
NVIDIA
Triton Inference Server Meetup
09/09/2024

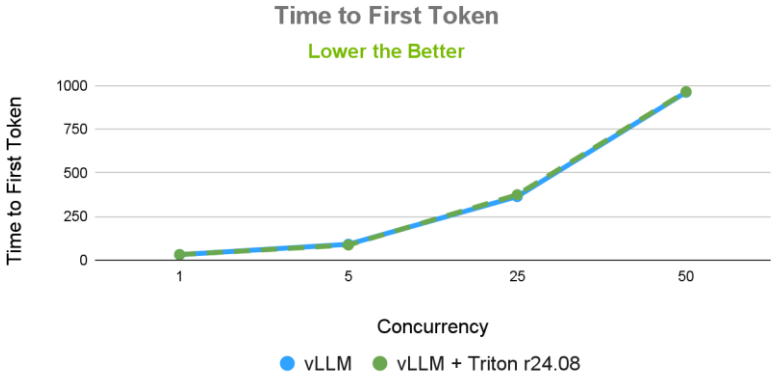
Agenda

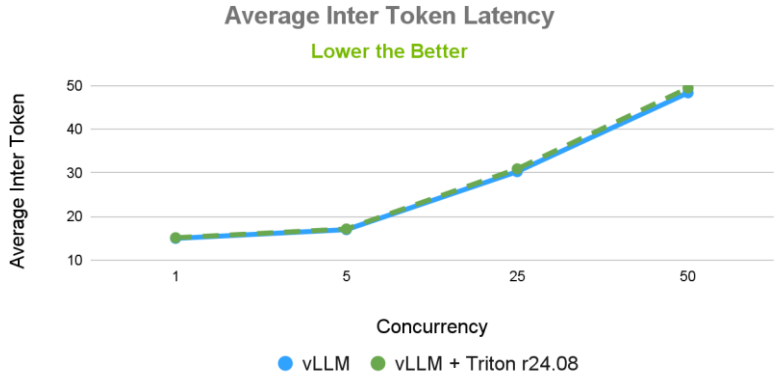
- Triton vLLM Backend Updates
- Triton Inference Server Updates
- Open Source Improvements and Deliverables
- GenAI Perf
- Triton 3 and Disaggregated Serving

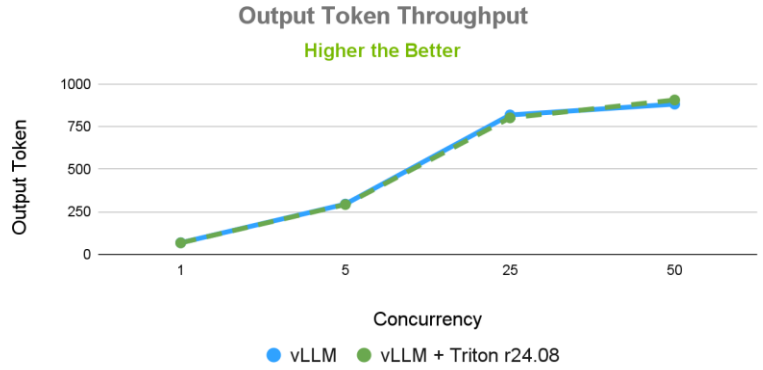
Triton vLLM Backend Updates

vLLM Backend Performance Improvement

Llama2-7B on single A100 40GB with 200 input - 1000 output tokens using vLLM 0.5.3 post







Within < 2% to vLLM's performance for both throughput and latency

Delegate response sending and cancellation checks to another thread and wait with GIL released, allowing vLLM Engine to have more CPU time.

vLLM Metrics Access Through Triton

In addition to counter and gauge metrics, Triton now supports **histogram metrics**.

The following vLLM metrics will be supported through Triton with 24.08 and 24.09 releases

Supported vLLM metrics in r24.08

- **vllm:prompt_tokens_total** : counter of prefill tokens processed.
- **vllm:generation_tokens_total** : counter of generation tokens processed.
- **vllm:time_to_first_token_seconds** : histogram of time to first token in seconds.
- **vllm:time_per_output_token_seconds**: histogram of time per output token in seconds.

More in upcoming r24.09

- **vllm:e2e_request_latency_seconds** : histogram of end to end request latency in seconds.
- **vllm:request_prompt_tokens** : histogram of prefill tokens processed.
- **vllm:request_generation_tokens** : histogram of generation tokens processed.
- **vllm:request_params_best_of** : histogram of the best_of request parameter.
- **vllm:request_params_n** : histogram of the n request parameter.

