LLM Inference Webinar

Sagar Desai, Dmitry Mironov

Welcome & Introduction

- Sagar Desai is a Senior Solution Architect at NVIDIA. He specializes in large-scale generative AI solutions, including LLMs, multimodal AI, and retrieval-augmented generation (RAG). He excels at designing and deploying scalable, reliable, and secure AI architectures that deliver business value.
- He’s adept at advanced LLM training techniques, such as FP8 precision, and optimizing model convergence and GPU utilization.
- Drawing on his background in MLOps, Sagar designs comprehensive deployment pipelines that ensure seamless, high-throughput inference with concurrent usability. Focused on accuracy, scalability, and reliability, he empowers organizations to unlock the full potential of generative AI technologies.

Agenda

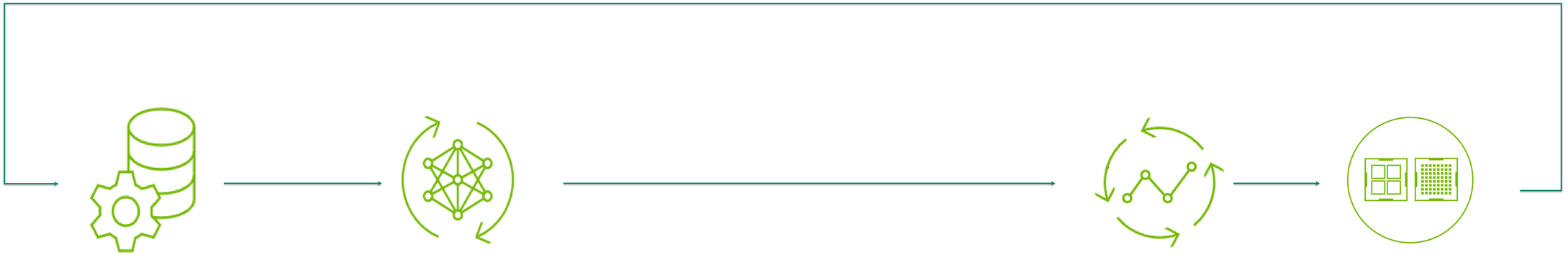
- Deep understanding of LLM inference theory and applications
- Efficient prompt processing and token generation techniques.
- NVIDIA Inference options
- NVIDIA NIM Microservices, TensorRT-LLM, Triton
- NVIDIA Dynamo
- Demo
- Case Studies

Challenges of LLM Inference in Production

Inferencing in the End-to-end AI Workstream

AI Training

AI Inference



DATA PROCESSING

Collect, cleanse and transform training data set

TRAIN

Select model and fine tune for desired accuracy

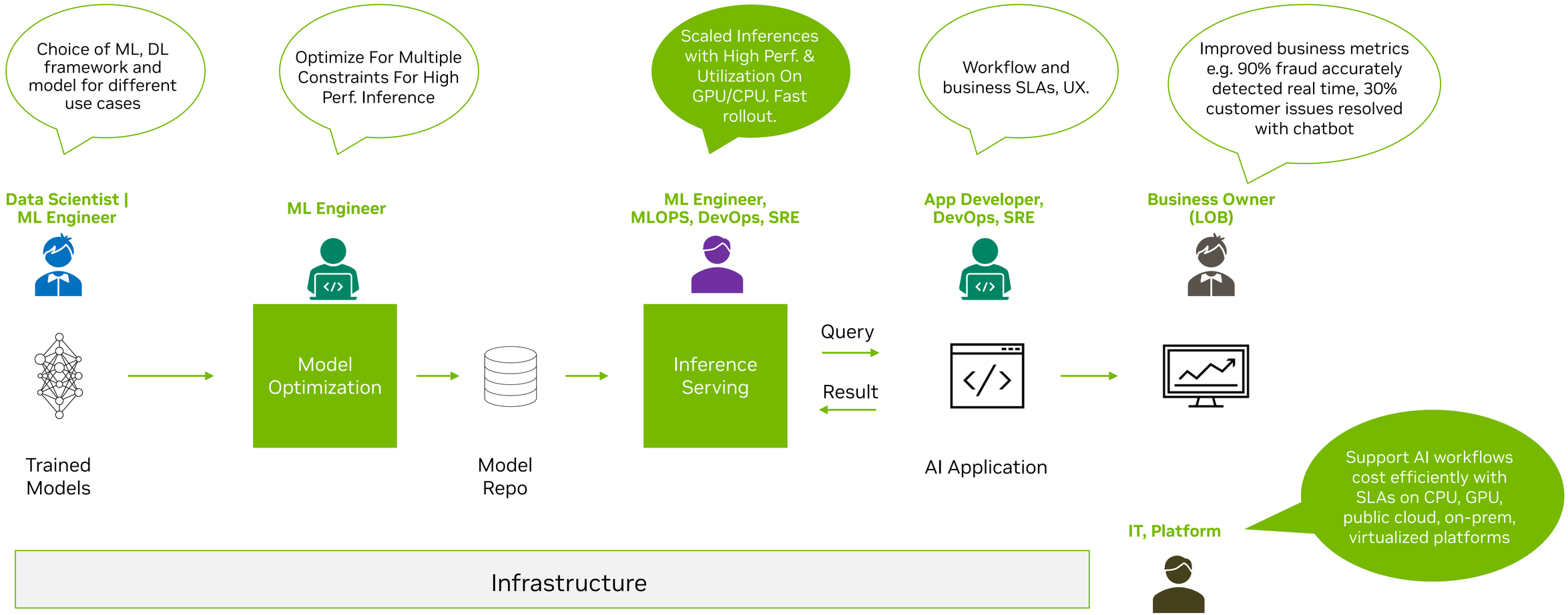
OPTIMIZE

Optimize model for target HW accelerator

DEPLOY

Serve model for optimal user experience & TCO

AI Inference Workflow



LLM Inference Fundamentals

Inference Latency (SLOs)

Tokens

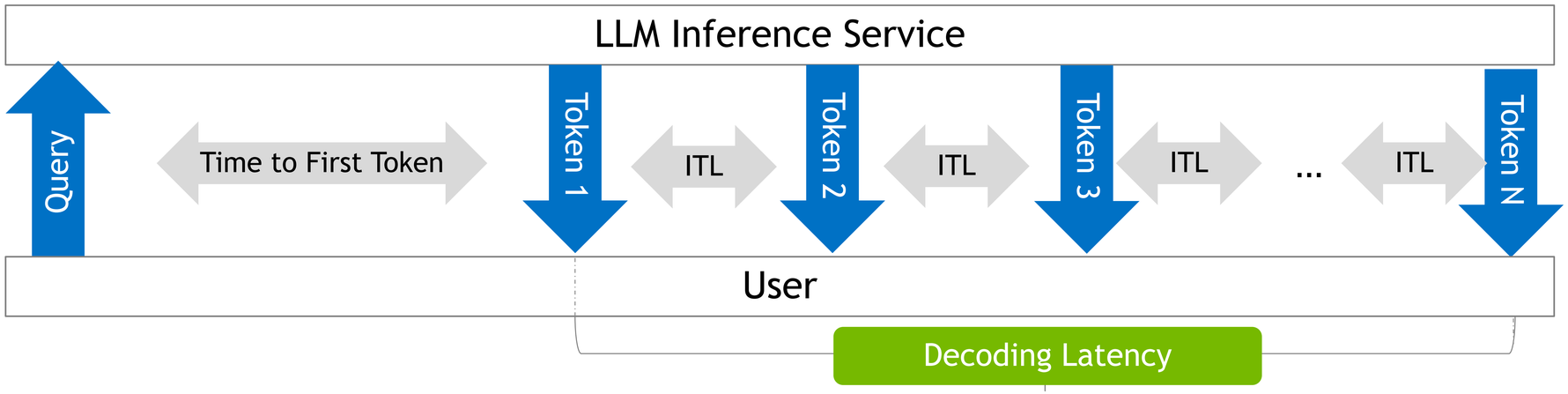
- Input sequence length of tokens (**ISL**) are fed into a model.
- LLM generates of output sequence length (**OSL**) of tokens one at a time

Time to First Token (TTFT)

- Time it takes for the model to generate the first token after receiving a request
- This part of execution is called **Prefill**
- Critical for chat-like applications

End-to-end Latency (E2E)

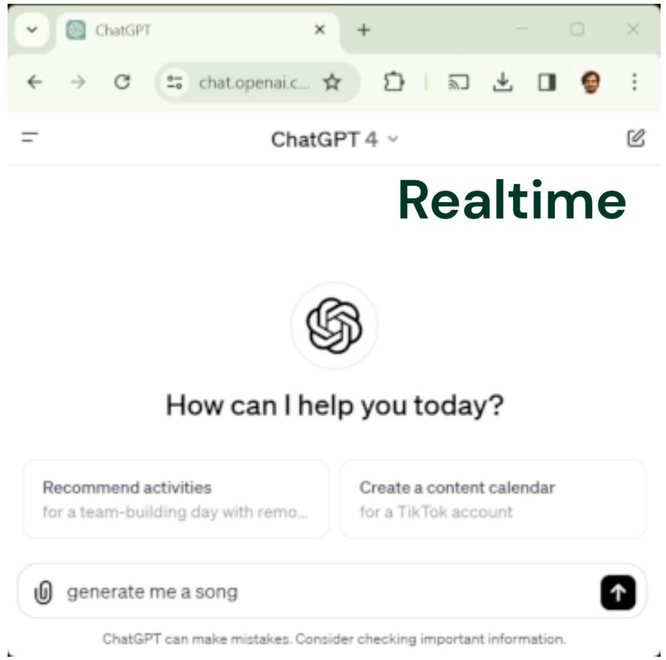
- Time it takes for a model to generate a complete response to a user's prompt.
- E2E Latency = TTFT + **Decoding** Latency
- Inter-token Latency (**ITL**) is an average time between output tokens
- Critical, when full response has to be processed further: guardrails, tool calling



Two Stages of LLM Execution

Prefill vs Decoding

- **Prefill** = time to first token (~word)
  - Loading the user prompt into the system
  - From the request reception to the first token
  - Depends only on the number of input tokens
  - Populate KV-cache for all the tokens from the prompt.
  - Compute-bound for most of the reasonable prompt lengths
- **Decoding** = inter-token latency
  - Generating the response token by token, word by word
  - Inter-token latency depends on the total token number, both input and output tokens.
  - Usually memory-bound



Prefill: 1.14s, 5 input tokens, 1 output token

`generate me a song`

Decoding: 1.62s, 33 output tokens

`Could you please provide me with some specifics`

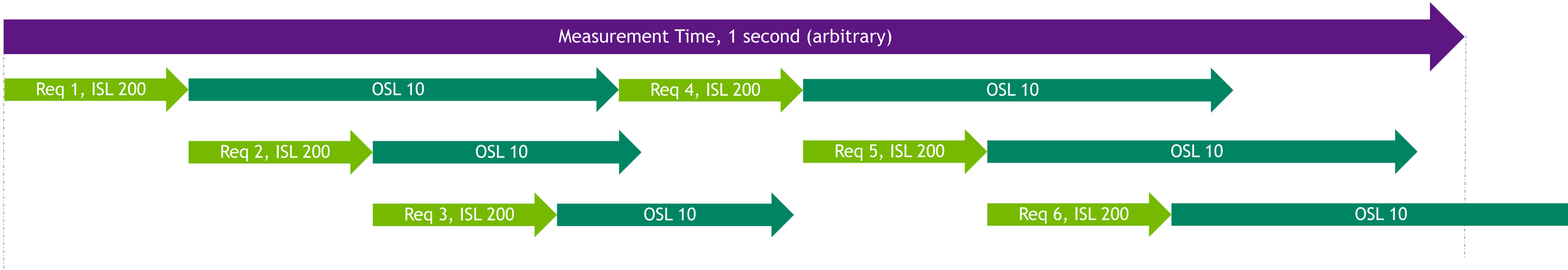
Prefill, first-response	Decoding							
User: generate me a song. AI: *Could*	you	please	me	with	some	specifics	for	your

prompt

user prompt

Inference Throughput Example (Costs)

Assuming 1 GPUs per model, 1 instance



Legend:

- Requests per second per deployments (**RPS**) = 5 started and completed requests / 1 second = 5 (200 tokens in, 10 tokens out)
- Output tokens per second per deployments (**TPS**) = 57 output tokens / 1 second = 57 (200 tokens in, 10 tokens out)
- **Requests per second per GPU** = RPS / #GPU = 5 / 1 = 5 (200 tokens in, 10 tokens out)
  - If 2X RPS needed keeping latency the same, need 2X servers.
  - **Best definition of throughput. Direct relationship to costs. Always try to calculate it.**

NVIDIA AI Inference Solutions

Challenges with AI inference in Production

IT leaders chose AI inference solutions that meet enterprise grade requirements

Fast

Meets use case and user experience latency constraints

Interoperable

Supports multiple development frameworks Adapt to SOTA architectures, e.g., reasoning agents

Scalable

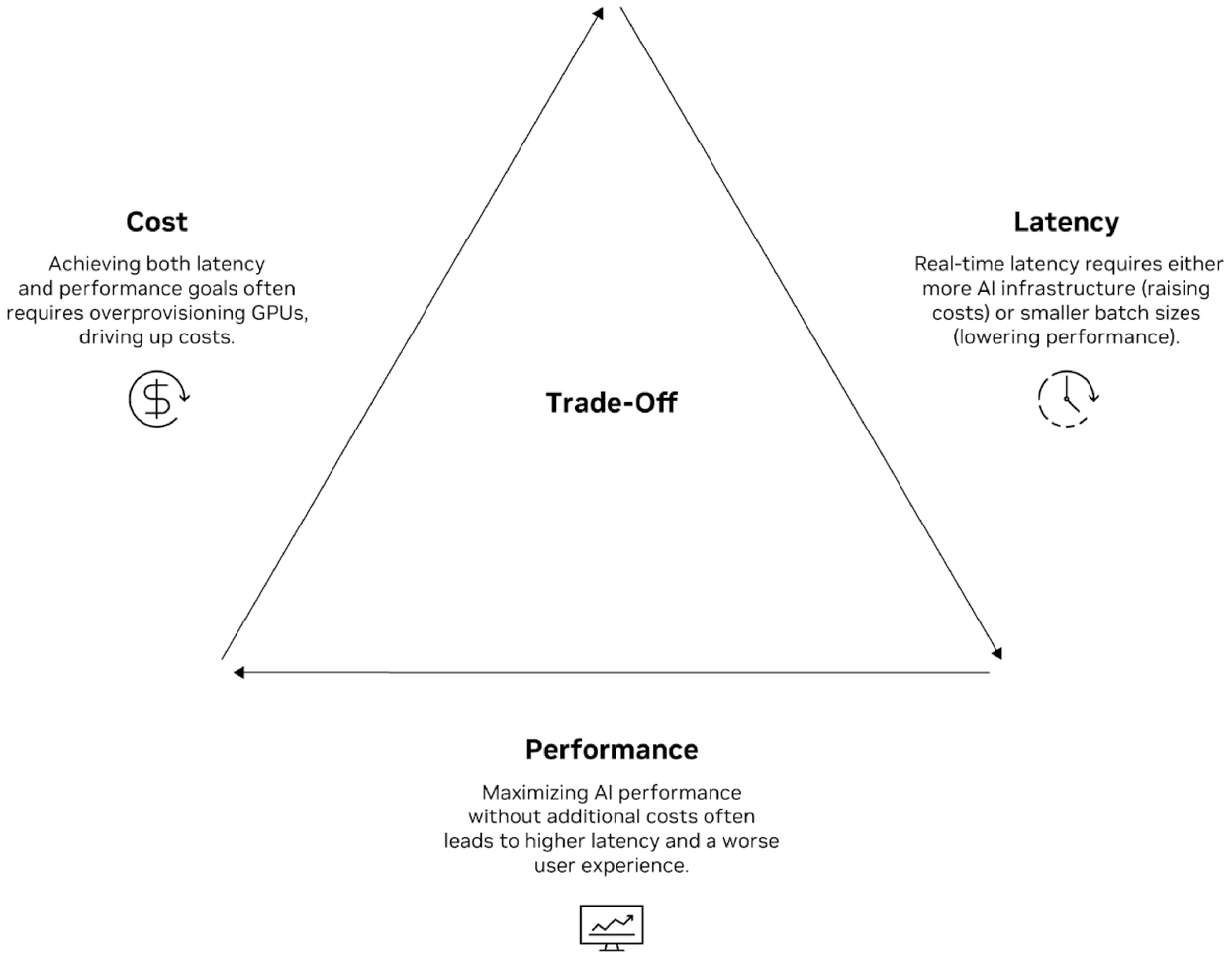
Easily add new models and support spikes in user demand

