tensor) → Tensor:
    return input * sigmoid(input)

#implement models like in DL FWs
class LlamaModel(Module)
  def __init__(…)
    self.layers = ModuleList([…])

  def forward (…)
    hidden = self.embedding(…)

    for layer in self.layers:
      hidden_states = layer(hidden)

    return hidden
```

LLM Batching with Triton

Maximize throughput and GPU utilization through new scheduling techniques for LLMs



TensorRT and TensorRT-LLM model compression

Efficient inference

Why is it challenging?

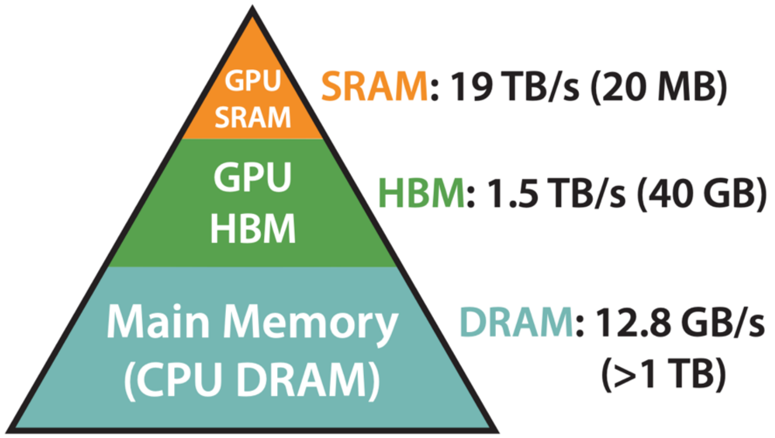
Memory

Operations

Memory for Inference

Even small LLMs are large

- Each billion parameters is ~2GB of memory
- Llama 8B is ~16GB of memory + the KV cache
- A H100 has 80GB of memory and finite bandwidth
- How can we make the most out of this memory?



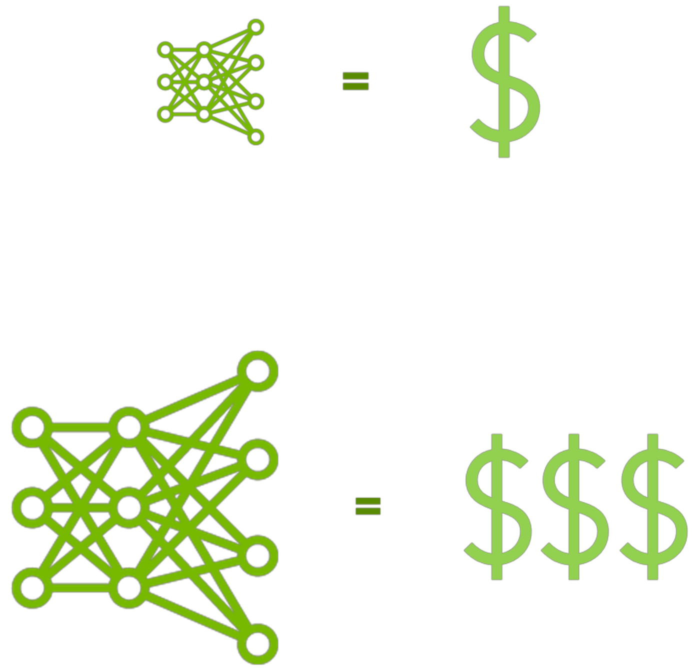
Memory Hierarchy with Bandwidth & Memory Size

Image from book "FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness"
"

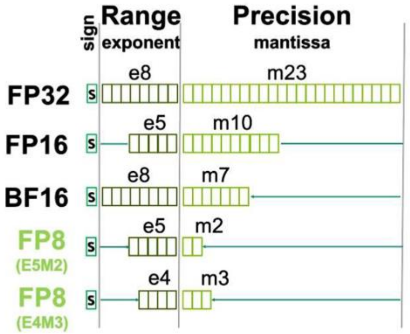
Operations for Inference

Billiions of operations increase the cost

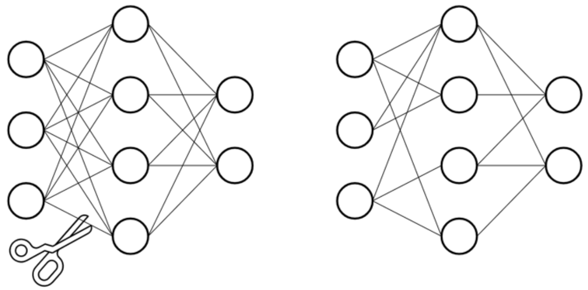
- Larger models perform better, but are costly
- Smaller LLMs can be a good tradeoff between cost and quality
- More efficient models drive the cost of inference down
- Can we make the inference computations cheaper?