Triton Inference Server Updates

Kubernetes (K8s) Multi-Node

Problem

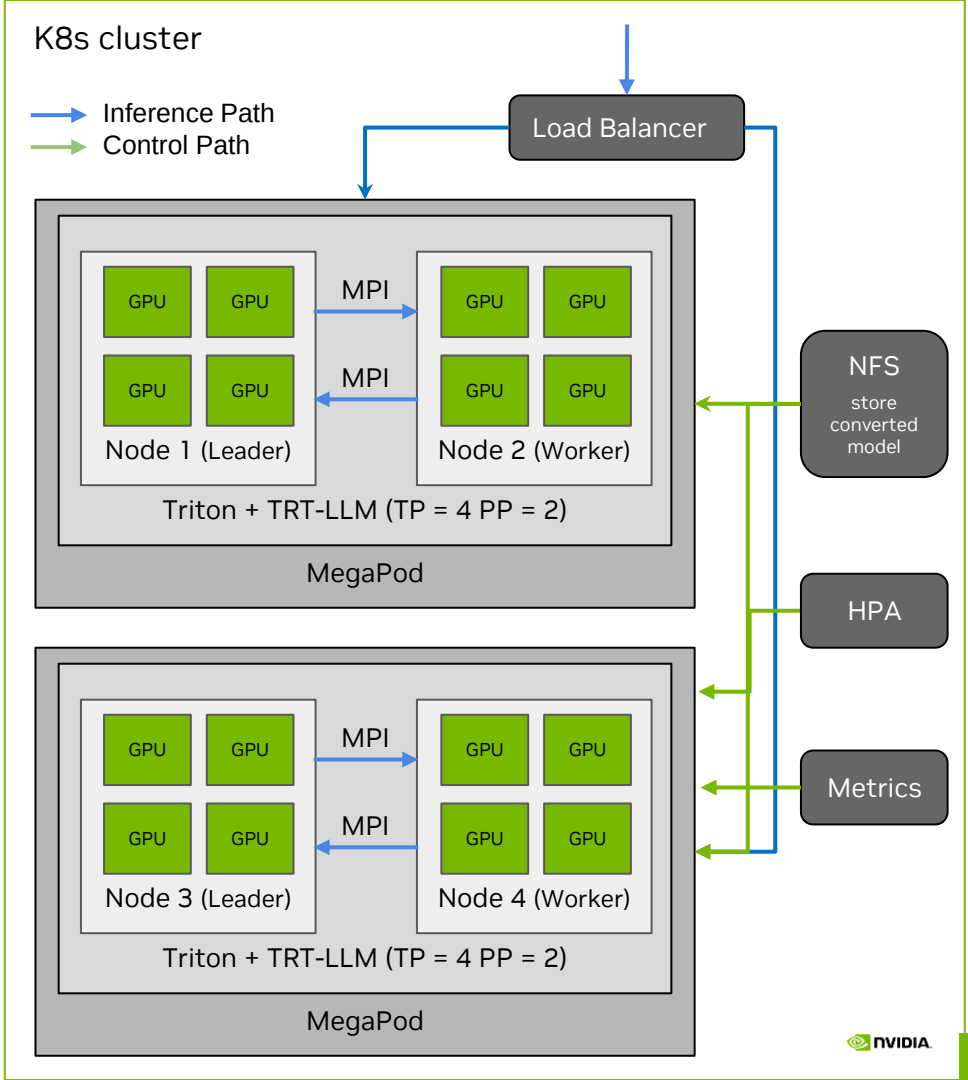
- **Deploy Massive LLMs** (100B - 1T+ parameters) that can't fit on a single node, or make use of older GPUs
- **Automatically Scale and Load Balance** multi-node replicas

Solution

- **Flexible Kubernetes definitions** that can scale any model supported by TRT-LLM on any GPU.