Cost

Meets budget constraints and delivers max ROI

Deploying LLM Inference: Throughput vs Latency

Balance between throughput and latency is crucial for optimal performance and user experience



Higher Throughput

- ✓ Higher resource utilization
  Lower deployment costs
- ⊘ Higher Latency
  Slower UX

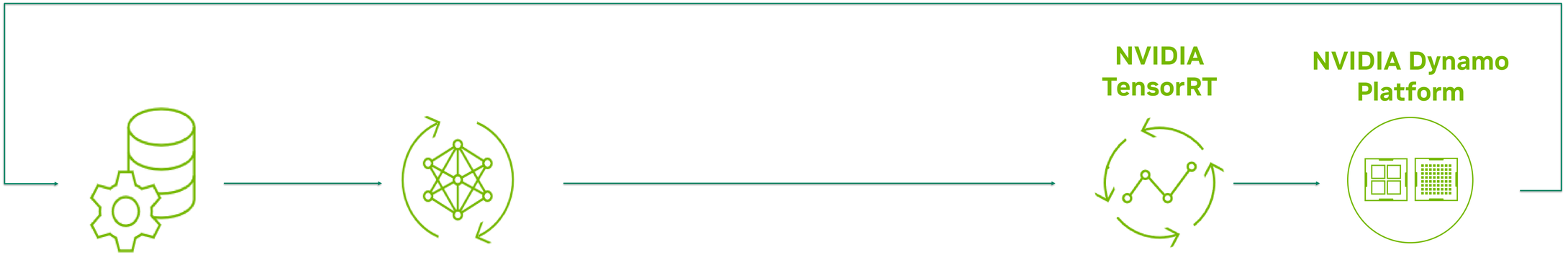
Lower Latency

- ✓ Better UX
- ⊘ Lower throughput
  Lower GPU utilization
  Higher deployment costs

Inferencing in the End-to-end AI Workstream

AI Training

AI Inference



DATA PROCESSING

Collect, cleanse and transform training data set

TRAIN

Select model and fine tune for desired accuracy

OPTIMIZE

Optimize model for target HW accelerator

DEPLOY

Serve model for optimal user experience & TCO

TensorRT-LLM

TensorRT-LLM in the *DL Compiler* Ecosystem

TensorRT-LLM builds on TensorRT Compilation

- **TensorRT-LLM**
  - Inference runtime & compiler specifically designed for LLMs
  - LLM specific optimizations:
    - KV Caching & Custom MHA Kernels
    - Inflight batching, Paged KV Cache (Attention)
    - Multi-GPU, Multi-Node
    - Grammar support
    - *& more*
  - *ONLY for LLMs*

- **TensorRT**
  - General purpose Deep Learning Inference Compiler
    - Graph rewriting, constant folding, kernel fusion
    - Optimized GEMMs & pointwise kernels
    - Kernel Auto-Tuning
    - Memory Optimizations
    - *& more*
  - *All AI Workloads*

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 GPU Support

GB200 NVL72

Blackwell

Hopper

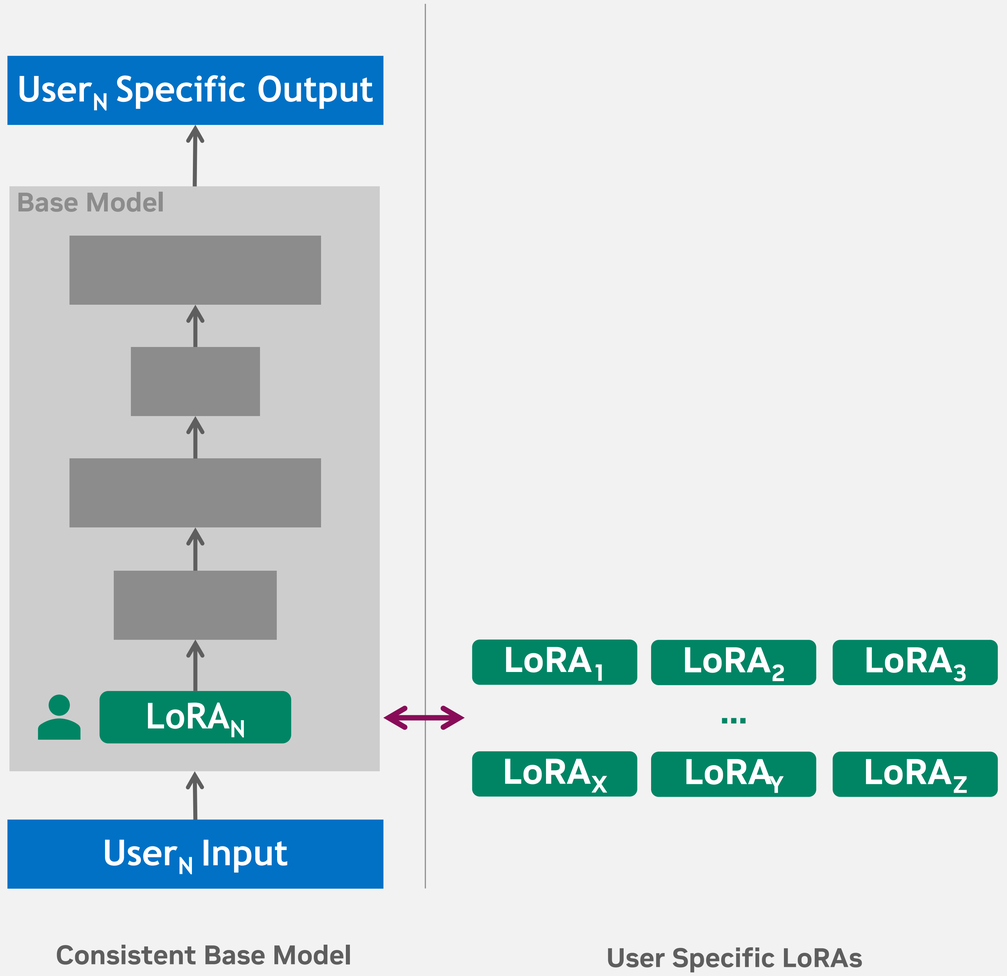
Ada Lovelace

Ampere

LoRA & Customization

Efficiently Supporting Customer User Experience

- LoRA & Prompt tuned models are support in TRT-LLM and NVIDIA NIM
- Support mulitple customers with a single model
- Dynamically swap LoRA's at runtime
- SLoRA / LoRAx caching adapters on device
- Base model can be quantized for memory savings
  - QLoRA in progress

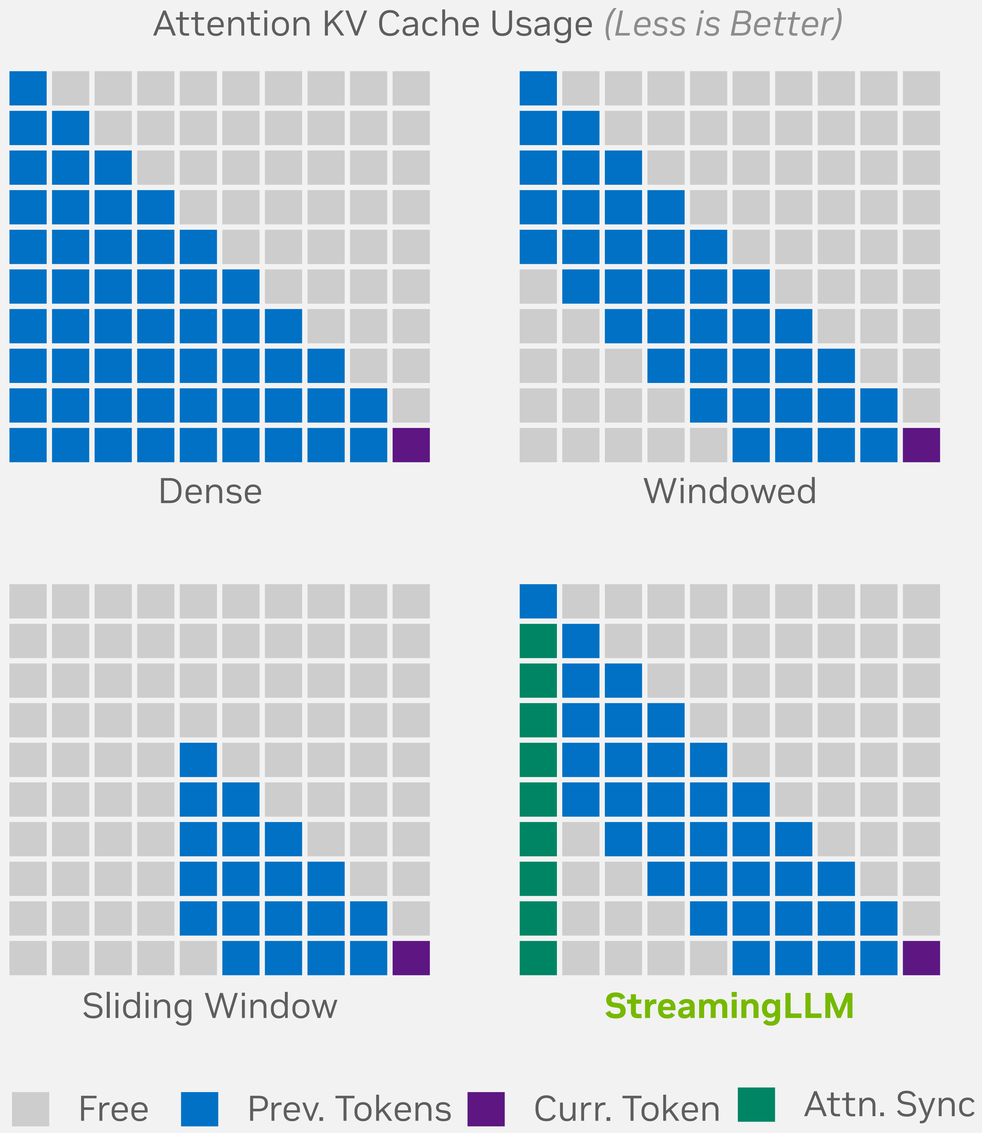


Dynamically Swap LoRAs based on User

KV Cache & Attention Techniques

(Sliding) Window Attention, & Streaming LLM

- Allow for longer (sometimes unlimited) sequence length
  - Reduces KV Cache Memory usage
  - Avoids OOM Errors
- (Sliding) Windowed Attention evict tokens based on arrival
  - Significantly reduces memory usage
  - Can negatively impact accuracy or require recomputing KV
- Streaming-LLM allows for unlimited sequence length
  - Does not evict Attention Sinks (important elements)
  - KV Cache stays constant size
  - Does not require recompute & does not impact accuracy
  - Particulary beneficial for multi-turn (ie. chat) usecases

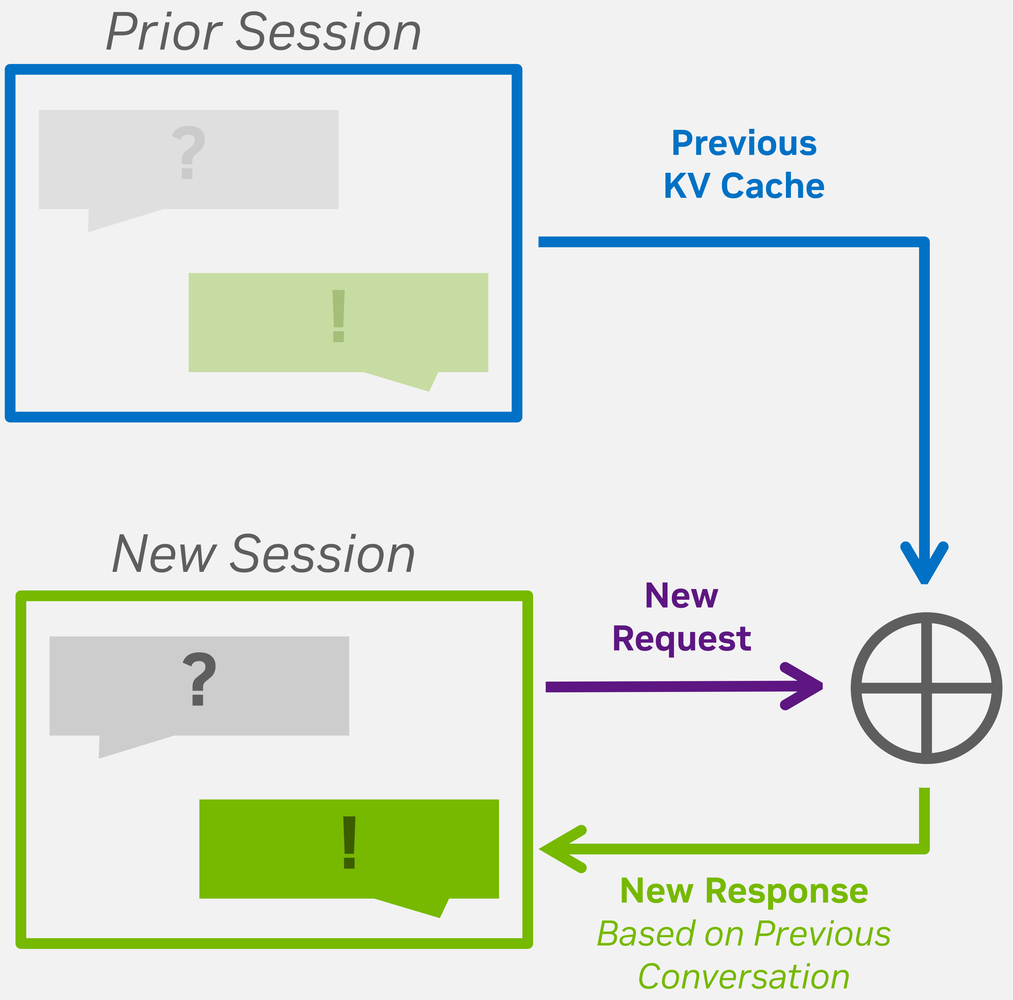


KV Cache Reusage

System Prompt Caching & Block reusage

Allows for interactive/ turn based systems & System Prompts

- Load prior KV cache blocks to avoid recomputation
  - Saves significant compute
  - Reduces Start-up time
- Block reusage allows for turn-based (chat) applications
  - Allows for additional options for intelligently reusing blocks
- System prompts allows for a preset KV cache for the LLM
  - E.g. to give rules, personality, or prior knowledge