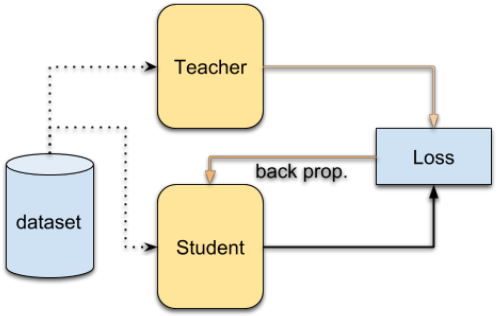
Model Compression Strategies



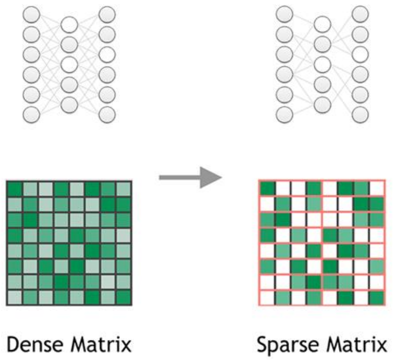
Quantization



Pruning



Distillation



Sparsity

NVIDIA

Quantization

Supported Precisions & Models

- Utilizes Hopper FP8 “Transfomer Engine”
- Support many 8bit & 4bit methods
  - FP8, INT8/INT4 Weight only, INT8 Smooth Quant, AWQ, GPTQ
  - Support varies by model
- Reduced model size, memory bandwidth, & compute
  - Improves performance & allows for larger models per GPU
- Model optimization toolkit to quantize pre-trained models
  - Allows for per layer quantization strategies
- Currently requires all weights to be in same precision
  - Would like to relax this constraint going forward
- Precision documentation

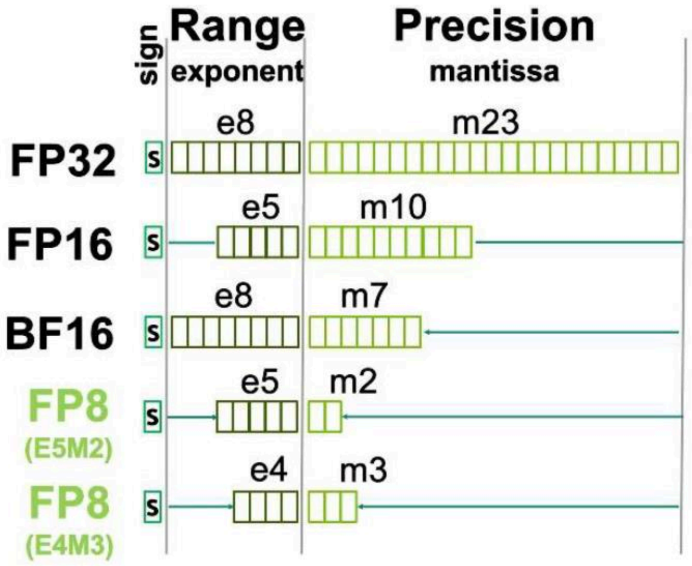
	FP32	FP16	BF16	FP8	INT8	INT4
Volta (SM70)	Y	Y	N	N	Y	Y
Turing (SM75)	Y	Y	N	N	Y	Y
Ampere (SM80, SM86)	Y	Y	Y	N	Y	Y
Ada-Lovelace (SM89)	Y	Y	Y	Y	Y	Y
Hopper (SM90)	Y	Y	Y	Y	Y	Y

Model	FP32	FP16	BF16	FP8	W8A8 SQ	W8A16	W4A16	W4A16 AWQ	W4A16 GPTQ
Baichuan	Y	Y	Y	Y	Y	Y	Y	Y	Y
BERT	Y	Y	Y	.	.	.	.	.	.
BLIP-2	Y	Y	Y	.	.	.	.	.	.
BLOOM	Y	Y	Y	.	Y	Y	Y	.	.
ChatGLM	Y	Y	Y	.	.	.	.	.	.
ChatGLM-v2	Y	Y	Y	.	.	.	.	.	.
ChatGLM-v3	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	.
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Flan-T5	Y	Y	[illegible]	.	.	.	.	.	.
GPT	Y	Y	Y	Y	Y	Y	Y	.	.
GPT-J	Y	Y	Y	Y	.	Y	Y	Y	.
GPT-NeMo	Y	Y	Y	.	.	.	.	.	.
GPT-NeoX	Y	Y	Y	.	.	.	.	.	Y
InternLM	Y	Y	Y	.	Y	Y	Y	.	.
LLaMA	Y	Y	Y	Y	Y	Y	Y	Y	Y
LLaMA-v2	Y	Y	Y	Y	Y	Y	Y	Y	Y
Mistral	Y	Y	Y	Y	Y	Y	Y	Y	.
MPT	Y	Y	Y	Y	Y	Y	Y	Y	.