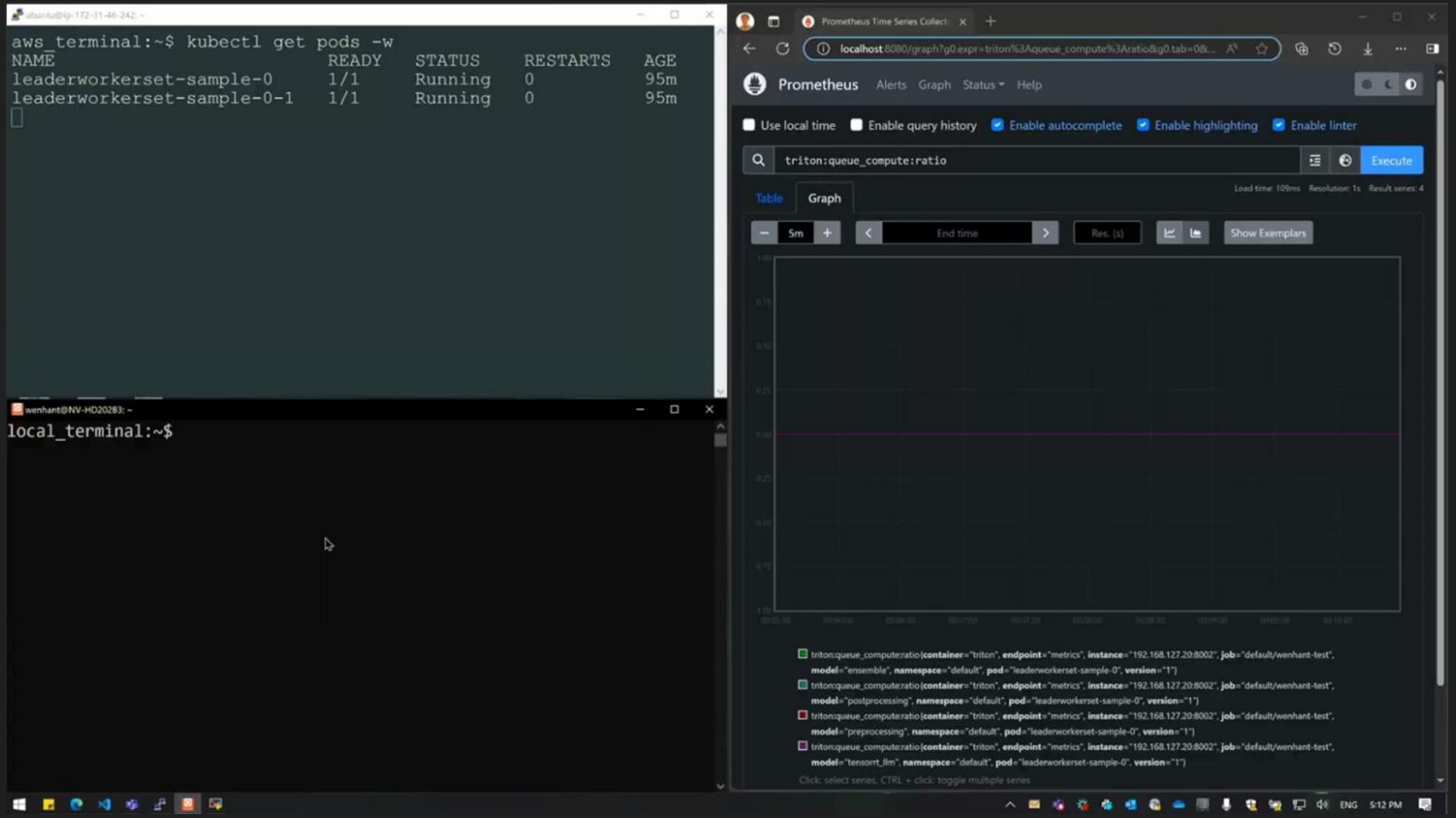
Key Challenges

- **Multi-Node Communication** (MPI)
- **Deploying Groups of Nodes** (Gang Scheduling)
- **Scaling Groups of Nodes** (LeaderWorkerSet)
- **Leader-Aware Load Balancing**
- **Auto-Scaling Metrics**



Kubernetes Multi-Node Demo

Llama 3 70B on 2 nodes of 4x A10Gs

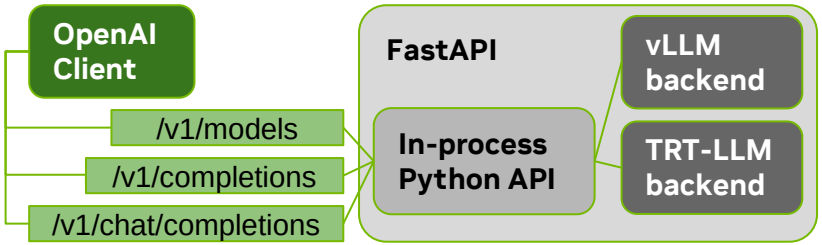


Triton In-Process Python API

- **Python bindings to C APIs** for inference, scheduling, and model management
- **Native Python** support for built-in types like dicts, lists, etc.
- **Non-blocking** inference calls to allow running business logic concurrently
- **Flexible Tensor interoperability** for zero-copy conversions between NumPy, CuPy, PyTorch, DLPack

```python
import tritonserver

# Configure
options = tritonserver.Options(
 model_repository="/path/to/models",
 log_info=True,
 log_warn=True,
 log_error=True,
)

# Serve
server = tritonserver.Server(options)
server.start(wait_until_ready=True)

# Inference: Non-blocking
inputs = {"text_input": ["Machine learning is"],
          "stream": [True]}
model = server.model("llama-3.1-8b-instruct")
responses = model.infer(inputs=inputs)

# Responses are populated as they are received, so other
# work can be done in parallel while awaiting responses.
for response in responses:
 print(response.outputs["text_output"].to_string_array())
```

OpenAI Compatible API (Beta)



- **Customizable** FastAPI app with support for OpenAI schemas
- **Drop-in replacement** for the supported APIs with OpenAI clients, genai-perf, curl, etc.
- **Triton In-Process Python API** used for managing state and handling inference requests
- **Optimized** Triton backends for vLLM and TensorRT-LLM used for the inference runtime

```python
from openai import OpenAI
client = OpenAI(
    base_url="http://localhost:8000/v1",  # Self-hosted
    api_key="empty",
)

completion = client.chat.completions.create(
 model="meta-llama/Meta-Llama-3.1-8B-Instruct",
 messages=[
    {"role": "system", "content": "You are a Triton
Inference Server expert."},
    {"role": "user", "content": "Hello vLLM meetup!"}
 ],
 max_tokens=256
)

print(completion.choices[0].message.content)
```

Constrained Decoding and Function Calling

LLM is a component of an **agentic workflow**

Constrained decoding ensures LLMs to enforce a specific formatting, e.g. JSON

- Constrained Decoding tutorial provides examples for structured generation via prompt engineering and external libraries such as LM Format Enforcer