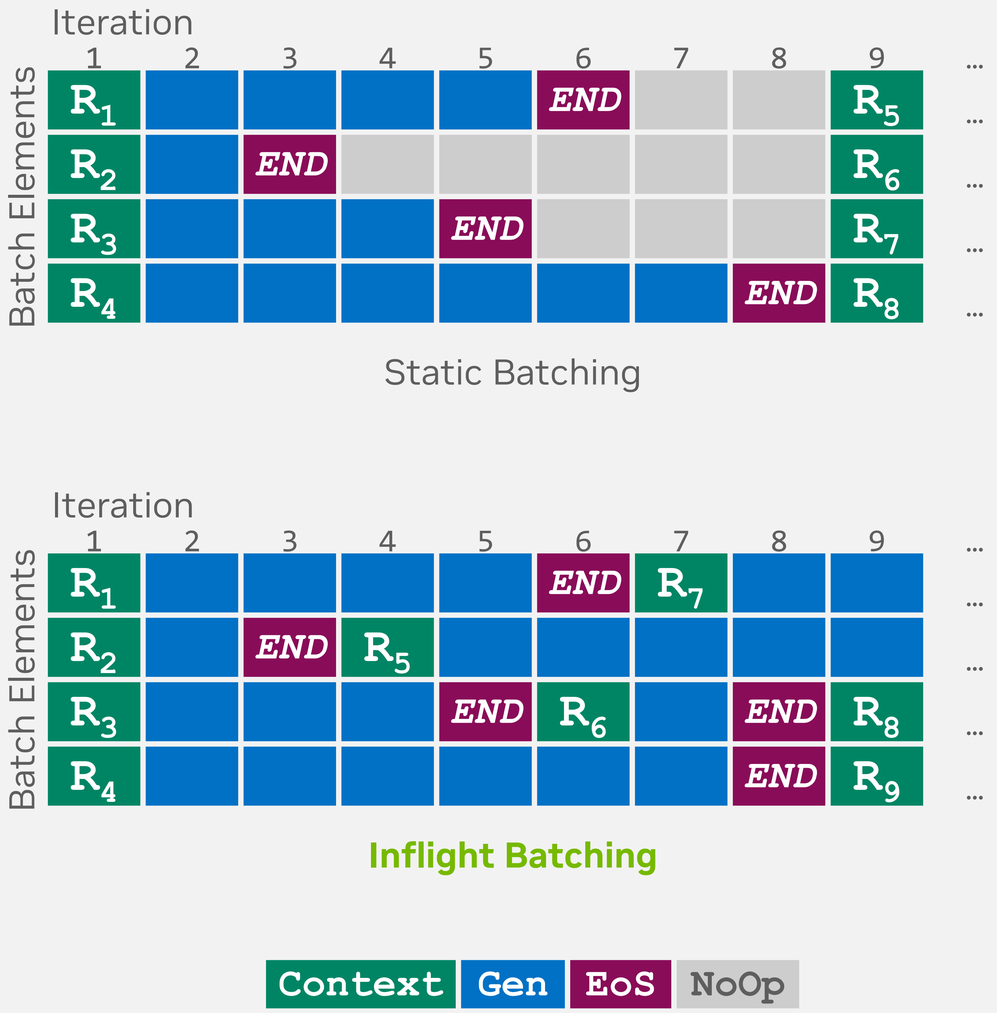


Inflight Batching

Maximing GPU Utilization during LLM Serving

TensorRT-LLM provides custom Inflight Batching to optimize GPU utilization during LLM Serving

- Replaces completed requests in the batch
  - Evicts requests after EoS & inserts a new request
- Improves throughput, time to first token, & GPU utilizaiton
- Integrated directly into the TensorRT-LLM Triton backend
- Accessible though the TensorRT-LLM Batch Manager



KV Cache Optimizations

Paged & Quantized KV Cache

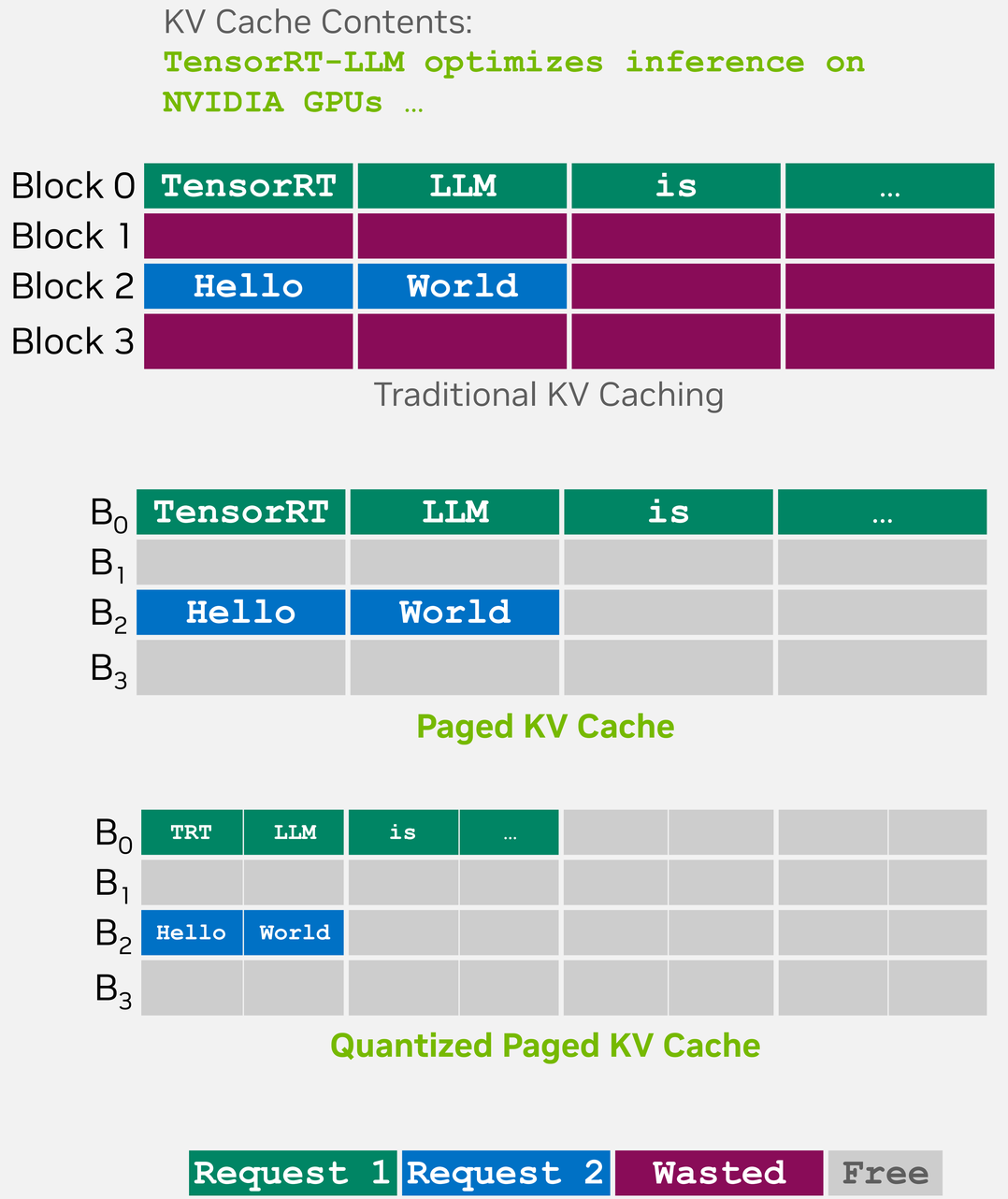
Paged KV Cache improves memory consumption & utilization

- Stores keys & values in non-contiguous memory space
- Allows for reduced memory consumption of KV cache
- Allocates memory on demand

Quantized KV Cache improves memory consumption & perf

- Reduces KV Cache elements from 16b to 8b (or less!)
- Reduces memory transfer improving performance
- Supports INT8 / FP8 KV Caches

Both allow for increased peak performance



Quantization

Supported Precisions & Models

- Utilizes Transfomer Engine on Hopper and Blackwell for FP8 and FP4

- Support many 8bit & 4bit methods
  - FP8, FP4, 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

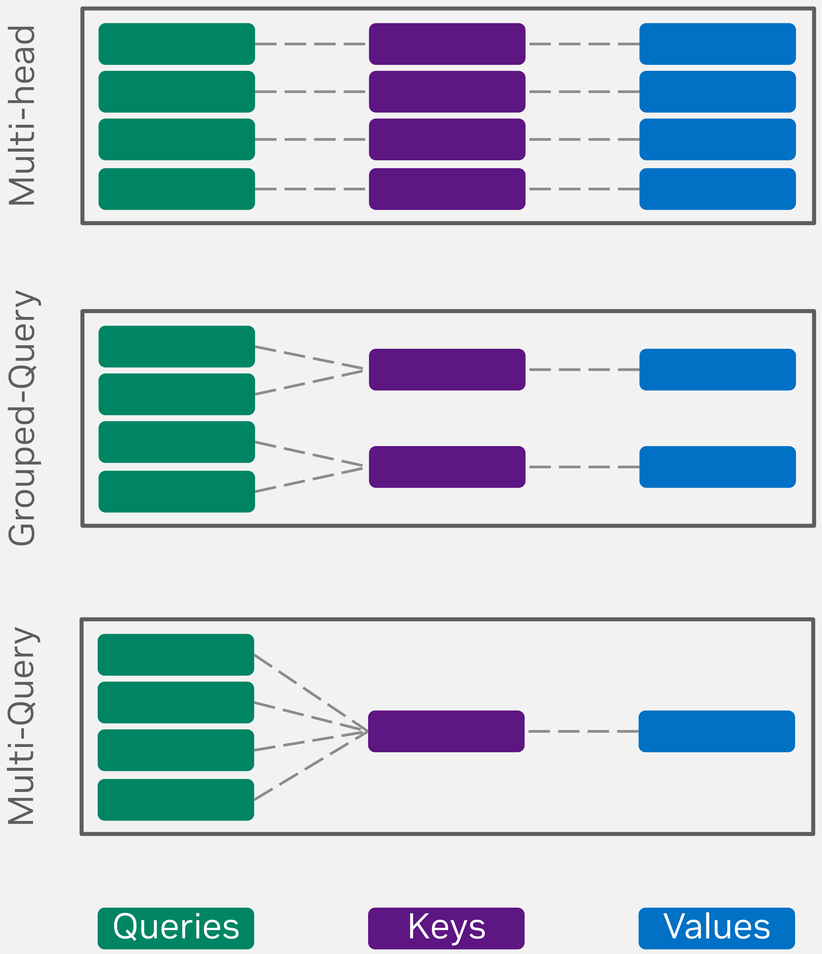
Model	FP32	FP16	BF16	FP8	W8A8 SQ	W8A16	W4A16	W4A16 AWQ	W4A16 GPTQ
Baichuan	Y	Y	Y	Y	Y	Y	Y	Y	Y
BERT	Y	Y	Y	.	.	.	.	.	.
BLIP-2	Y	Y	Y	.	.	.	.	.	.
BLOOM	Y	Y	Y	.	Y	Y	Y	.	.
ChatGLM	Y	Y	Y	.	.	.	.	.	.
ChatGLM-v2	Y	Y	Y	.	.	.	.	.	.
ChatGLM-v3	Y	Y	Y	.	.	.	.	.	.
Flan-T5	Y	Y	Y	.	.	.	.	.	.
GPT	Y	Y	Y	Y	Y	Y	Y	.	.
GPT-J	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

Optimized Attention

Custom Implementations for Attention

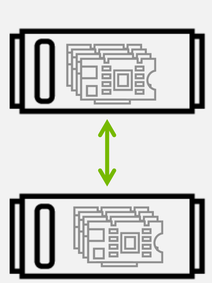
- Custom optimized CUDA kernels for Attention
  - Similar to FlashAttentionV2
- Optimized for A100 & H100 & B200
- Kernels for Encoder & Decoder, as well as context & prefill
- Supports MHA, MQA, GQA



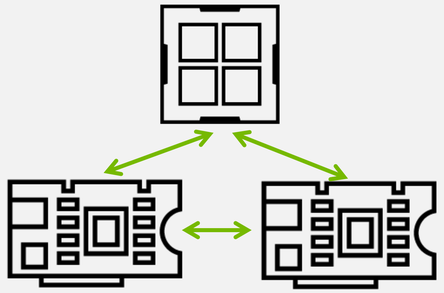
Multi-GPU Multi-Node

Sharding Models across GPUs

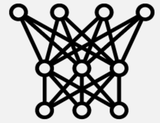
- Supports Tensor & Pipeline parallelism
- Allows for running very large models (tested up to 530B)
- Supports multi-GPU (single node) & multi-node
- TensorRT-LLM handles communication between GPUs
- Examples are parametrized for sharding across GPUs



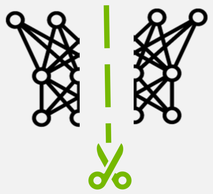
Multi-Node



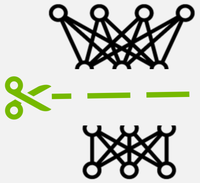
Multi-GPU



No Parallelism



Tensor Parallel



Pipeline Parallel

TensorRT-LLM Usage

NEW High-Level API

***Experimental* High-Level API**

- Simplifies LLM workflow for common models & configs
- Few lines of Python Code to get started with TRT-LLM
- Static configs, and support for inflight-batching

Check out **examples/high-level-api** and provide feedback!

```python
from tensorrt_llm import LLM, ModelConfig

# Instantiate the model
config = ModelConfig(model_dir=[...])
llm = LLM(config)

# Run Inference!
prompts = ["How do I get the fastest LLM Inference?"]
for output in llm.generate(prompts):
    print(output) # >> "Use TensorRT-LLM!"
```

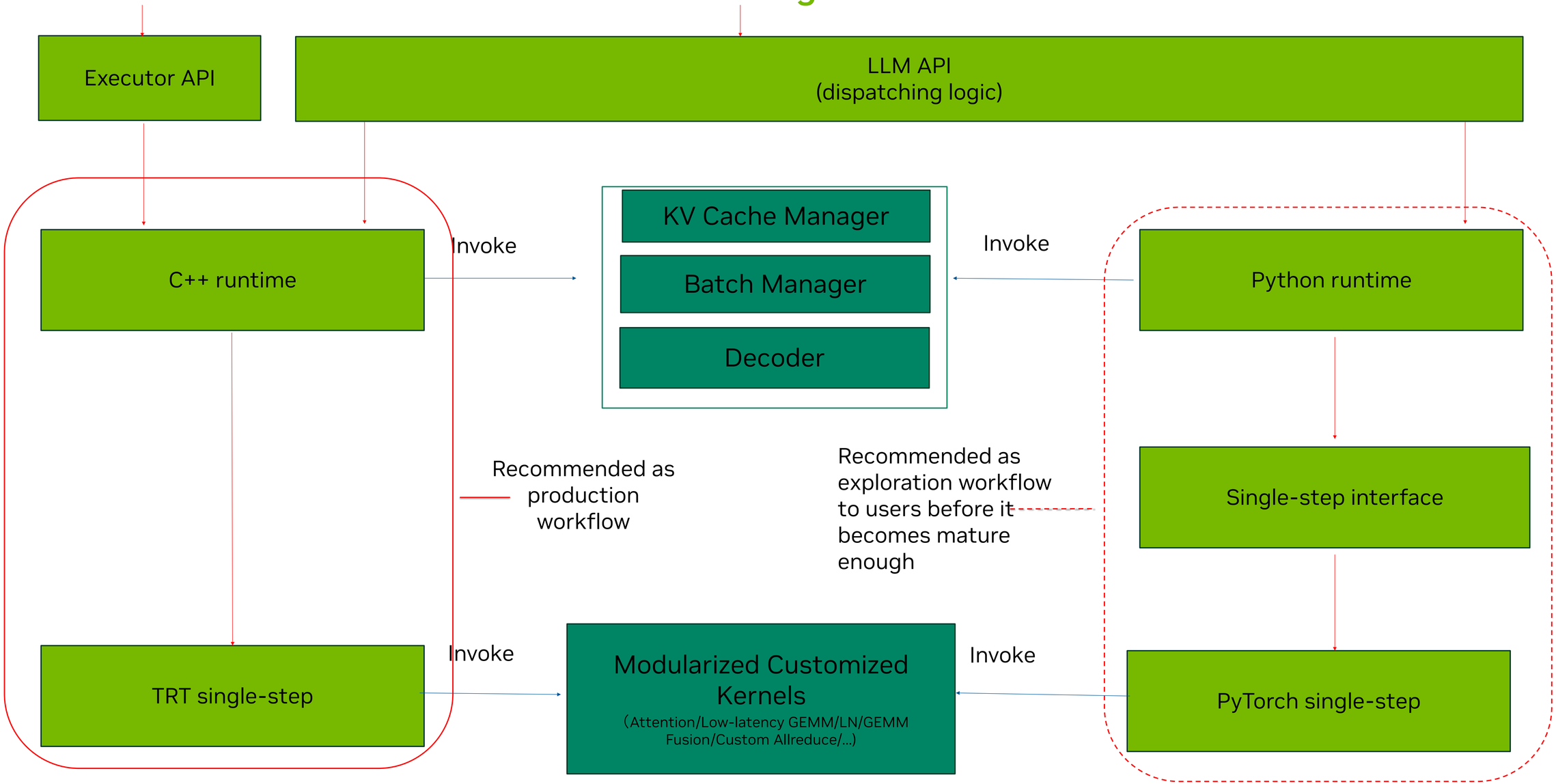
Standard Usage

```python
# Instantiate the model
config = ModelConfig(model_dir=[...])
llm = LLM(config, async=True)

# Run Inference!
prompts = ["How do I get the fastest LLM Inference?"]

async for output in llm.generate_async(prompts, streaming=True):
    print(output)
```