OPT	Y	Y	Y	.	.	.	.	.	.
Phi	Y	Y	Y	.	.	.	.	.	.
Replit Code	Y	Y	Y	.	Y	Y	Y	.	.
SantaCoder	Y	Y	Y	.	.	Y	Y	.	.
StarCoder	Y	Y	Y	.	.	Y	Y	.	.
T5	Y	Y	Y	.	.	.	.	.	.

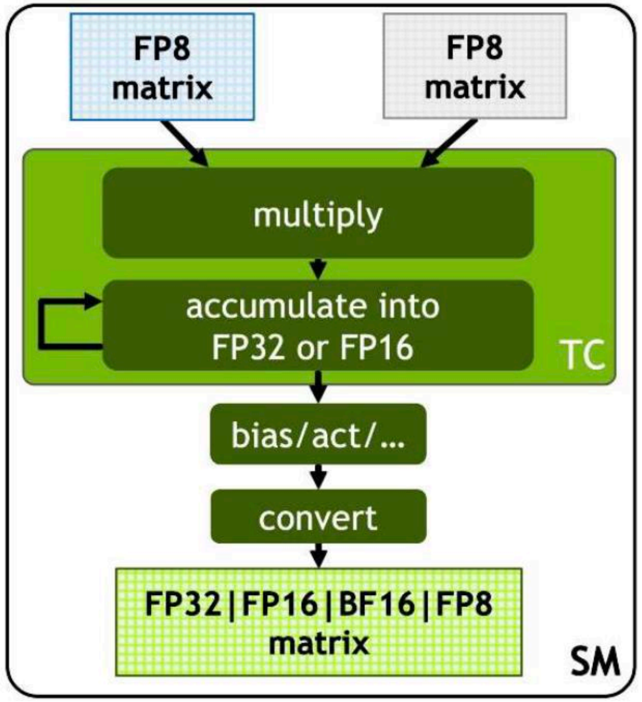
Quantization Examples

Supported Models

Quantization of FP Formats



Allocate 1 bit to either range or precision

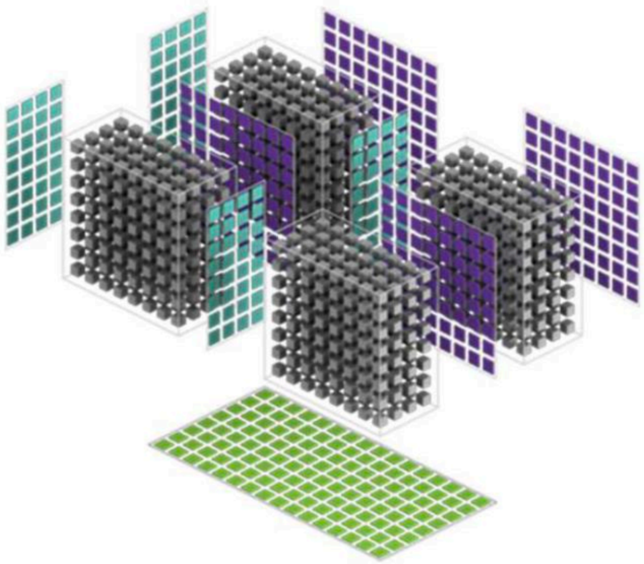


Support for multiple accumulator and output types

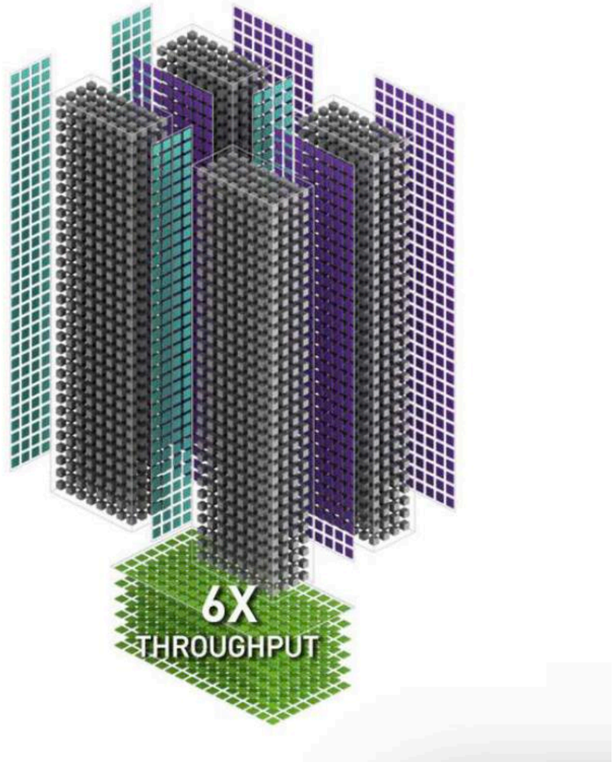
Images from https://resources.nvidia.com/en-us-tensor-core

Comparison of Throughput Across FP Formats

A100 FP16



H100 FP8



Images from https://resources.nvidia.com/en-us-tensor-core

Quantization

How to Chose a Precision

- Best precision varies by application
  - FP8 activations generally provides best performacne
- Weight quantization reduces memory footprint & traffic
  - Reduces latency
  - Can fit larger models
  - Costs compute time to unpack the weights
- Activation quantization saves on compute
  - Improves throughput
  - Can run larger batch sizes
- *WXAY* = weights quantized to *X* bits, and activations to *Y*
- Quantization Guide

Method	Performance Improvement small batch BS <=4	Performance Improvement large batch BS>=16	Accuracy impact	Calibration time