Function calling enables LLMs to perform complex tasks requiring specific computations or data retrieval

- Function Calling tutorial provides further insights into controlling model behavior

```
# Specify the custom logits post-processor to use
executor_config.logits_post_processor_map = {
   "<name>": custom_logits_processor
}
self.executor = trtllm.Executor(
      model_path=...,
      model_type=...,
      executor_config=executor_config)
…
# At inference time
request.logits_post_processor_name = "<name>"
```

```
python3 client.py --prompt "How's Nvidia doing?" -o 200 --verbose

{
   "step": "1"
   "description": "Get the current stock price for NVIDIA",
   "tool": "get_current_stock_price",
   "arguments": {
       "symbol": "NVDA"
   }
}
=====================================
Executing function:
get_current_stock_price({'symbol':'NVDA'})
Function response: 106.38
=====================================
```

Revamped documentation (24.09)

Triton – TensorRT-LLM Guide

- **Run TRT-LLM models with Triton Server**
- **Advanced Configurations**
- **Deployment Strategies**
- **Tutorials for popular LLM models**

Triton Scaling Guide

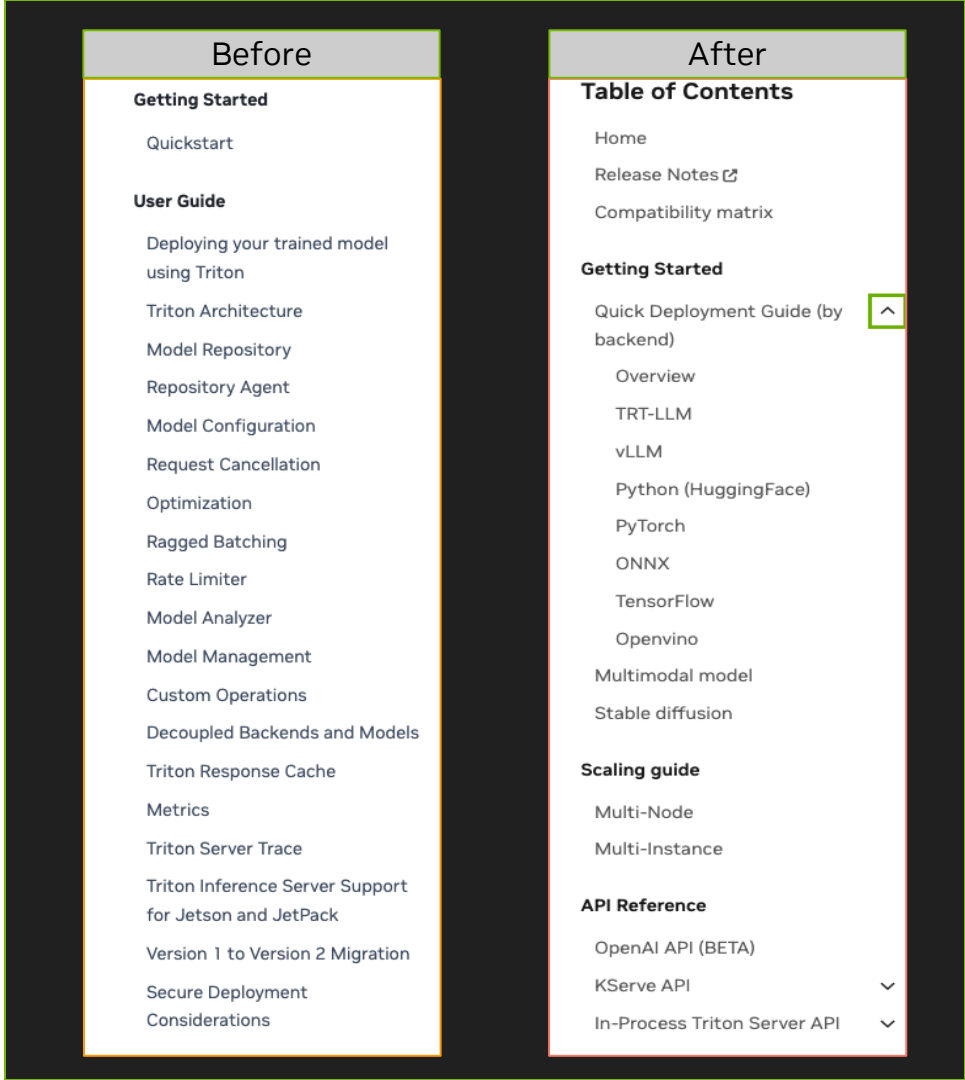
- **Multi-Node**
- **Multi-Instance**

AI Agents Guide

- **Constrained Decoding**
- **Function Calling**

More in upcoming months

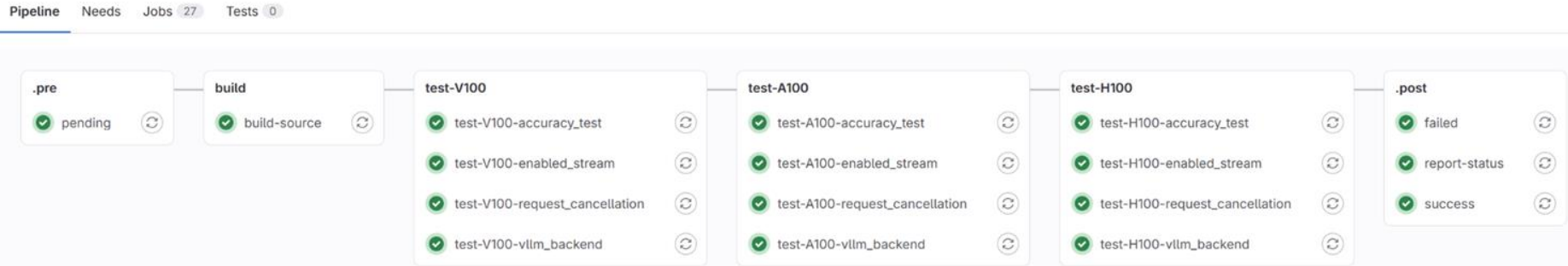
- **Versioning**
- **OpenAI API Guide**



Open Source Improvements and Deliverables

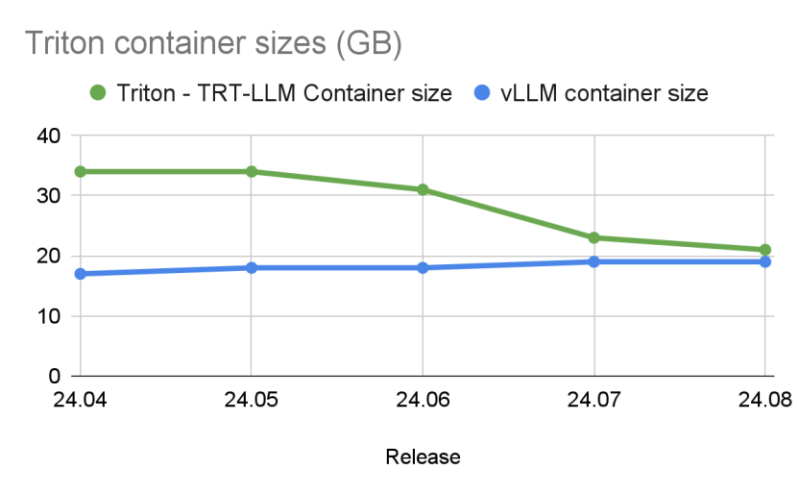