Inflight Batching Usage

PyTorch Workflow release

Starting v0.17

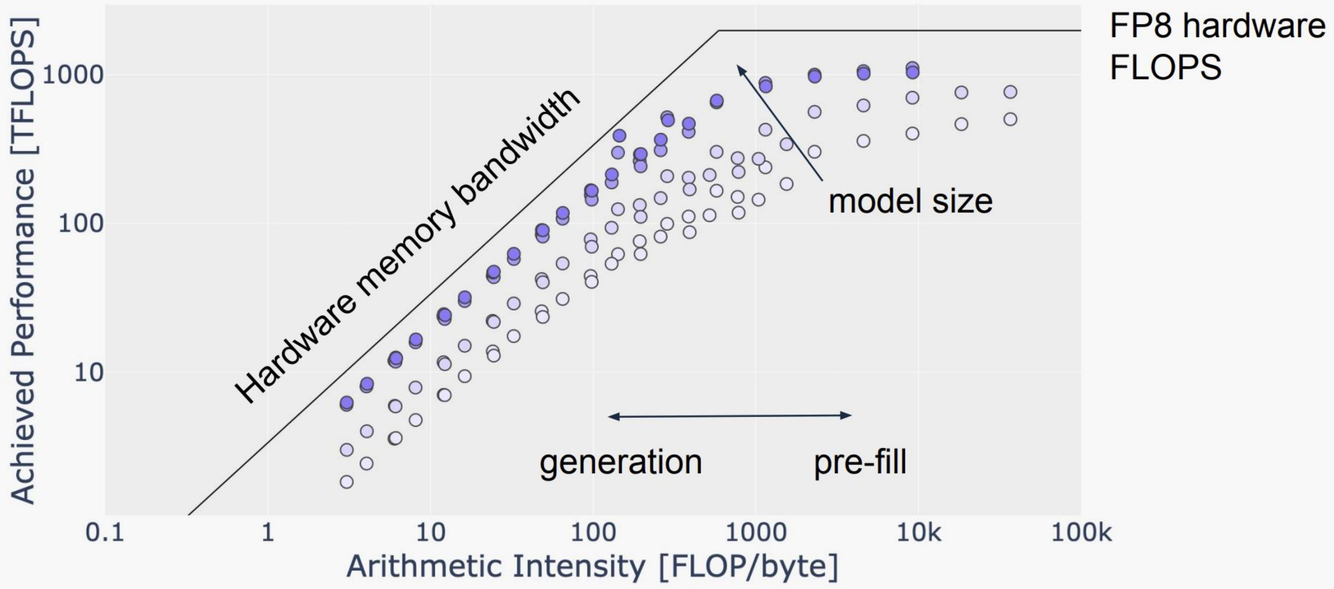


DeepL Use Case

From FP8 LLM Training to Inference: Language AI at Scale
https://www.nvidia.com/en-us/on-demand/session/gtc25-s72799/

DeepL

TensorRT-LLM roofline model



Sweep over different model sizes (1B - 40B dense LLMs), batch sizes, generation and pre-fill lengths on NVIDIA H100

26

NVIDIA Dynamo Platform

The Operating System for AI Factories

NVIDIA Dynamo Platform

NVIDIA Dynamo

Distributed and Disaggregated Generative AI Serving

NVIDIA Dynamo Triton

(formerly Triton Inference Server)

Standardized Model Deployment Across Every AI Workload

NVIDIA NIM

Fastest and Easiest Way to Deploy NVIDIA Dynamo Platform in Production

NVIDIA NIM Microservices

NVIDIA NIM Optimized Inference Microservices

Rapidly deploy reliable building blocks for accelerated generative AI anywhere

Portable Run cloud-native microservices anywhere, maintaining security and control of data and apps

Easy to Use Move fast with the latest agentic AI building blocks for reasoning, retrieval, images and more, deployed in minutes with standard APIs

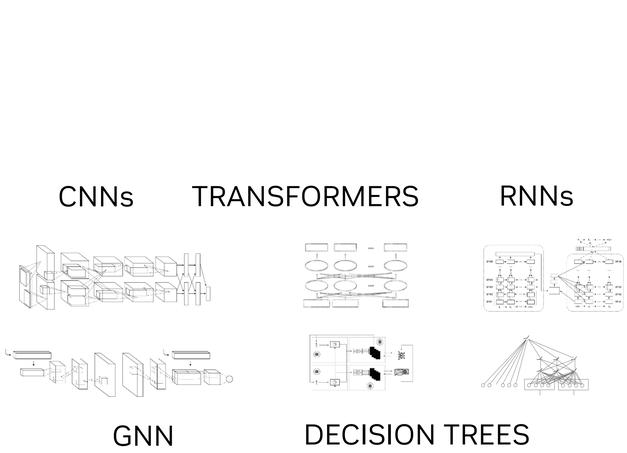
Enterprise Supported Gain confidence with stable APIs, quality assurance, continuous updates, security patching, and support

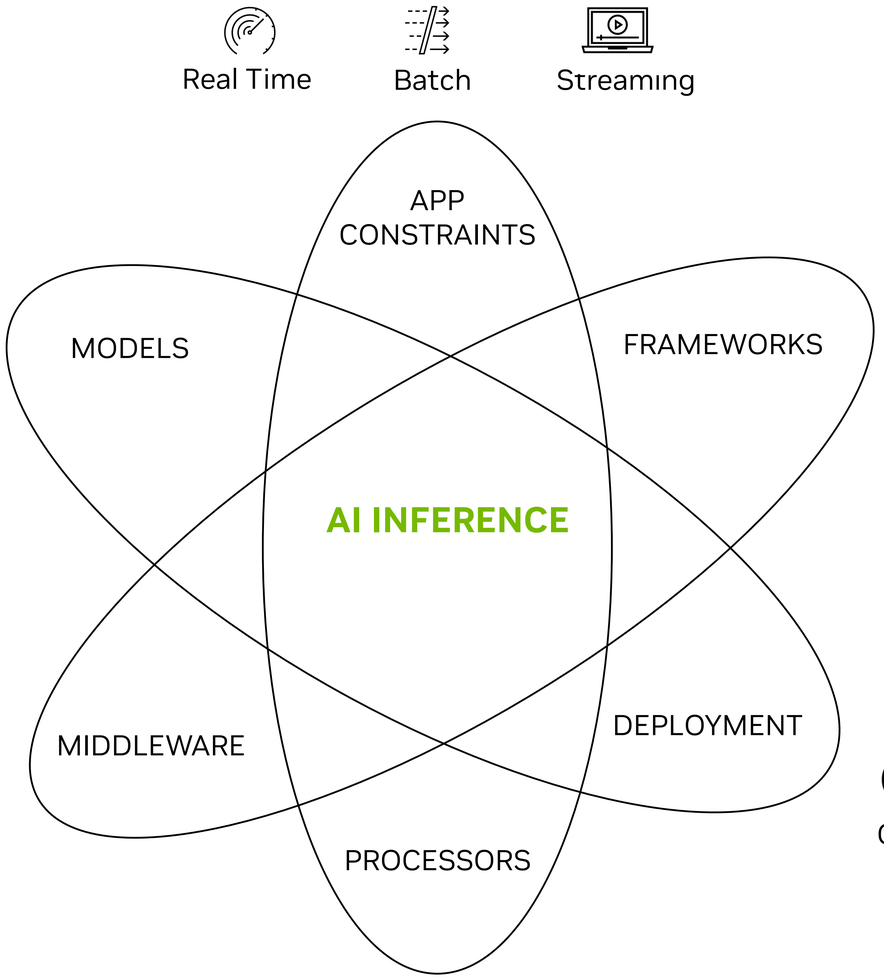
Performance Optimize accuracy, latency and throughput to meet requirements with lowest TCO

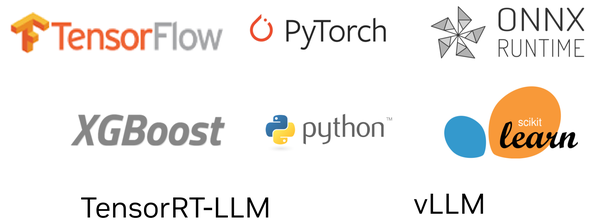
Microsoft Azure · aws · Google Cloud · ORACLE · DGX & DGX Cloud · cisco · DELL Technologies · Hewlett Packard Enterprise · Lenovo · SUPERMICRO

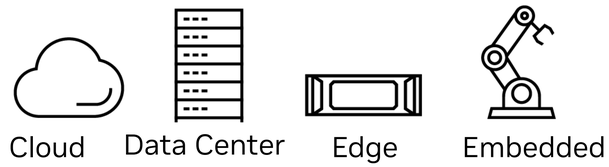
NVIDIA Dynamo Triton
(Formerly Triton Inference Server)

Challenges of AI Inference









Amazon SageMaker
Azure Machine Learning
Google Vertex AI
OCI Data Science
vmware vSphere

NVIDIA Dynamo Triton (formerly Triton Inference Server)

Deploy models from all popular frameworks across GPUs and CPUs

Any Framework

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

Benefits of NVIDIA Dynamo Triton

Standardize AI model deployment and execution across every workload

App developer | ML engineer | MLOPS

Standardized serving across AI models and frameworks

High performance inference

Scale effortlessly with application demand

Faster model rollout

Open source

Constant release of new features

IT, DevOps, Platform team

Consistent deployment across public cloud, on-prem and edge

CPUs and GPUs supported as 1st class citizens

Easy to integrate in existing production infrastructure

Start with CPU, evolve to GPU

GPU accelerated inference leads to $ savings (compute, floor space, power)

NVIDIA AI Enterprise

Enterprise grade & support w/ SLA

Certified on major software platforms

Welcome & Introduction

- Dmitry Mironov is a Senior Deep Learning Solutions Architect at NVIDIA. His focus is on large-scale efficient deployment of LLMs and their inference.
- He has co-authored NVIDIA Deep Learning Institute (DLI) workshops on the topic, including the one on sizing LLM inference systems. Prior to NVIDIA, Dmitry served as a co-founder and CTO of a startup.
- He had been integrating computer vision into gold mining, transportation, energy, and other industries.

Demo 1: TRT-LLM + Triton

https://nvidia-my.sharepoint.com/:v:/r/personal/dmitrym_nvidia_com/Documents/Videos/Clipchamp/Triton-TRT-LLM/Exports/Triton-TRT-LLM.mp4?csf=1&web=1&e=pXDf5c

NVIDIA Dynamo

Pre-Training: Teaching AI Models Knowledge

What is NVIDIA?

NVIDIA is a multinational technology company that specializes in designing and manufacturing graphics processing units (GPUs), high-performance computing hardware, and artificial intelligence (AI) technologies. The company was founded in 1993 by Jensen Huang, Chris Malachowsky, and Curtis Priem.

NVIDIA is headquartered in Santa Clara, California, and has become one of the leading companies in the field of computer graphics, gaming, and AI.

Actual Llama 8B Response

400 TB Web Data

100 Trillion Tokens

Knowledge is not the same as Thinking

I need to figure out where to seat my family of 8 for dinner at a round table.
My in-laws don't get along with my parents.
My sister needs to sit next my little brother to help him eat.
My wife really doesn't want to sit next to my mom.
Where should each person be seated?

Actual Llama 8B Response

Post Training: Teaching Models to Think

6000+ Thinking Tokens – 20X more

300 Answer Tokens

I need to figure out where to seat my family of 8 for dinner at a round table.
My in-laws don't get along with my parents.
My sister needs to sit next my little brother to help him eat.
My wife really doesn't want to sit next to my mom.
Where should each person be seated?

To seat your family of 8 harmoniously at a round table while respecting all constraints, here's the optimal arrangement:

You Wife Dad Mom Sister Little Brother Mother-in-law Father-in-law

Key Constraints Met:

In-laws vs. Parents: Separated by your sister, brother, and you.

Sister & Brother: Adjacent for assistance.

Wife & Mom: Not next to each other (separated by you and dad).

Okay, so I need to figure out seating for a family dinner with 8 people. Let's list out the requirements first to make sure I don't miss anything.

First, the family has 8 members. Let me confirm who they are. The user mentions: in-laws (which I assume are the wife's parents), parents (user's own parents), sister, little brother, wife, and the user themselves. That adds up to 8: user, wife, user's mom, user's dad, wife's mom, wife's dad, sister, and little brother. Wait, that's 7. Hmm, maybe I miscounted. Let me check again. The user says "my family of 8" which includes the user, their spouse (wife), their parents (mo… little brother. That's 8: user, wife, user's mom, user's dad, wife's m…

Now, the constraints:

1. In-laws (wife's parents) don't get along with user's parents. S…
2. Sister needs to sit next to little brother to help him eat. So si…
3. Wife doesn't want to sit next to user's mom. So wife and use…

Additional considerations: Typically, seating arrangements at a r… table. The problem doesn't specify, so maybe I should assume a… circular arrangement. If it's a round table, seating is in a circle, s… neighbor, but the problem might be easier with a round table. Le…

So, the goal is to seat 8 people around a round table with the give…

First, list all the people:

1. User (let's call him U) Wife (W) User's Mom (UM) User's Dad (…
7. Sister (S) Little Brother (LB)

Constraints:

- WM and WD (in-laws) should not be next to UM and UD (parents). So WM and WD cannot be adjacent to UM or UD.
- S must sit next to LB.
- W should not sit next to UM.

Let me start by placing some people to satisfy the strictest constraints first. The sister and little brother needing to be adjacent is a strict requirement. So let's fix S and LB next to each other. Let's place them in two adjacent seats. Let's say positions 1 and 2: S at 1, LB at 2.

Now, the next strict constraint is the in-laws (WM and WD) not being next to parents (UM and UD). So we need to ensure that WM and WD are separated from UM and UD by at least one seat. Also, W doesn't want to sit next to UM.