FP8 (W8A8)	Medium	Medium	**Very low / None**	**O(1min)**
INT8 SQ (W8A8)	Medium	Medium	Medium	O(1min)
INT8 WO (W8A16)	Medium	***None***	**Low**	***None***
INT4 WO (W4A16)	**High**	***None***	**High**	***None***
INT4 AWQ (W4A16)	**High**	***None***	**Low**	O(10min)
INT4 GPTQ (W4A16)	**High**	***None***	**Low**	O(10min)
INT4-FP8 AWQ (W4A8)	**High**	Medium	**Low**	O(10min)

SQ = Smooth Quant
WO = Weight Only
AWQ = Activation Aware Quantization

NVIDIA Triton Inference Server

Triton Inference Server

Open-Source Software For Fast, Scalable, Simplified Inference Serving

Any Framework

Supports Multiple Framework Backends Natively e.g., TensorFlow, PyTorch, TensorRT, XGBoost, ONNX, Python & More

Any Query Type

Optimized for Real Time, Batch, Streaming, Ensemble Inferencing

Any Platform

X86 CPU | Arm CPU | NVIDIA GPUs | MIG

Linux | Windows | Virtualization

Public Cloud, Data Center and Edge/Embedded (Jetson)

DevOps & MLOps

Integration With Kubernetes, KServe, Prometheus & Grafana

Available Across All Major Cloud AI Platforms

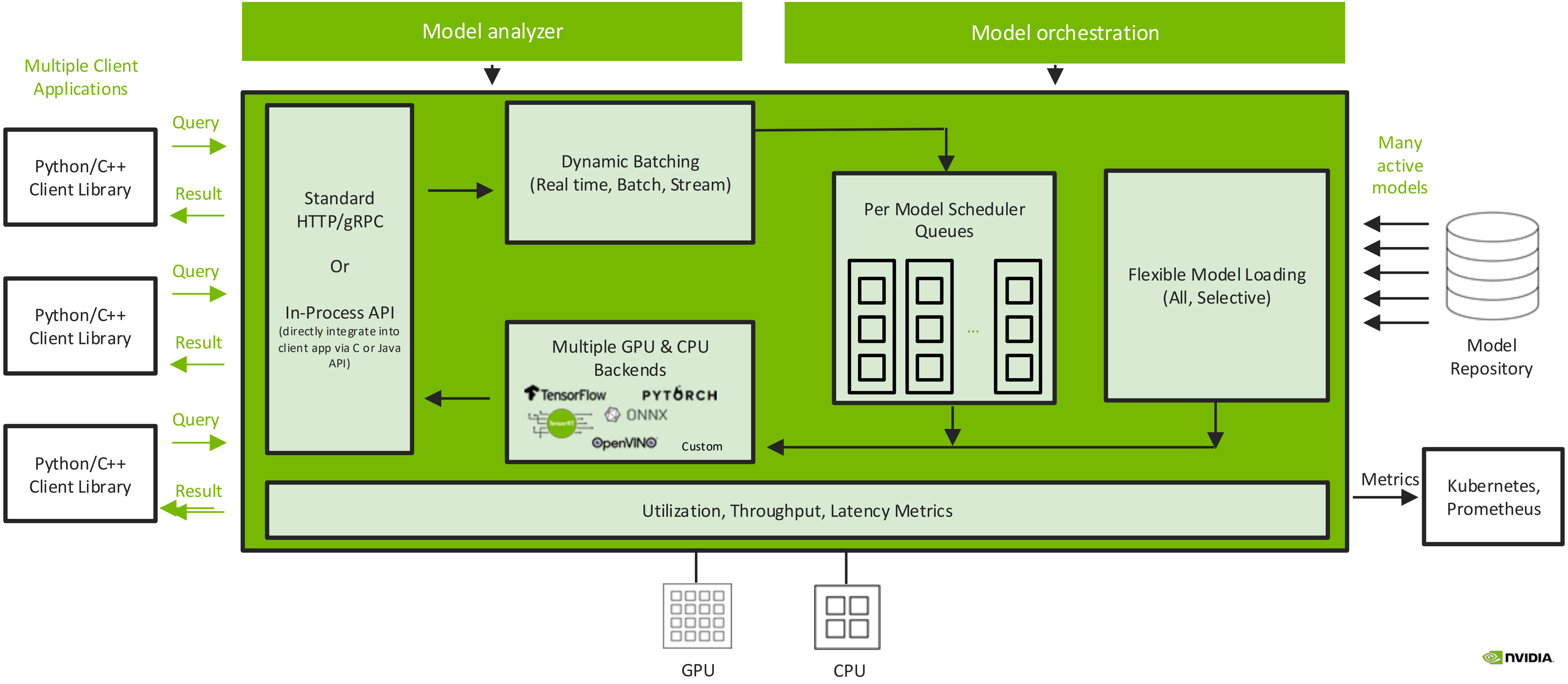
Performance & Utilization

Model Analyzer for Optimal Configuration