Public CI

- Enable community members to confidently contribute.
- Verify your changes are not breaking
  - Every new vLLM release to be validated against the latest Triton release
  - Every Pull Request on vllm_backend to be validated against the latest vLLM release
  - Provide coverage across state-of-the-art hardware
- Improve transparency of our build process



Container Size Reduction and Manylinux Support

Container size reduction



- **Targeting 15GB**
- **40% container size reduction** from 34GB to 21GB, over the course of last three releases
  - Removed unnecessary installations
  - Optimized our docker build

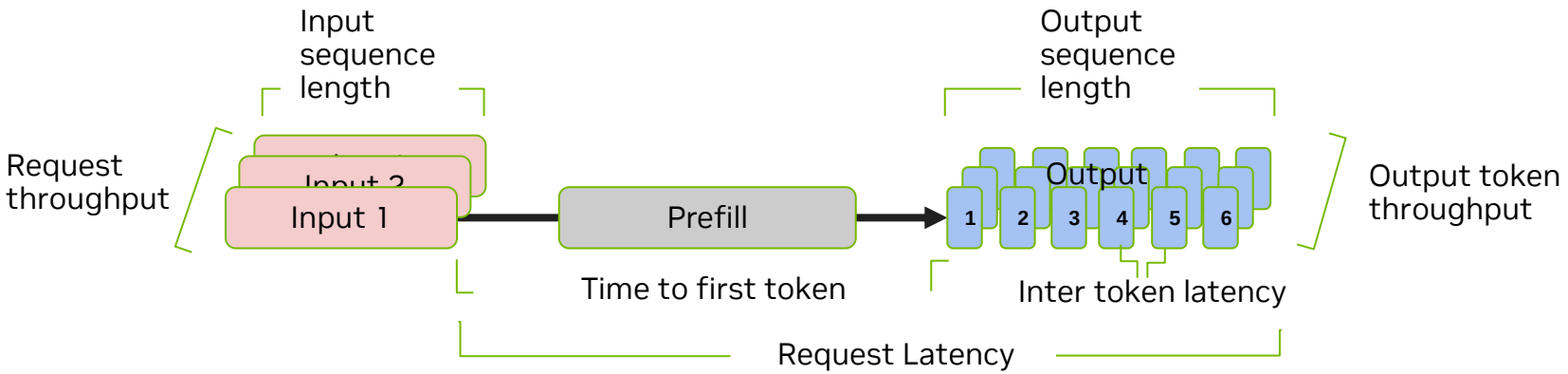
Manylinux

- **vLLM** is coming soon!
- **24.11 GA**: Targeting complete feature parity with our NGC containers
- **Early Access** release available upon request
  - Supported backends
    - PyTorch
    - TensorFlow2
    - ONNX Runtime
  - Supported HW
    - X86 CPU
    - ARM64 CPU
    - GPU
  - Triton server core

GenAI Perf

GenAI Perf

Enables apples to apples comparison for GenAI perf evaluation



Key features

- HW/SW agnostic performance benchmarking via **KServe and OpenAI API**
- Supports LLM, Visual Language Model, embedding, re-ranking, and multi-LoRA
- In-process benchmarking for TRT-LLM

Planned features

- **Customizable front-end** to allow custom benchmarking logic to easily extend GenAI-Perf.
- Access to **raw metric data** for easy post processing and **scripts for generating plots**.