Another thing to consider is that in many families, couples might want to sit together, but the problem doesn't mention that. However, it's possible that UM and UD would sit together, and WM and WD would sit together. But the problem doesn't specify, so maybe we can assume that couples can be separated if needed, but perhaps it's better to keep them together unless conflicting with constraints. Let's see.

But let's proceed step by step.

First, place S and LB next to each other. Let's fix them at positions 1 and 2.

Now, we need to place the rest: U, W, UM, UD, WM, WD.

We need to keep WM and WD away from UM and UD. Also, W should not be next to UM.

Let me try to place the user and wife in positions that can act as buffers. Maybe placing U and W opposite each other or in positions that separate the conflicting parties.

Alternatively, since in-laws (WM and WD) don't get along with parents (UM and UD), maybe we can place them on opposite sides of the table. Let's try to group the parents and in-laws into separate groups with others in between.

Let me try to create two blocks: one with UM and UD, and another with WM and WD, separated by other people.

But also, W shouldn't be next to UM. So W needs to be away from UM.

Let me attempt a possible arrangement.

Starting with S at 1, LB at 2. Then moving clockwise, let's try to place U and W in positions that can help separate the conflicting groups.

Let's try:

1: S 2: LB 3: ? 4: ? 5: ? 6: ? 7: ? 8: ?

We need to fill positions 3-8 with U, W, UM, UD, WM, WD.

Let me consider placing the user (U) next to LB (position 2) to have a family member next to the little brother. So position 3: U. Then position 8 (next to S) could be someone else.

But then U at 3 would be next to LB at 2 and position 4. Let's see.

Alternatively, maybe place U and W opposite S and LB. Let's see, in an 8-seat round table, opposite of S (position 1) would be position 5. So position 5 could be U or W. Let's try placing U at 5 and W at 6, but need to check constraints.

Wait, maybe a better approach is to divide the table into sections. Let's say S and LB are at 1 and 2. Then, to separate the in-laws from parents, we can place the parents (UM and UD) on one side and in-laws (WM and WD) on the other side, with U and W in between.

But also, W shouldn't sit next to UM. Let's try:

1: S 2: LB 3: UD 4: UM

But wait, if we place UM at 4, then position 3 is UD. Then position 8 (next to S at 1) would be someone. If position 8 is WM or WD, that would put in-laws next to S, which is okay unless S is part of the parents' group. But S is the user's sister, so she's part of the user's …

Now check adjacents:

- S (1) next to LB (2) and UM (8). UM is at 8 next to S (1) and UD (7). But UM is next to UD (her spouse), which is okay. However, S is next to UM, but S is the user's si… between in-laws and pare…

Now, in-laws WM (3) and W… to W (5) and UD (7). UD is…

Now check constraints:

- In-laws (WM and W… at 3, adjacent to 2 (…
- Wife (W at 5) is next…
- Sister (S at 1) is nex… next to UM, which i…

So this arrangement see…

1. In-laws (WM and W… at 7 and 8, next to U…

Okay, so I need to figure… anything.

First, the family has 8 m… parents (user's own par… wife's mom, wife's dad, … of 8" which includes the… little brother. That's 8: u…

Now, the constraints:

1. In-laws (wife's pare…
2. Sister needs to sit n…
3. Wife doesn't want t…

Additional considerati… table. The problem does… circular arrangement. If… neighbor, but the proble…

So, the goal is to seat 8 pe…

First, list all the people:

1. User (let's call him U) Wife (W) Us…
7. Sister (S) Little Brother (LB)

Constraints:

- WM and WD (in-laws) should not be next to UM and UD (parents). So WM and WD cannot be adjacent to UM or UD.
- S must sit next to LB.
- W should not sit next to UM.

Let me start by placing some people to satisfy the strictest constraints first. The sister and little brother needing to be adjacent is a strict requirement. So let's fix S and LB next to each other. Let's place them in two adjacent seats. Let's say positions 1 and 2: S at 1, LB at 2.

Now, the next strict constraint is the in-laws (WM and WD) not being next to parents (UM and UD). So we need to ensure that WM and WD are separated from UM and UD by at least one seat. Also, W doesn't want to sit next to UM.

Another thing to consider is that in many families, couples might want to sit together, but the problem doesn't mention that. However, it's possible that UM and UD would sit together, and WM and WD would sit together. But the problem doesn't specify, so maybe we can assume that couples can be separated if needed, but perhaps it's better to keep them together unless conflicting with constraints. Let's see.

But let's proceed step by step.

First, place S and LB next to each other. Let's fix them at positions 1 and 2.

Now, we need to place the rest: U, W, UM, UD, WM, WD.

We need to keep WM and WD away from UM and UD. Also, W should not be next to UM.

Let me try to place the user and wife in positions that can act as buffers. Maybe placing U and W opposite each other or in positions that separate the conflicting parties.

Alternatively, since in-laws (WM and WD) don't get along with parents (UM and UD), maybe we can place them on opposite sides of the table. Let's try to group the parents and in-laws into separate groups with others in between.

…by other people.

…rate the conflicting groups.

…ween W and UM.

…4). U and W are next to …a plus.

…n't sit next to UM. The only …

… this seems to work.

…it's allowed.

…V in between. But the key is …

…ts 5,6,7,8, so no adjacency.

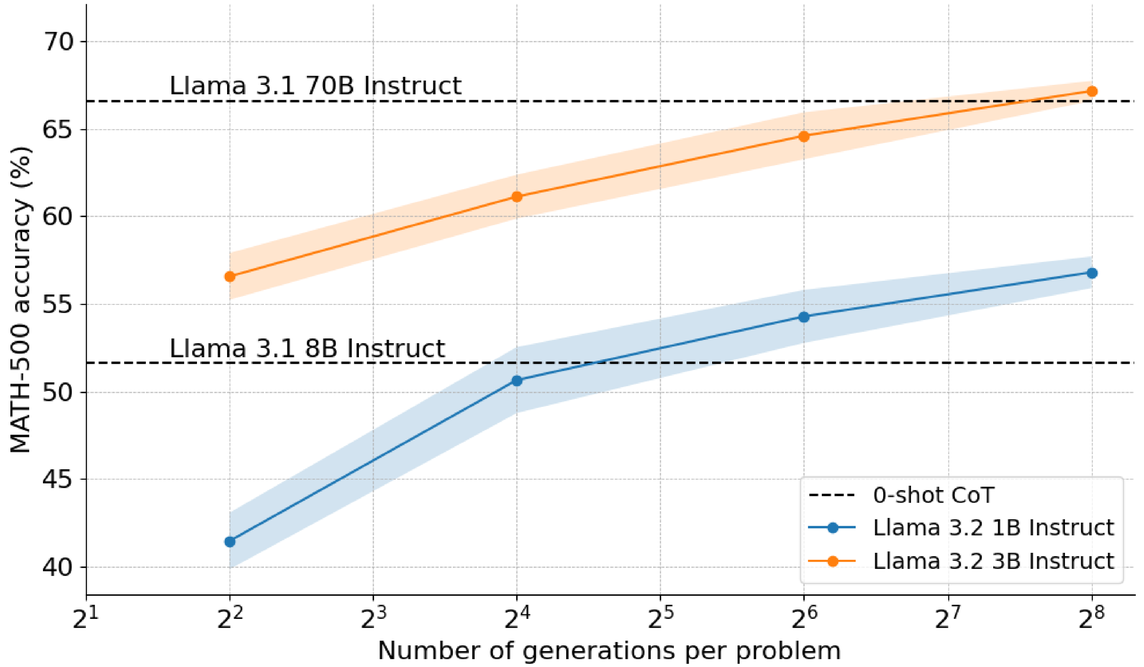
…end solution.

Actual DeepSeek-R1 Response

NVIDIA

What Is Different for Reasoning?

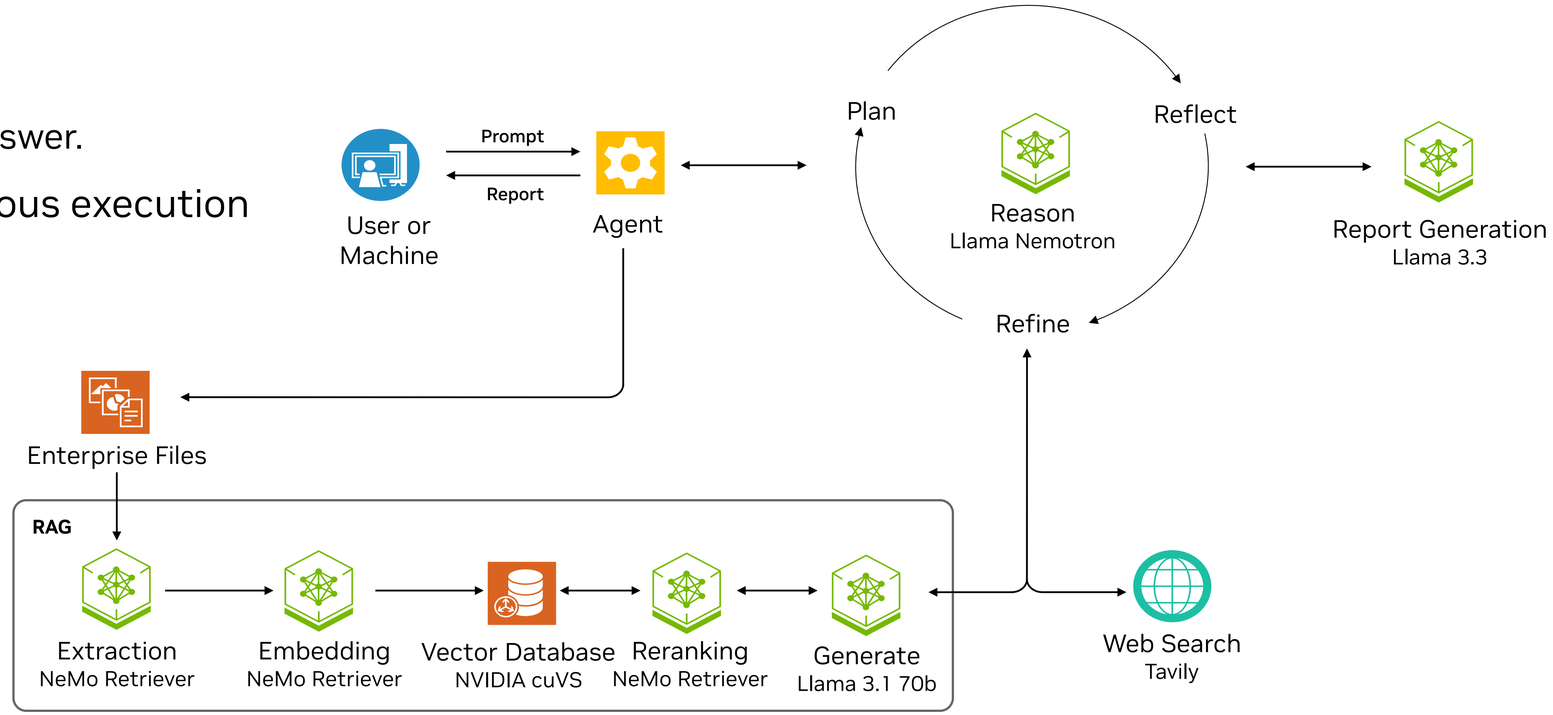
- Reasoning = Inference Time Compute
  - Spending more tokens to get higher accuracy
- Reasoning models have longer outputs.
  - More GPU time / request.
- Distributed Inference is coming.
  - System is optimized as a whole.
  - 2 GPUs are now better than 2 instances 1 GPU each
- Each query is more expensive – therefore input guardrails are common.



https://huggingface.co/spaces/HuggingFaceH4/blogpost-scaling-test-time-compute

How Do Multi-Agentic Workflows Impact Inference

- Complex workflows, where each step may be an LLM call, a tool call, guardrails, RAG pipeline, and other action.
- Lots of steps require E2E latency, because next steps depend on their full output.
- Thus, latencies are long (minutes to hours to days). More of batch-based, offline workflows.
- Complex Inference Time Compute
  - May require reward model to select the answer.
- One user request may include tens of various execution steps.



Inference Compute Requirements Scaling Exponentially

Fueled by reasoning models and AI agents

Larger Models

Hundreds of billions of parameters

Long Thinking Time

100x more thinking tokens

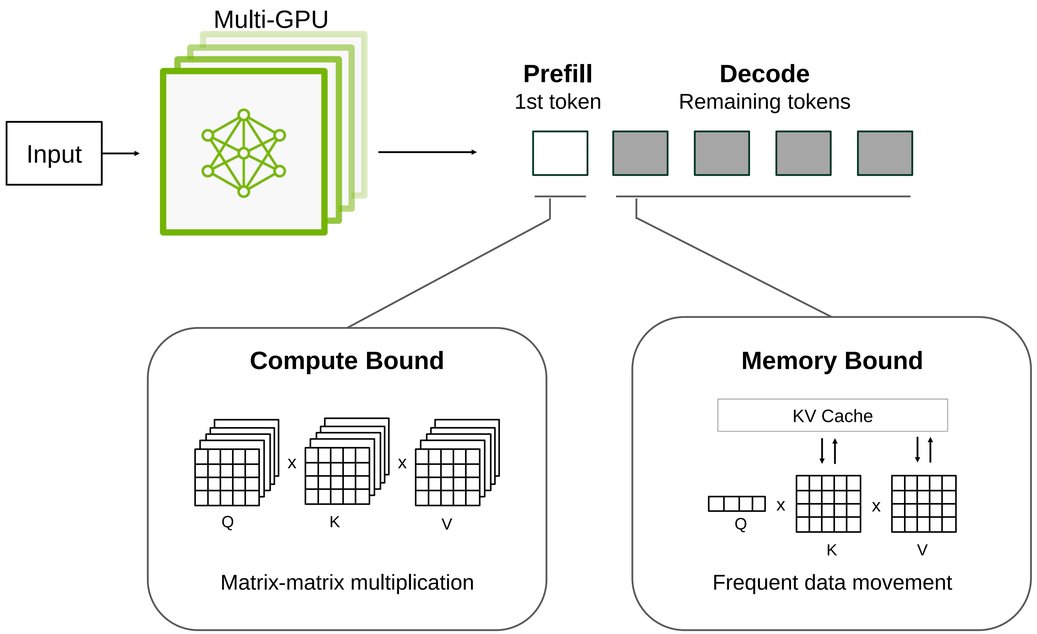
Larger Context

Millions of input tokens

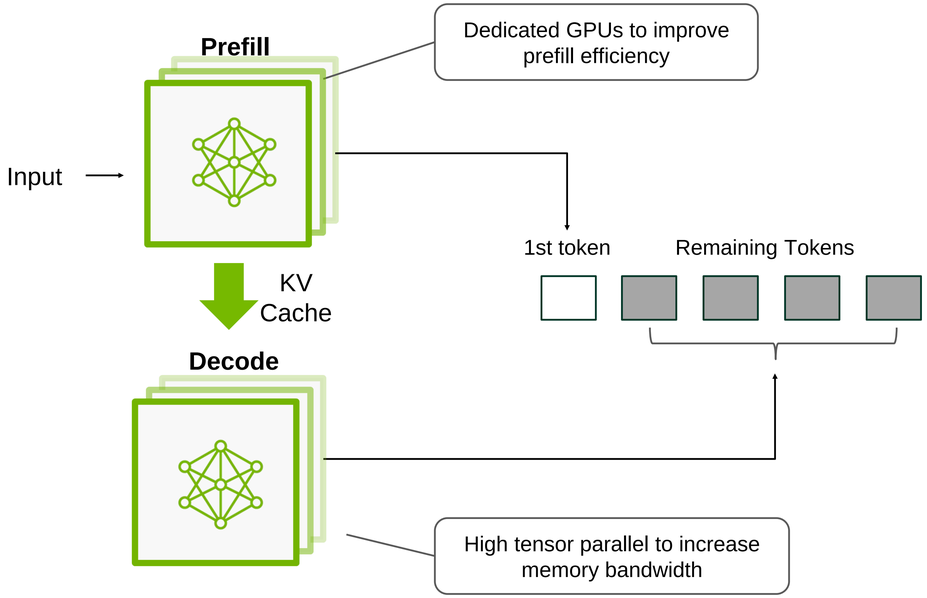
New Inference Optimization Techniques to Boost Inference