Optimized for High GPU/CPU Utilization, High Throughput & Low Latency

Delivering High Performance Across Frameworks

Triton's architecture



Supports Multiple Model Execution Backends

TensorFlow 1.x/2.x
Any Model
SavedModel | GraphDef

PyTorch
Any model
JIT/Torchscript | Python

TensorRT
All TensorRT optimized models

TF-TensorRT & TorchTRT
Any TensorFlow and PyTorch model

FIL (RAPIDS)
Tree based models
(e.g., XgBoost, Scikit-learn RandomForest, LightGBM)

ONNX RT
ONNX format

Python
Custom code in Python e.g., pre/post processing, any Python model

Custom C++ Backend
Custom framework in C++

DALI
Preprocessing logic using DALI operators

OpenVINO
OpenVINO optimized models on Intel architecture

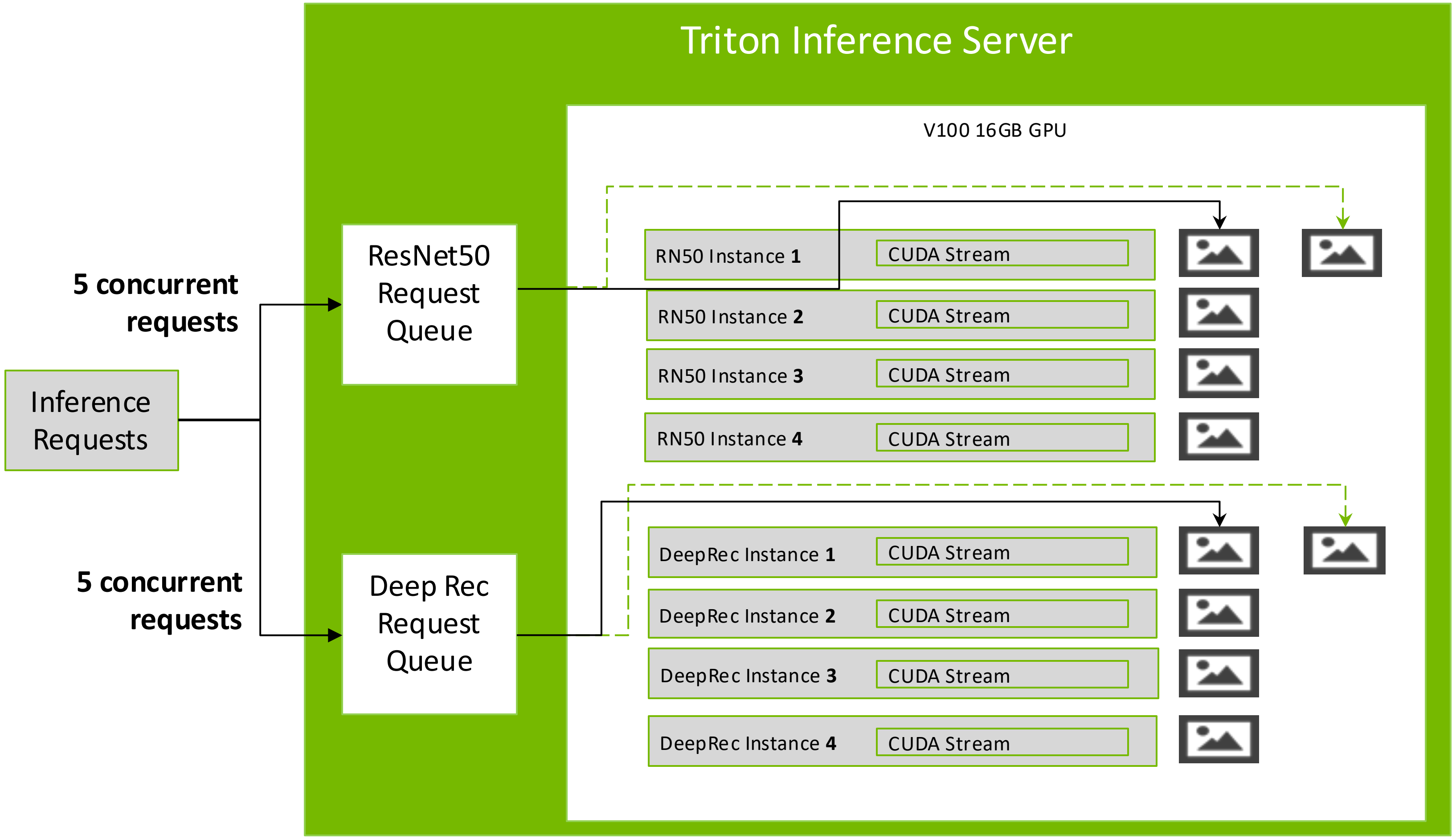
Faster Transformer
Multi-GPU, multi-node inferencing for large transformer models (GPT and T5)

NVTabular
Feature engineering and preprocessing library for tabular data

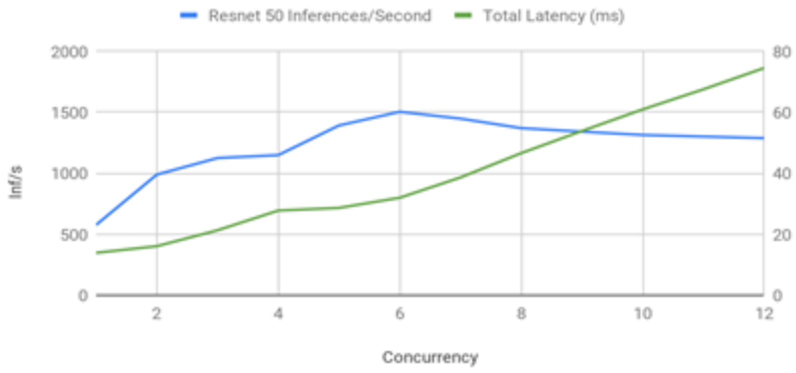
HugeCTR
Recommender model with large embeddings

Paddle Paddle
Paddle paddle models

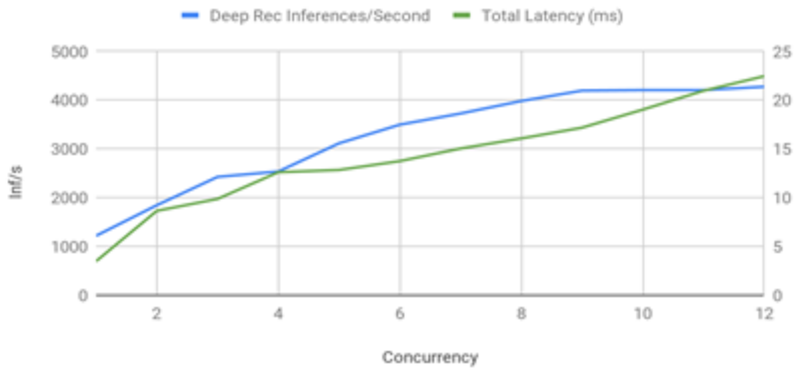
Concurrent Model Execution



TRT FP16 Resnet 50 Inferences/Second vs Total Latency BS8 Instance 4 on T4