GenAI-Perf Usage and Output

Three Input Formats

- **Synthetic**
- **Bring Your Own Data (File)**
- **External Dataset**

Server-agnostic and model-agnostic

Available via Docker and local build

Example command

```
genai-perf profile \
 -m gpt2 \
 --service-kind openai \
 --endpoint-type completions \
 --synthetic-input-tokens-mean 200 \
 --synthetic-input-tokens-stddev 50 \
 --streaming \
 --output-tokens-mean 100 \
 --output-tokens-stddev 25
```

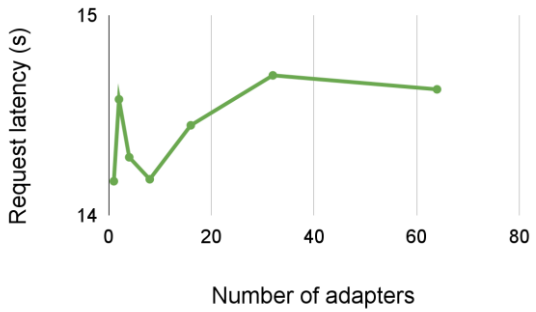
Example output

NVIDIA GenAI-Perf | LLM Metrics

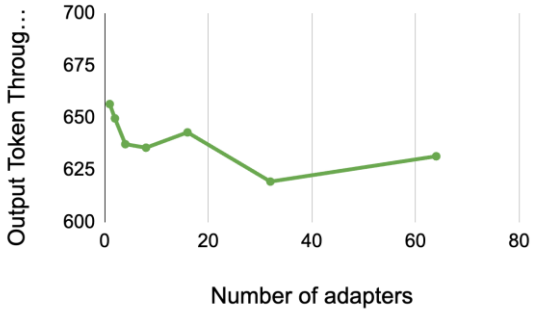
Statistic	avg	min	max	p99	p90	p75
Time to first token (ms)	30.82	18.76	59.07	58.41	52.63	29.02
Inter token latency (ms)	5.29	2.98	15.75	15.71	13.68	4.05
Request latency (ms)	647.73	183.16	2,344.49	2,265.43	1,563.13	582.00
Output sequence length	114.78	49.00	184.00	174.82	146.60	138.50
Input sequence length	206.16	127.00	293.00	287.60	261.20	238.00
Output token throughput (per sec)	176.97	N/A	N/A	N/A	N/A	N/A
Request throughput (per sec)	1.54	N/A	N/A	N/A	N/A	N/A