Disaggregated serving separates prefill and decode allowing each to be optimized independently

Traditional Serving



Disaggregated Serving



More flexibility to optimize cost and user experience

Announcing NVIDIA Dynamo

AI Inference Software for Reasoning Inference at Scale

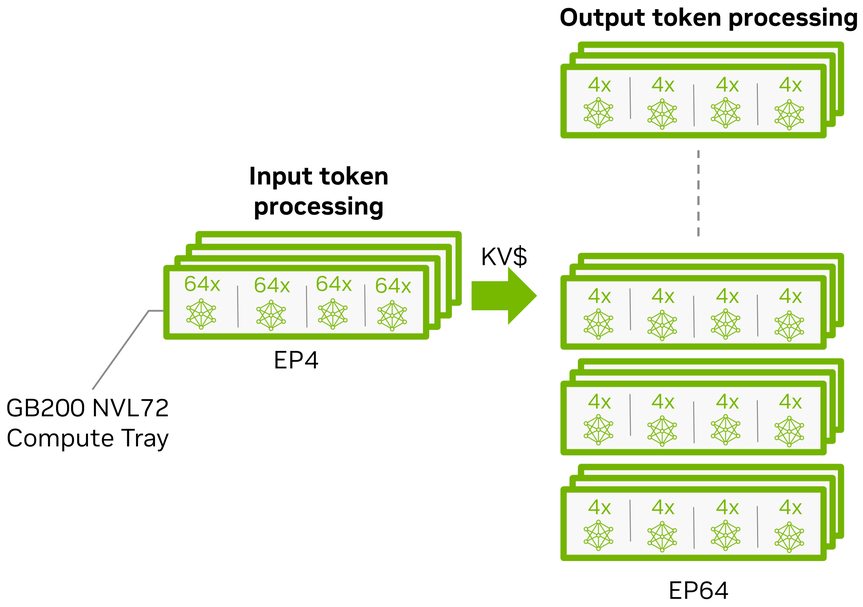
30X
AI Factory Throughput & Revenue
DeepSeek models on Blackwell

1000+
GPU Scale for a single query

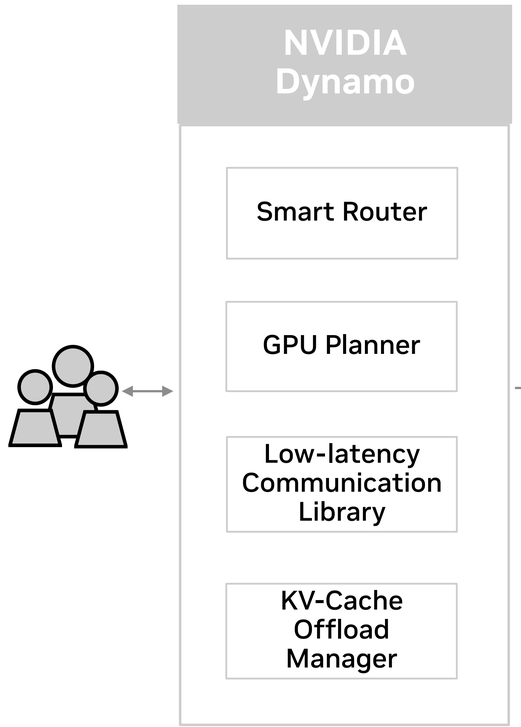
2X
Throughput & Revenue
Llama Models On Hopper

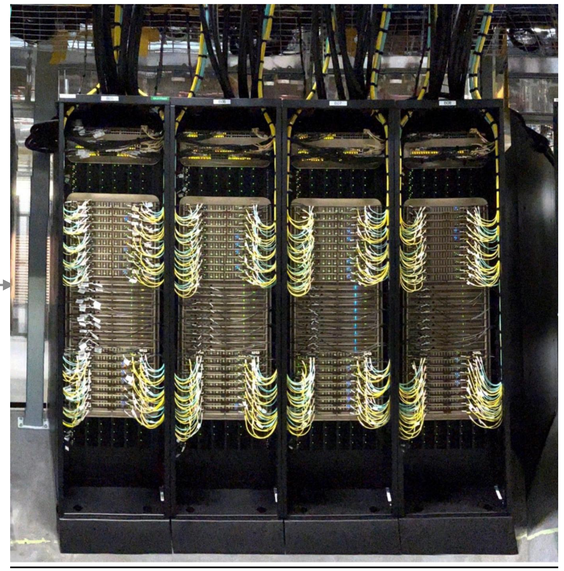
Distributed and Disaggregated Serving

DeepSeek R1



PyTorch SGL TensorRT vLLM

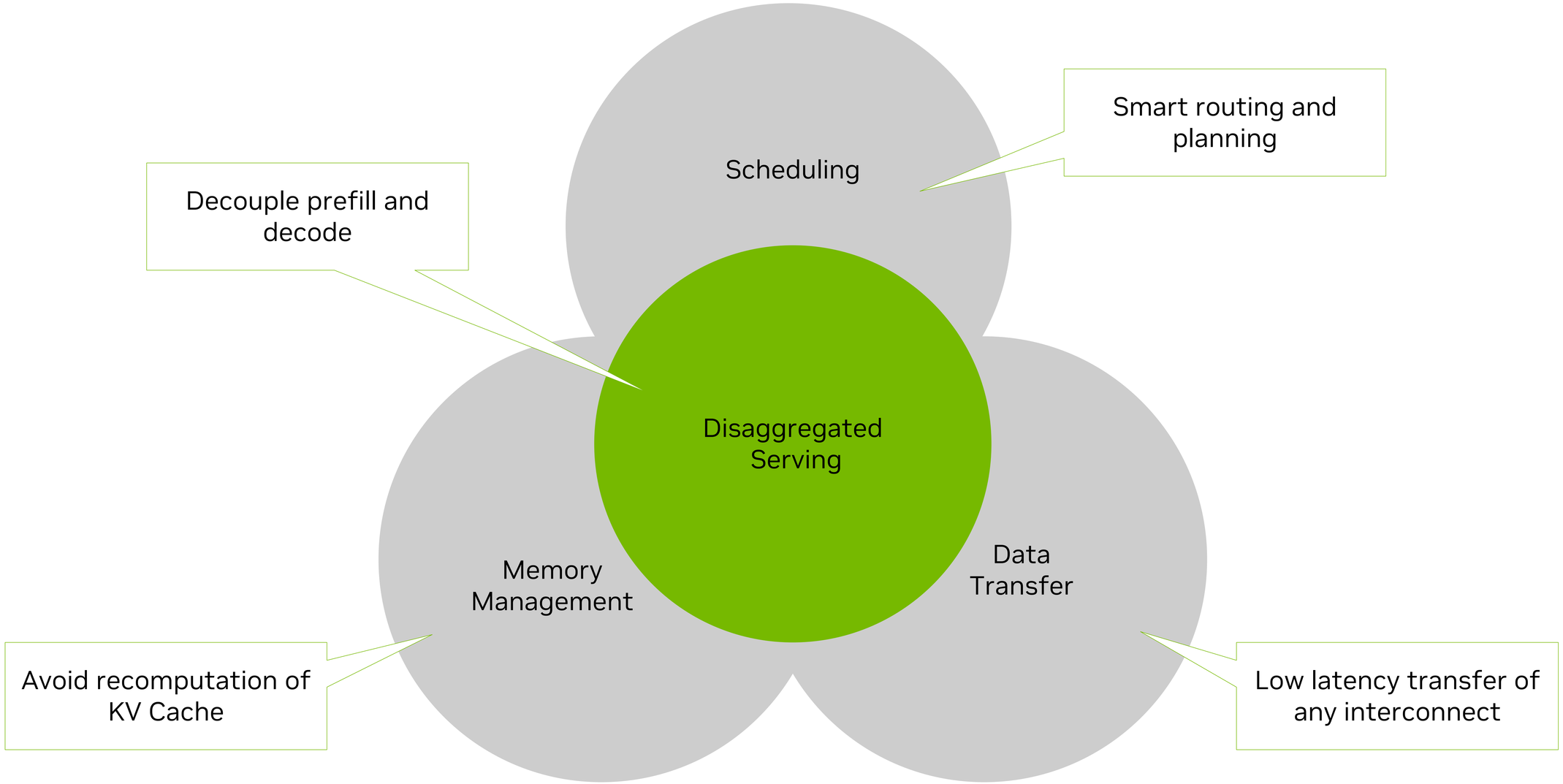




NVIDIA

NVIDIA Dynamo

AI Inference Software for Reasoning Inference at Scale



NVIDIA Dynamo

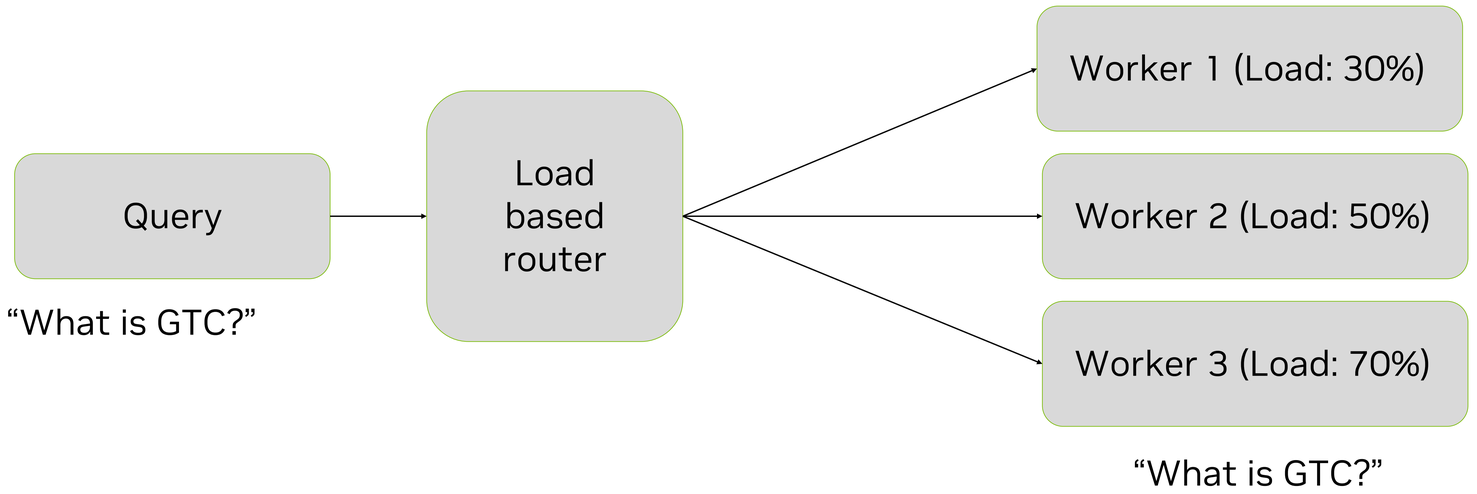
A modular generative AI inference server designed for reasoning models and AI agents

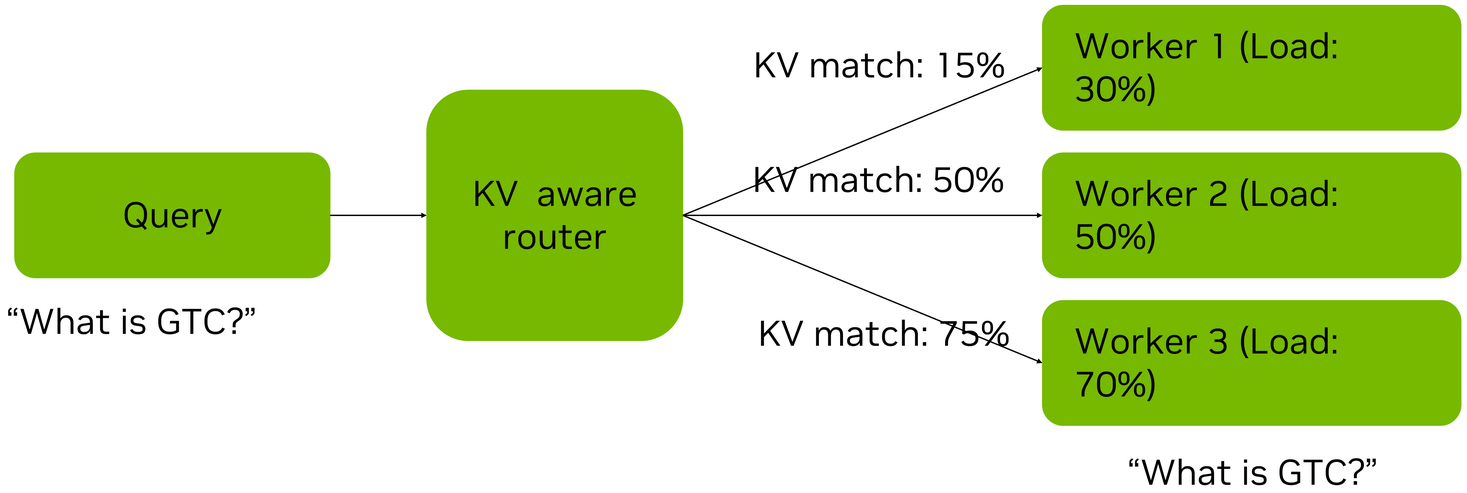
NVIDIA Dynamo

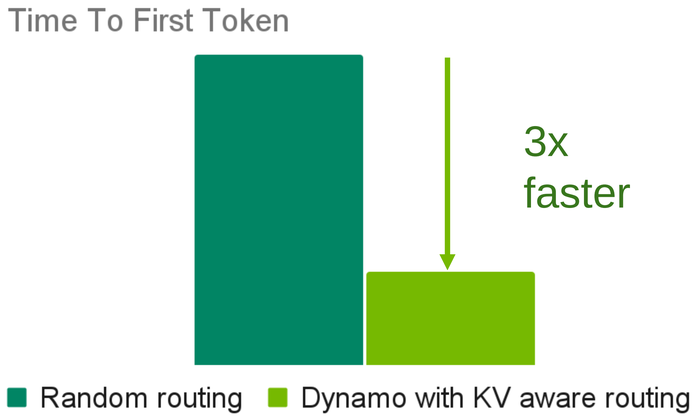
Distributed Inference Serving	Seamlessly scale LLMs from a single GPU to thousands of GPUs
GPU Planning & Scheduling	Meet changing demand patterns w/o over or under provisioning of resources
Smart Request Router	Free up GPU resources by reducing re-computations for similar requests
LLM Engine Plugin	Support for any TensortRT-LLM, vLLM, or SGLang modes
KV Cache Manager	Preserve GPU memory by offloading context (KV$) to cheaper storage