TRT FP16 Deep Rec Inferences/Second vs Total Latency BS8 Instance 4 on T4

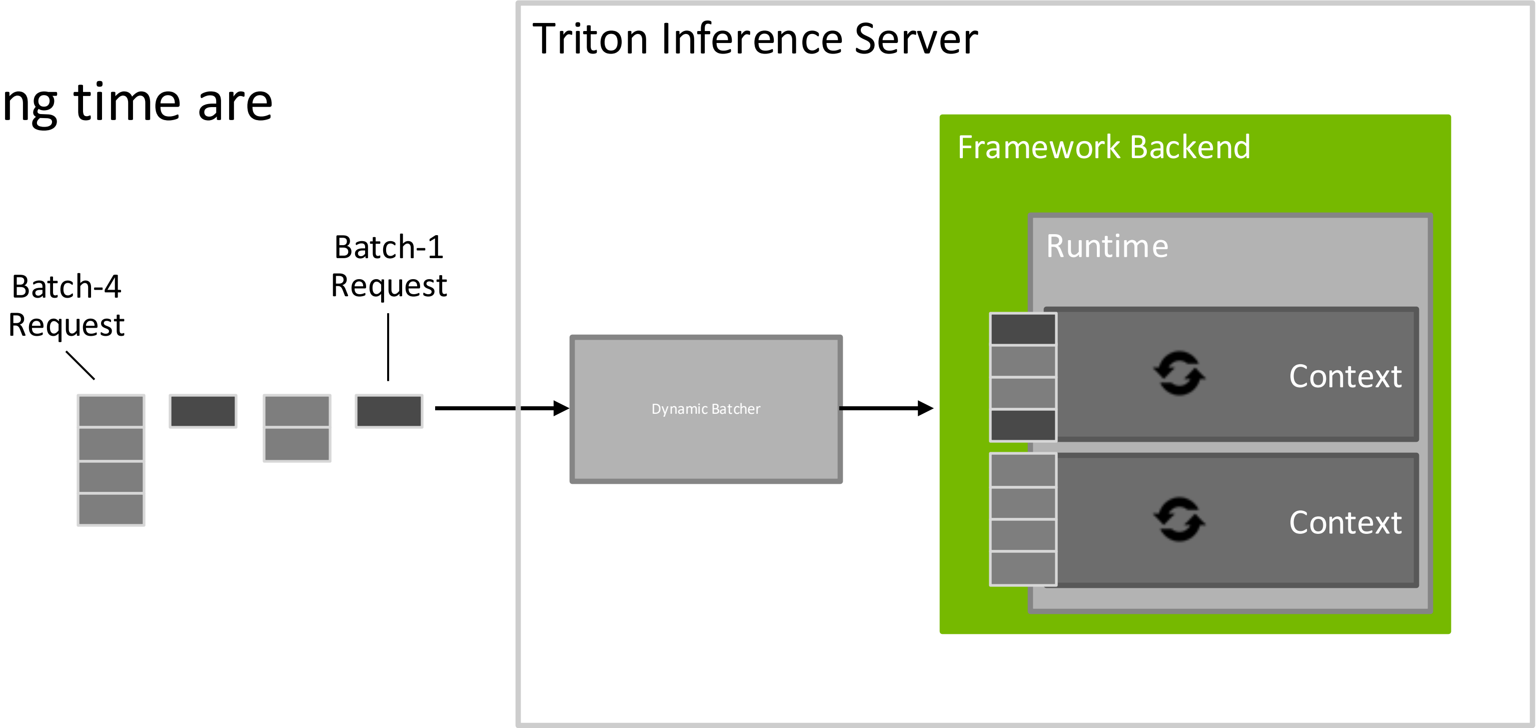


NVIDIA

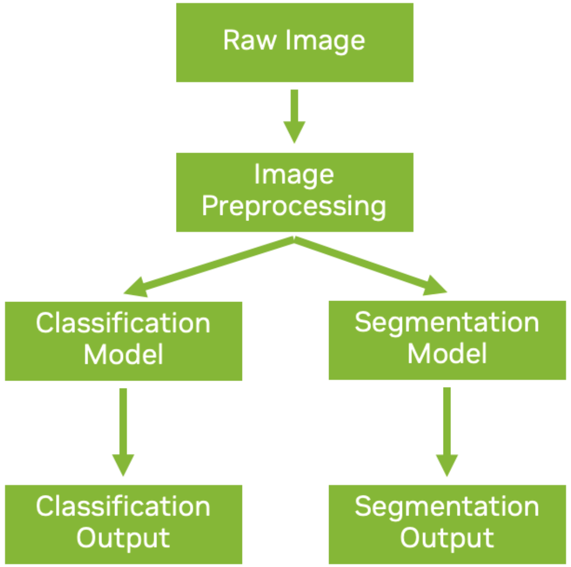
Dynamic Batching

Group requests to form larger batches, increase GPU utilization

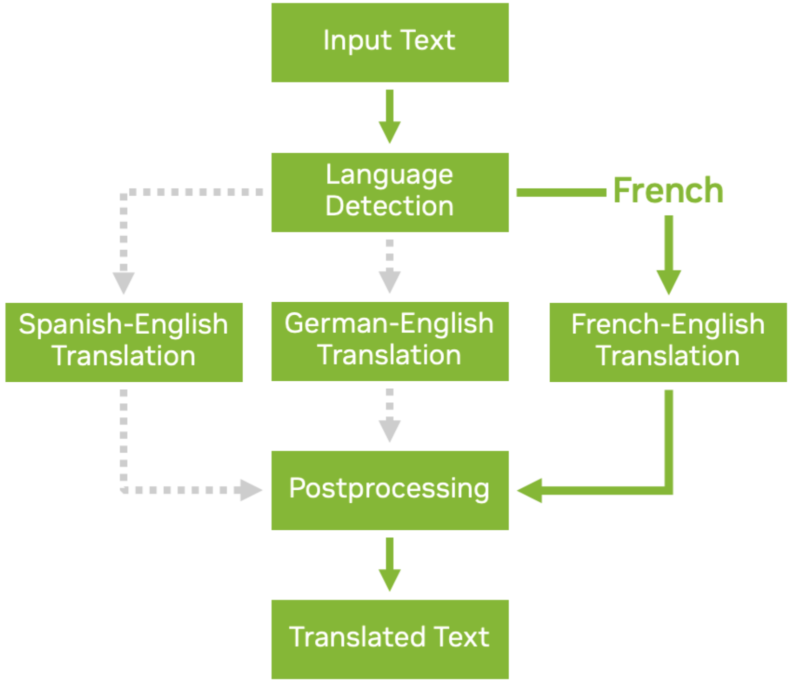
- Client sends independent requests
- Triton groups requests into a single batch to increase overall throughput
- Preferred batch size and waiting time are configuration options



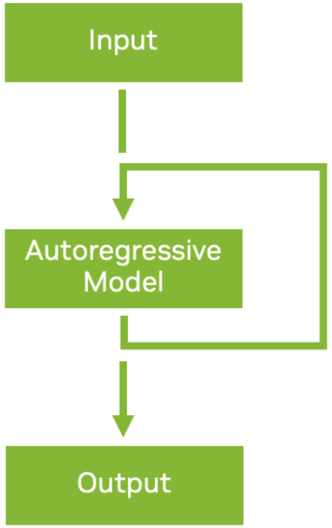
Model Pipelines: Ensembles & Business Logic Scripting



Model Ensemble



Conditional Execution



Looping execution

- ✓ **Models from any framework**
- ✓ **GPU shared memory for optimal performance**
- ✓ **Run on GPU or CPU**

NIM: fastest path to AI inference

NVIDIA NIM is the Fastest Path to AI Inference

Reduces engineering resources required to deploy optimized, accelerated models

	NVIDIA NIM	Triton + TRT-LLM Opensource
Deployment Time	5 minutes	~1 week
API Standardization	Industry standard protocol OpenAI for LLMs, Google Translate Speech	User creates a shim layer (reducing performance) or modify Triton to generate custom endpoints