GenAI-Perf Benchmarks Multi-LoRA Models

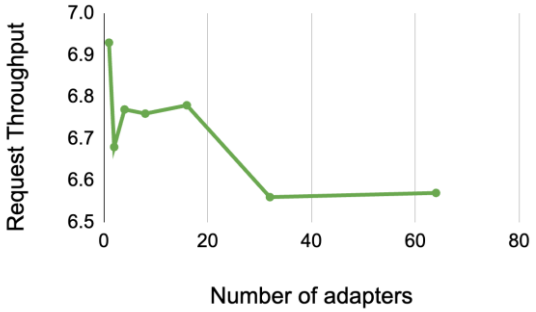
Insight 1 : Stable latency



Insight 2: Stable token throughput



Insight 3 : Lower throughput at >16 adapters



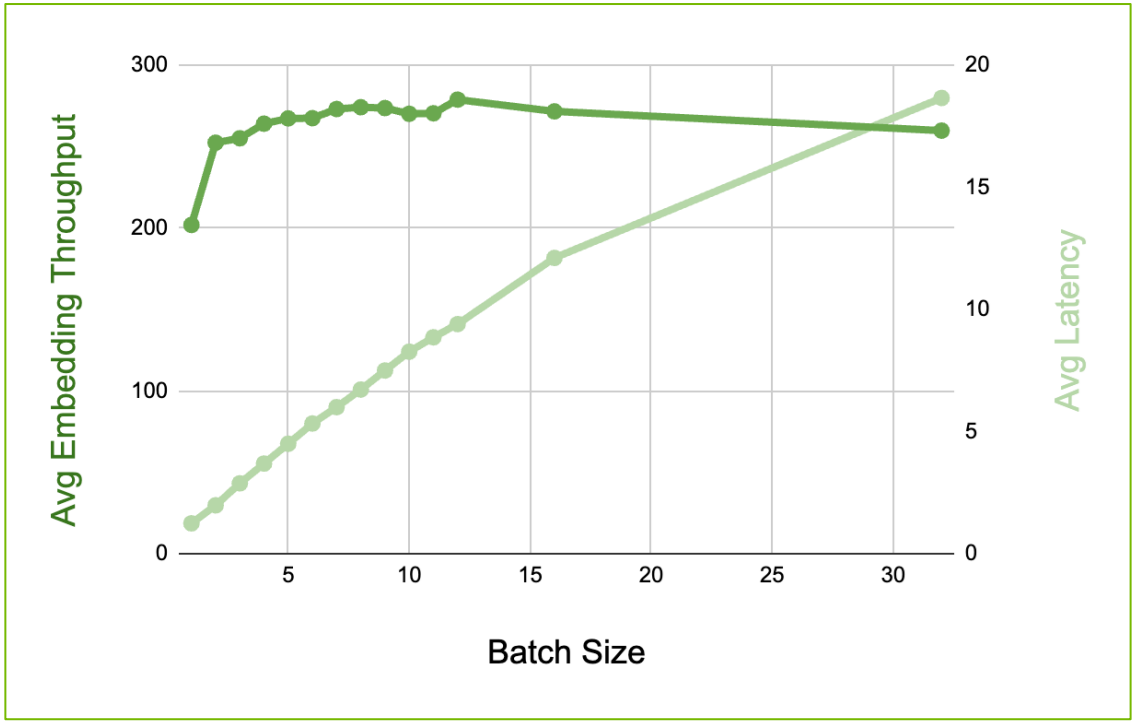
How many adapters can we deploy without compromising performance?

We can serve up to 64 adapters on one server. For more adapters, we can use GenAI-Perf for further benchmarking.

Benchmarking Setup
Server: Mistral-7B-v0.1 loaded on a single TGI server running on an NVIDIA RTX 5880
GenAI-Perf: concurrency of 128, random model selection, 500 synthetic prompts
Key arguments to GenAI-Perf:

- "-m <adapter_name> <adapter_2_name> ... <adapter_n_name>"
- "--model-selection-strategy random"

GenAI-Perf Benchmarks OpenAI API Embeddings Models

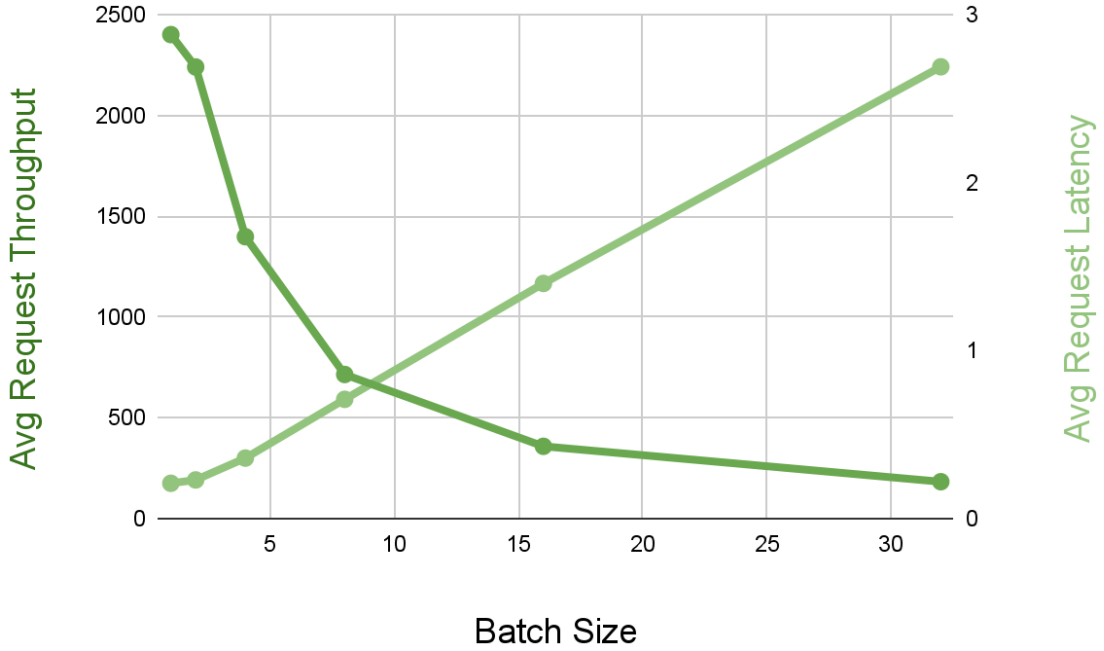


At what batch size do we maximize our embedding throughput?

Embedding throughput is maximized at a batch size of 12. Latency increases continually.

Benchmarking Setup
Server: intfloat/e5-mistral-7b-instruct model loaded on a single vLLM OpenAI server running on an NVIDIA RTX 5880
GenAI-Perf: concurrency of 256

GenAI-Perf Benchmarks Hugging Face Re-Ranker API Models



What is the impact of increasing the number of passages per re-ranking request?

The number of passages per request significantly impacts latency and throughput.