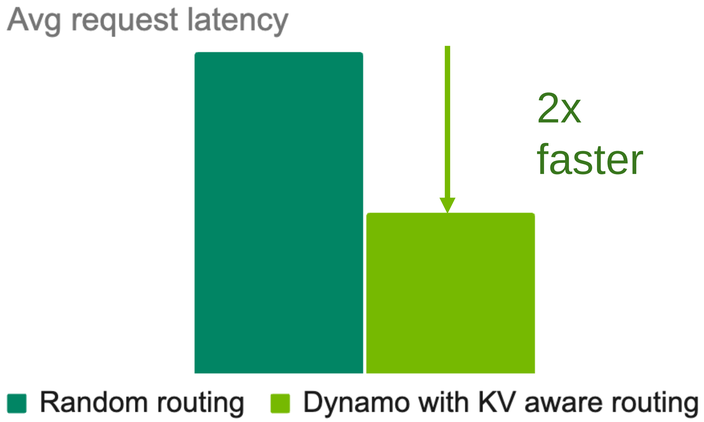
KV Cache Aware Routing

Significant boost in TTFT and End to End Latency with real data (100K requests with R1)





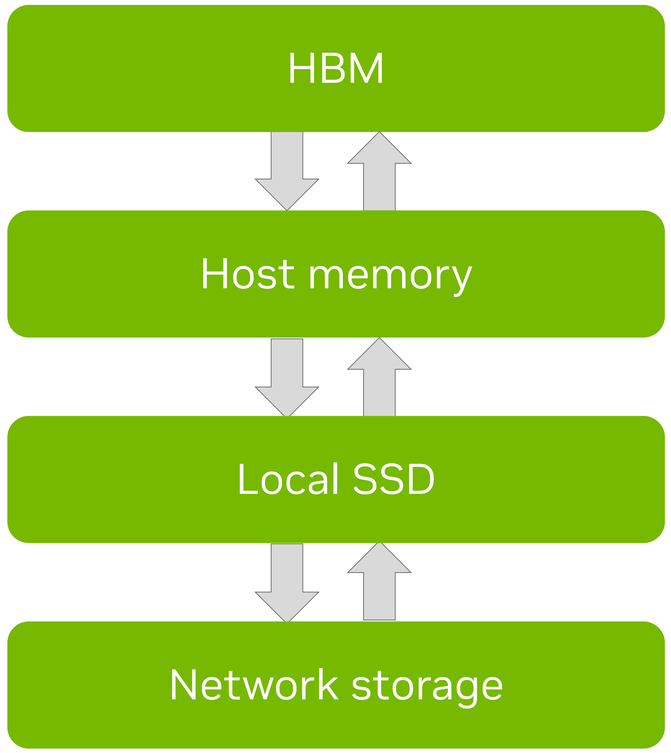




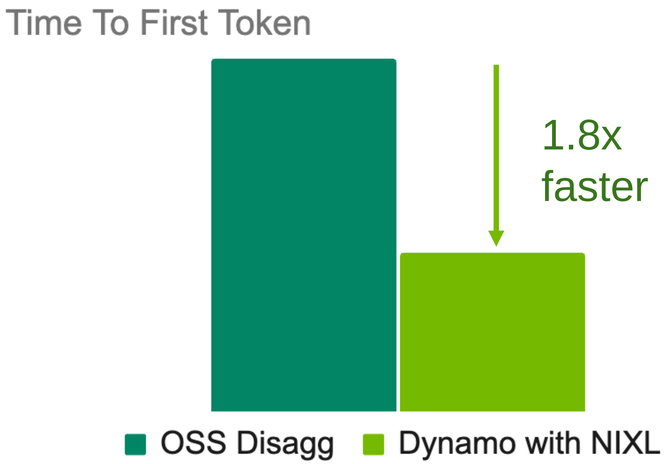
Tested with R1 Distilled Llama 70B over 2 nodes of 8 x H100s with vLLM 0.7.3

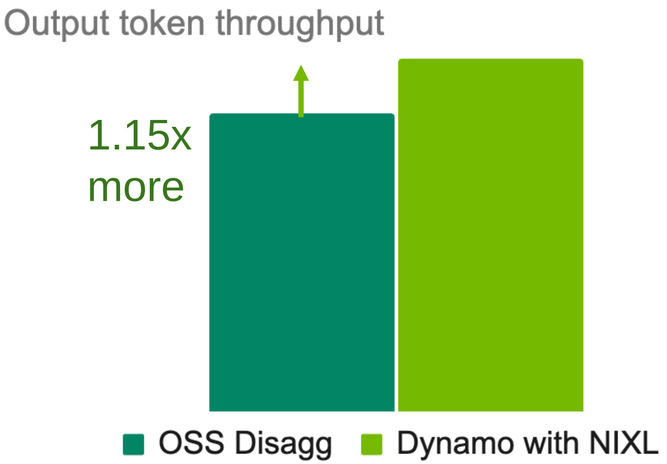
Data Transfer with NIXL (Nvidia Inference TranXfer Library)

Boosts both TTFT and throughput



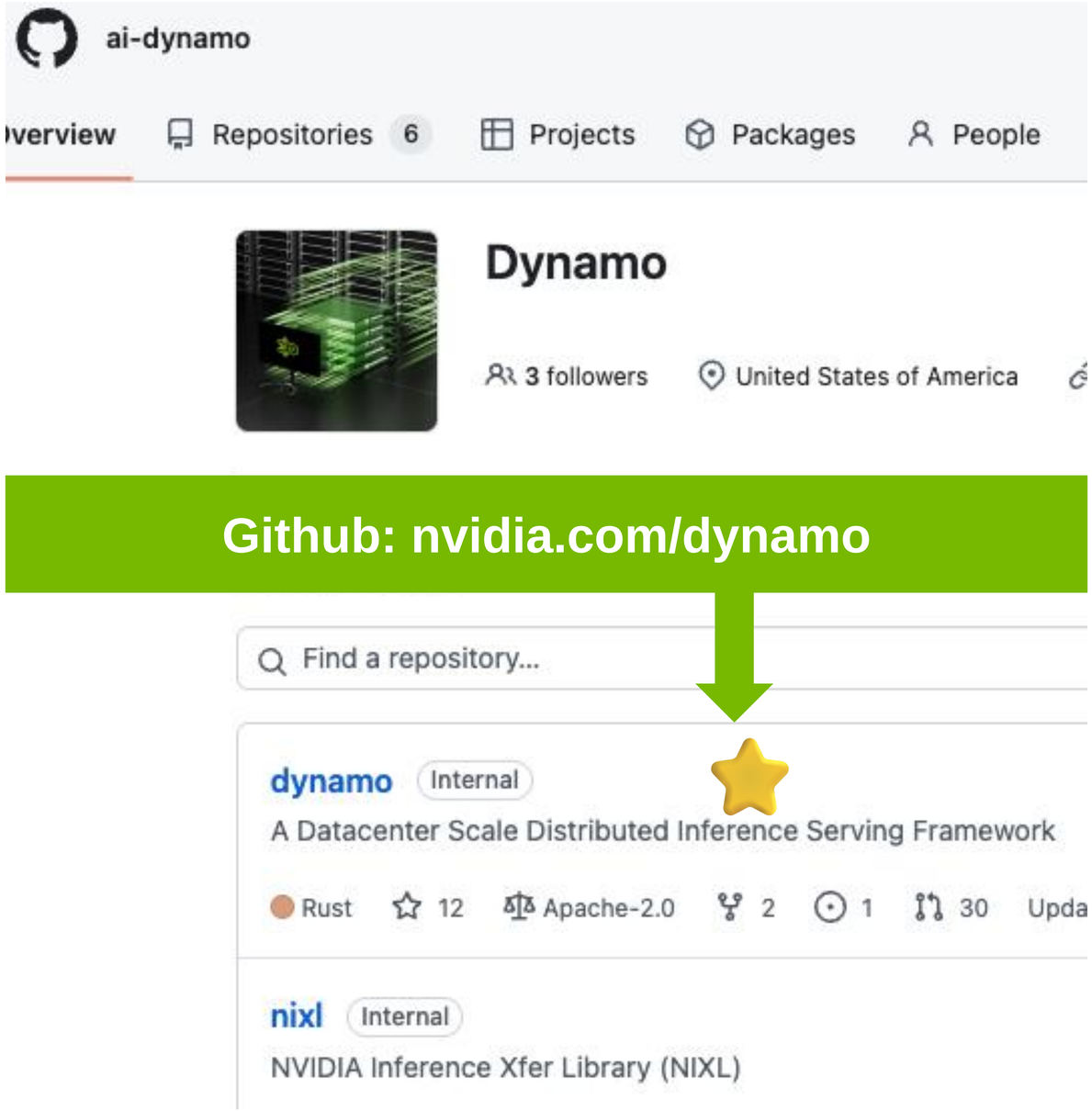
- Low latency transfer between multiple nodes
- Dynamicity required for scaling and transfer to storage
- Common layer for inference specific network optimization.





Tested with vLLM 0.7.3

Dynamo as of Today



- Apache 2 license and public CI
- Pip wheels on PyPi
- Rust for perf and Python for extensibility
- Discord for developer community

- Dynamo CLI
  - dynamo run: Quick start with model, input and output
  - dynamo serve: Construct graph of workers and serve
  - (EA) dynamo build: Containerize
  - (EA) dynamo deploy: Deploy to K8

- Three backends: TRT-LLM, vLLM, & SGLang
- Disaggregated serving with TRT-LLM and vLLM
- KV aware routing with TRT-LLM and vLLM
- (EA) KV manager with vLLM
- NIXL for RDMA and TCP (fallback for AWS EFA)

Customer Testimonial Highlights

Cohere

“Scaling advanced AI models requires sophisticated multi-GPU scheduling, seamless coordination and low-latency communication libraries that transfer reasoning contexts seamlessly across memory and storage. We expect NVIDIA Dynamo will help us deliver a premier user experience to our enterprise customers.” **Saurabh Baji, Senior Vice President of Engineering at Cohere**

Perplexity AI

"Handling hundreds of millions of requests monthly, we rely on NVIDIA’s GPUs and inference software to deliver the performance, reliability, and scale our business and users demand, "We'll look forward to leveraging NVIDIA Dynamo with its enhanced distributed serving capabilities to drive even more inference serving efficiencies and meet the compute demands of new AI reasoning models." **Denis Yarats, CTO of Perplexity AI**.

Together AI

“Scaling reasoning models cost-effectively requires new advanced inference techniques, including disaggregated serving and context-aware routing. Together AI provides industry leading performance using our proprietary inference engine. The openness and modularity of NVIDIA Dynamo will allow us to seamlessly plug its components into our engine to serve more requests while optimizing resource utilization—maximizing our accelerated computing investment." **Ce Zhang, CTO of Together AI**.

Dynamo Single GPU.mp4

Demo 2: Dynamo

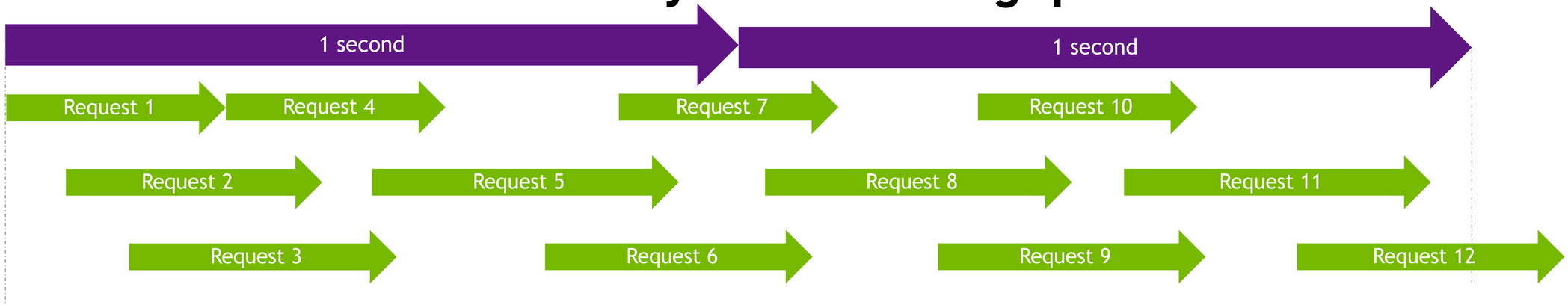
https://nvidia-my.sharepoint.com/:v:/r/personal/dmitrym_nvidia_com/Documents/Videos/Clipchamp/Dynamo%20Single%20GPU/Exports/Dynamo%20Single%20GPU.mp4?csf=1&web=1&e=adLKjk

Choosing the Optimal Solution for Your Needs

	NIM	Dynamo Triton (formerly Triton Inference Server)	Dynamo
Key Benefits	Fastest path for deployment	Standardizes AI model deployment and execution across every workload	Serves generative AI models in large scale distributed environments
Reasons to select	Out of the box performance with desired model.	Extensible deployment with any model and desired backend	Taking advantage of system level optimization such as disaggregated serving, smart routing with KV awareness
Supported via NVAIE	Yes	Yes	Planned
Support for VLMs	Yes	Yes	Planned
Support K8s	Yes	Yes	Yes
Separating Prefill and Decoding Nodes	Planned	No	Yes
Supports for any LLM	Coming soon	Yes	Yes

GenAI-Perf

Concurrency is NOT throughput



- Top and bottom: two different settings of a serving system, same model, same requests, same ISL and OSL.
- Top system: **concurrency** 3 (serving 3 concurrent requests), average E2E Latency 250 ms, RPS = 5.5
- Bottom system: **concurrency** 6, average E2E Latency 2000 ms (slower!), RPS = 3 (more GPUs needed!)

Concurrency is NOT Number of Concurrent Users

Example

- Application developers may call it 100 concurrent users:
  - some are typing
  - some are listening to the generated response
  - some are waiting for the interface to load

- **Convert Concurrent Users to Target Throughput**
- **Given**
- 100 **concurrent users** at peak
- Each session is 5 minutes
- 10 requests to the LLM per session
- **Then**
- Each user submits 10 requests / 5 min = 2 req / min / user
- 100 concurrent users submit 100 * 2 = 200 req / min = 3.3 req / s at peak (**target RPS**)

GenAI-Perf

Open-Source Benchmarking Client

1. Simulate LLM requests

2. Measure time to get response
to calculate latency and throughput

GenAI-Perf

Any OpenAI-like API

Open Source Benchmarking tool for GenAI Performance measurement

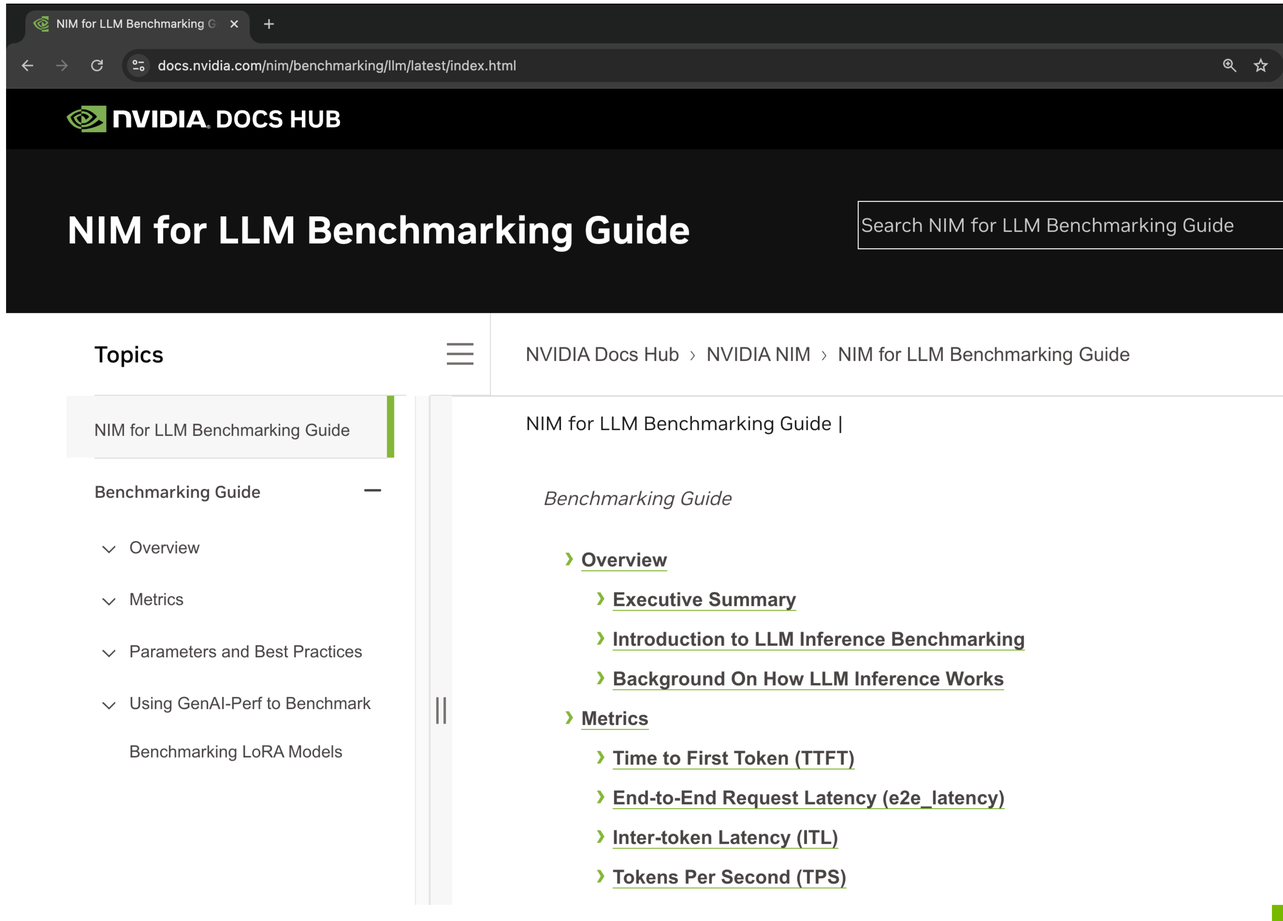
Enables apples to apples comparison for evaluation and perf improvement.