Pre-Built Engine	Pre-built TRT-LLM engines for NV and community models MISTRAL AI_ Llama 2 starcoder NVIDIA Nemotron	User converts checkpoint to TRT-LLM format and creates and runs sweeps through different parameters to find the optimal config
Triton Ensemble/ BLS Backend	Pre-built with TRT-LLM to handle pre/post processing (tokenization)	User manually sets up + configures
Triton Deployment	Automated	User manually sets up + configures
Customization	Supported – P-tuning and LORA, more planned	User needs to create custom logic
Container Validation	Pre-validated with QA testing	No pre-validation
Support	NVIDIA AI Enterprise - Security and CVE scanning/patching and tech support	No enterprise support

NVIDIA NIM Optimized Inference Microservices

Accelerated runtime for generative AI

NVIDIA NIM

Prebuilt container and Helm chart

Industry standard APIs

Support for custom models

Domain specific code

Optimized inference engines

Deploy anywhere and maintain control of generative AI applications and data

Simplified development of AI application that can run in enterprise environments

Day 0 support for all generative AI models providing choice across the ecosystem

Improved TCO with best latency and throughput running on accelerated infrastructure

Best accuracy for enterprise by enabling tuning with proprietary data sources

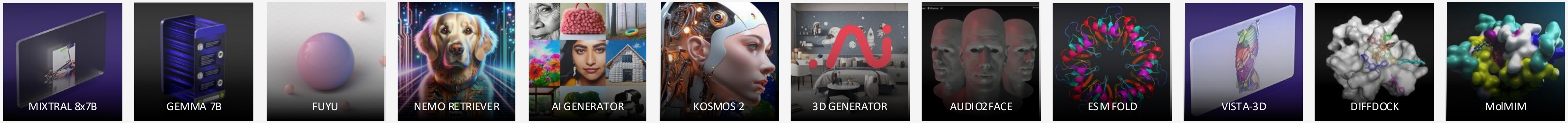
Enterprise software with feature branches, validation and support

Microsoft Azure | aws | Google Cloud | ORACLE | DGX & DGX Cloud | DELL Technologies | Hewlett Packard Enterprise | Lenovo | SUPERMICRO

Inference Microservices for Generative AI

NVIDIA NIM is the fastest way to deploy AI models on accelerated infrastructure across cloud, data center, and PC

NVIDIA API Catalog



MISTRAL AI_ | Google | ADEPT | NVIDIA | gettyimages | Microsoft | shutterstock | NVIDIA | Meta | NVIDIA | MIT | NVIDIA

Microsoft Azure | aws | Google Cloud | ORACLE | DELL Technologies | Hewlett Packard Enterprise | Lenovo | SUPERMICRO

NVIDIA NIM for LLM Architecture

- HTTP REST API conforms to OpenAI specification for easy developer integration
- Liveness, health check and metrics endpoints for monitoring and enterprise management
- NVIDIA NIM includes multiple LLM runtimes
  - TensorRT-LLM and vLLM
  - Runtime is selected based on detected hardware and available optimized engines, with preference given to optimized engines