- Measured via KServe & OpenAI API
- Agnostic of underlying software and hardware

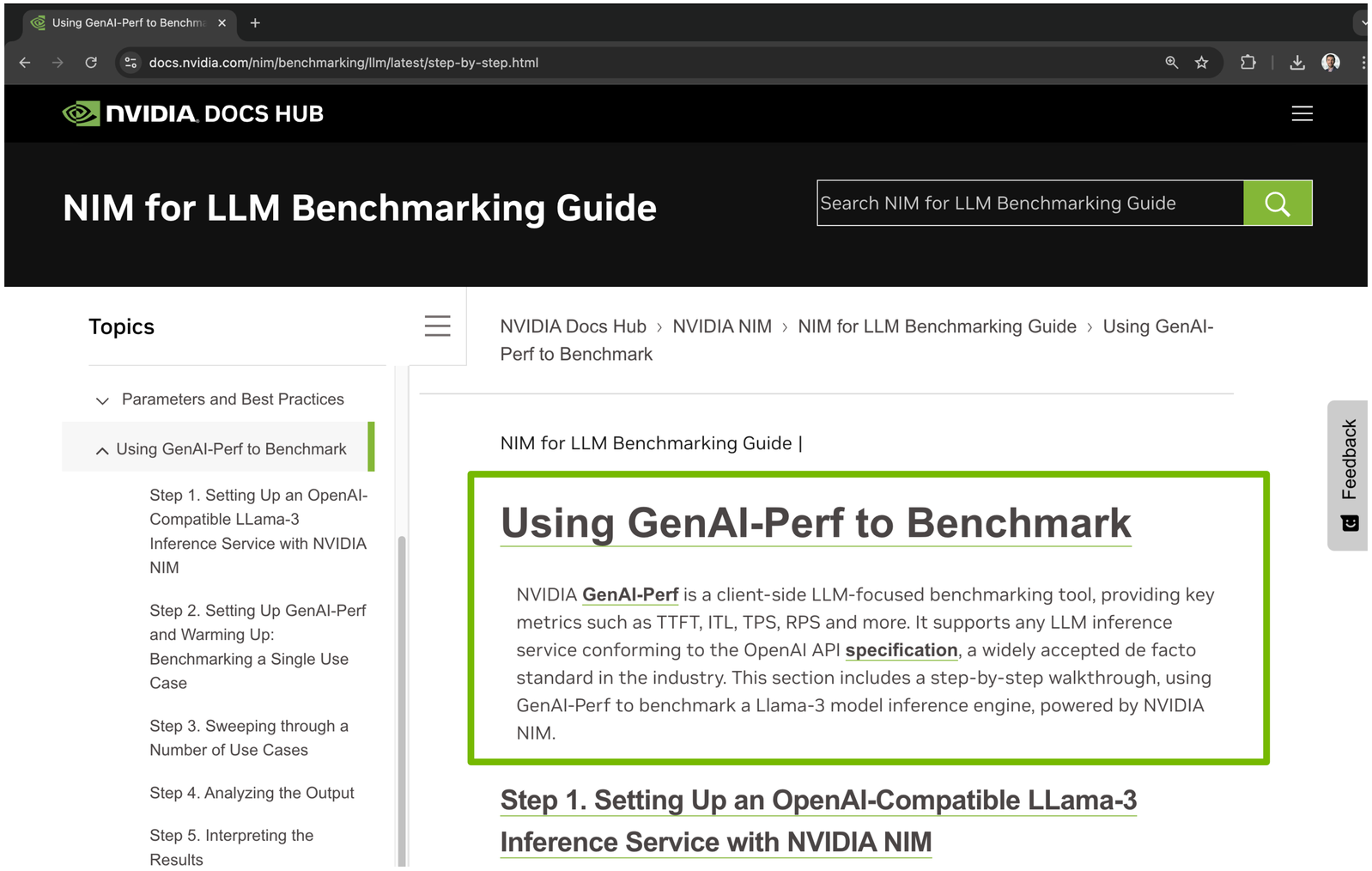
Provides refined implementation of metrics to guide action

- **Inter Token Latency (ITL):** GenAI Perf's ITL metric focuses solely on decoding, excluding the initial token. This adjustment ensures a more targeted assessment of processing efficiency during decoding.

- Documented in NIM benchmarking guide
- https://docs.nvidia.com/nim/benchmarking/llm/latest/index.html



GenAI-Perf to Benchmark



GenAI-Perf command

Sample output generated by GenAI-Perf

```
export INPUT_SEQUENCE_LENGTH=200
export INPUT_SEQUENCE_STD=10
export OUTPUT_SEQUENCE_LENGTH=200
export CONCURRENCY=10
export MODEL=meta/llama3-8b-instruct

genai-perf profile \
    -m $MODEL \
    --endpoint-type chat \
    --service-kind openai \
    --streaming \
    -u "http://meta-llama3-1-8b-instruct:8000" \
    --synthetic-input-tokens-mean $INPUT_SEQUENCE_LENGTH \
    --synthetic-input-tokens-stddev 0 \
    --concurrency $CONCURRENCY \
    --output-tokens-mean $OUTPUT_SEQUENCE_LENGTH \
    --extra-inputs max_tokens:$OUTPUT_SEQUENCE_LENGTH \
    --extra-inputs min_tokens:$OUTPUT_SEQUENCE_LENGTH \
    --extra-inputs ignore_eos:true \
    --measurement-interval 5000 \
    --artifact-dir /genai-perf-results \
    --profile-export-file ${INPUT_SEQUENCE_LENGTH}_${OUTPUT_SEQUENCE_LENGTH}_${CONCURRENCY}.json
    -- \
    -v \
    --request-count $((10 * CONCURRENCY)) \
    --max-threads=256
```

LLM Metrics

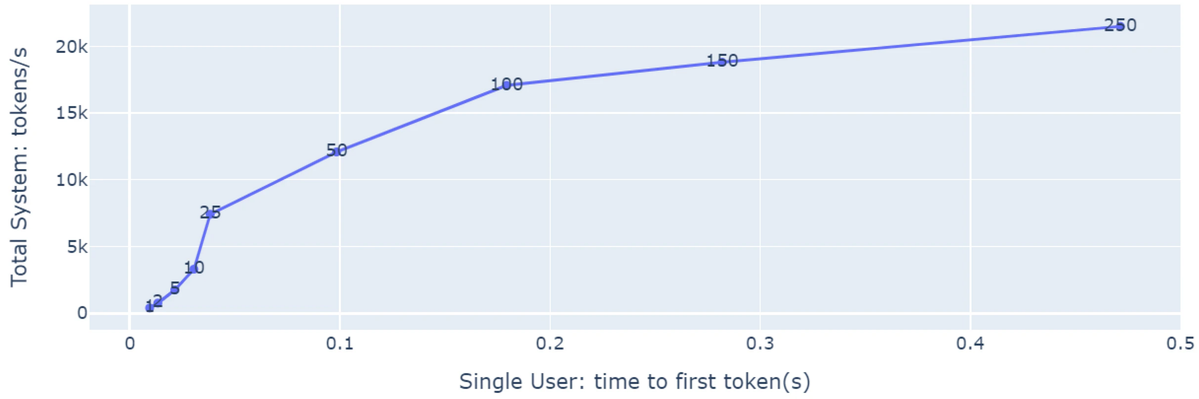
Statistic	avg	min	max	p99
Time to first token (ns)	85,485,242	27,402,273	152,621,817	130,194,943
Inter token latency (ns)	8,847,758	2,113,030	74,794,303	9,477,464
Request latency (ns)	1,848,822,497	1,844,511,394	1,924,017,143	1,905,132,459
Num output token	184	177	190	189
Num input token	200	198	201	200

Output token throughput (per sec): 995.61
Request throughput (per sec): 5.41

Sweeping across concurrencies

Running multiple GenAI-Perf calls

```
for concurrency in 1 2 5 10 50 100 250; do

    local INPUT_SEQUENCE_LENGTH=$inputLength
    local INPUT_SEQUENCE_STD=0
    local OUTPUT_SEQUENCE_LENGTH=$outputLength
    local CONCURRENCY=$concurrency
    local MODEL=meta/llama3-8b-instruct

    genai-perf \
        -m $MODEL \
        --endpoint-type chat \
        --service-kind openai \
        --streaming \
        -u localhost:8000 \
        --synthetic-input-tokens-mean $INPUT_SEQUENCE_LENGTH \
        --synthetic-input-tokens-stddev $INPUT_SEQUENCE_STD \
        --concurrency $CONCURRENCY \
        --output-tokens-mean $OUTPUT_SEQUENCE_LENGTH \
        --extra-inputs max_tokens:$OUTPUT_SEQUENCE_LENGTH \
        --extra-inputs min_tokens:$OUTPUT_SEQUENCE_LENGTH \
        --extra-inputs ignore_eos:true \
        --tokenizer meta-llama/Meta-Llama-3-8B-Instruct \
        --measurement-interval 10000 \
```



GenAI-Perf Demo.mp4

Demo 3: GenAI-Perf

https://nvidia-my.sharepoint.com/:v:/r/personal/dmitrym_nvidia_com/Documents/Videos/Clipchamp/GenAI-Perf%20Demo/Exports/GenAI-Perf%20Demo.mp4?csf=1&web=1&e=aE0nyA

Call to Action

- Intro to LLMs
- Discover NVIDIA AI Enterprise Software
- Discover AI Inference Solution
- Explore NVIDIA AI Inference Performance
- Blog: Mastering LLM Techniques: Inference Optimization Technical Blog
- Explore NVIDIA TensorRT SDK
- NGC NIM Containers: Phind-CodeLlama-34B-v2-Instruct
  - NGC Containers: Llama-3.1-Nemotron-70B-Instruct
  - NGC Containers: Llama-3-Taiwan-70B-Instruct
- LLM Inference Sizing: Benchmarking End-to-End Inference Systems
- Transforming Medical Workflows With AI: A Deep Dive into CLLMs

Claim your Free Self–Paced Course

Access essential technical training

Sharpen your skills or learn a new technology. In partnership with NVIDIA Deep Learning Institute, we are offering a free self-paced course (worth up to $90).

Courses on offer include:

- Accelerating End-to-End Data Science Workflows
- Building LLM Applications With Prompt Engineering
- Building RAG Agents with LLMs
- Building Real-Time Video AI Applications
- Deploying a Model for Inference at Production Scale
- Fundamentals of Accelerated Computing with CUDA Python
- Generative AI with Diffusion Models
- Get Started with Highly Accurate Custom ASR for Speech AI
- Sizing LLM Inference Systems
- Introduction to Deploying RAG Pipelines for Production at Scale
- Introduction to NVIDIA NIM™ Microservices

Scan the QR code to access the full course list and redeem your free training. (Redemption rule applies)

Thank You !

GTC 2025- Watch on Demand

What's Next in AI Starts Here

March 17–21, 2025

Demo Listing

```
ARG
  BASE_IMAGE=nvcr.io/nvidia/tritonserver:2
  5.03-trtllm-python-py3
FROM ${BASE_IMAGE} AS base

  FROM base AS dev

  # Install additional packages
RUN pip install transformers==4.51.3 && \
    pip install flash-attn --no-build-
  isolation && \
    pip install
  /opt/tritonserver/python/triton*.whl

  # Clone and set up OpenAI frontend
RUN git clone https://github.com/triton-
  inference-server/server.git && \
    cd server/python/openai/ && \
    pip install -r requirements.txt

  # Set working directory
WORKDIR /workspace

  FROM dev AS final
```

```
from transformers import AutoModelForCausalLM, AutoTokenizer
import torch

   # Load model and tokenizer from local path
# model_path = "/models/models--meta-llama--Llama-4-Scout-17B-16E-
   Instruct/current_snapshot"
model_path = "/models/Llama-3.1-8B-Instruct"
print(f"Loading model from local path: {model_path}")

   try:
    # Use trust_remote_code=True for custom model code
    model = AutoModelForCausalLM.from_pretrained(
        model_path,
        device_map="auto",
        torch_dtype=torch.float16,  # Use float16 as specified in the
   build command
        local_files_only=True
    )
    tokenizer = AutoTokenizer.from_pretrained(
        model_path,
        local_files_only=True
    )
    print("Model and tokenizer loaded successfully!")
    print(f"Model type: {type(model).__name__}")
    print(f"Tokenizer type: {type(tokenizer).__name__}")

       # Test inference
    print("\nRunning test inference...")
    test_prompt = "What is the capital of France?"
    print(f"Test prompt: {test_prompt}")

    # Tokenize input
    inputs = tokenizer(test_prompt,
   return_tensors="pt").to(model.device)

    # Generate response
    with torch.no_grad():
        outputs = model.generate(
            **inputs,
            max_new_tokens=50,
            temperature=0.7,
            top_p=0.9,
            do_sample=True
        )

    # Decode and print response
    response = tokenizer.decode(outputs[0], skip_special_tokens=True)
    print(f"Model response: {response}")

   except Exception as e:
       print(f"Error loading model: {str(e)}")
```

Demo Listing 2


```
pip install --upgrade
huggingface-hub
huggingface-cli download meta-
llama/Llama-3.1-8B-Instruct --
local-dir
/mnt/shared/work/Llama-3.1-8B-
Instruct
docker compose up --build
triton-trt-llm-build
docker compose up trt-llm
docker compose up trt-llm-build
docker compose up triton-deploy
docker compose up triton-sdk-
genai-perf
```

```
export HOME=$(pwd)
 python3 -
 m venv venv && . venv/bin/act
 ivate

 # install dynamo
 pip install ai-dynamo[all]

 # run dynamo
 dynamo run out=vllm deepseek-
 ai/DeepSeek-R1-Distill-Llama-
 8B
```