Benchmarking Setup
Server: BAAI/bge-reranker-large model loaded on a single Hugging Face TEI server running on an NVIDIA RTX 5880
GenAI-Perf: concurrency of 512

Vision Language Model

Rising popularity of Vision Language Models (VLM) over the past year

- **OpenAI multi-modal chat API** for VLM inference
  - contains text and image (base64)

- Supports 2 input data sources:
  - **Synthetic data generation**
    - Image resolution (height / width)
    - Image format
    - Prompt length (or, input sequence length)
  - **Bring your own data (BYOD)**

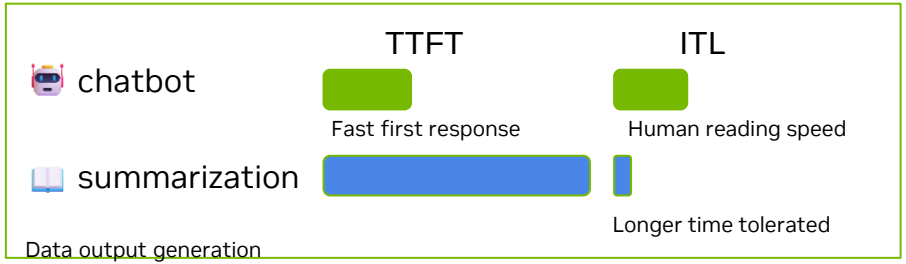
```
payload = {
   "model": "llava-hf/llava-v1.6-mistral-7b-hf",
   "messages": [
     {
       "role": "user",
       "content": [
         {
           "type": "text",
           "text": "What's in this image?",
         },
         {
           "type": "image_url",
           "image_url": {
             "url":
f"{base64_image}"
           }
         }
       ]
     }
   ],
   "max_tokens": 100
 }
```

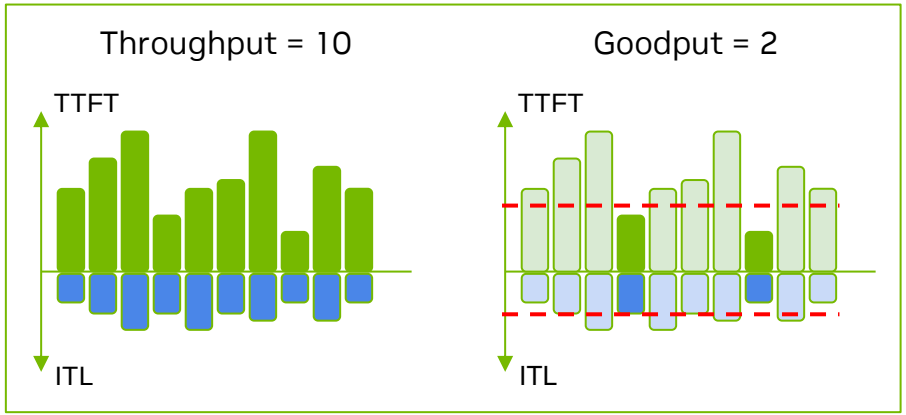
[1] OpenAI Vision: https://platform.openai.com/docs/guides/vision

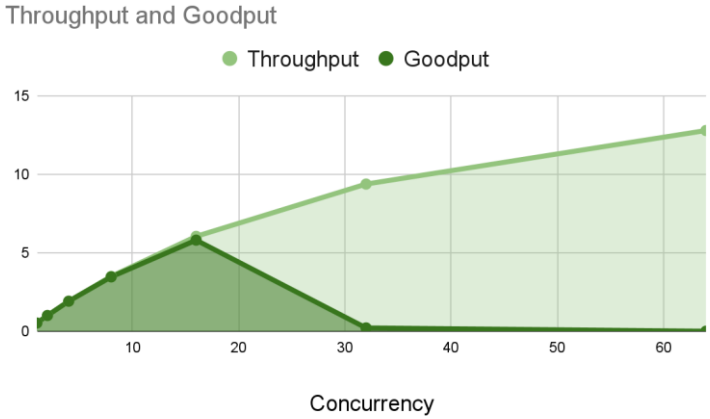
Goodput

Benchmarking GenAI performance from user's perspective

Goodput is defined as the number of completed requests per second that meet certain user-defined service level objectives (SLO) such as time to first token (TTFT) and inter token latency (ITL)







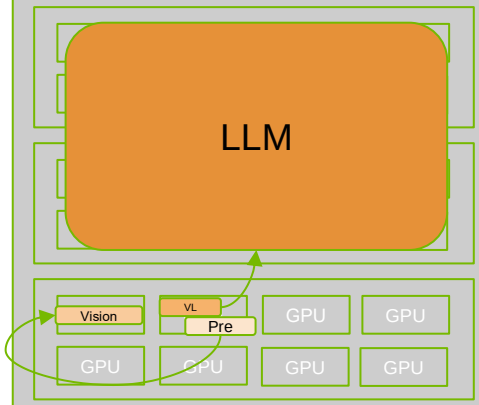
Benchmark Env:
Server: Triton + vLLM on an NVIDIA RTX 5880
Model: Llama-3.1-8B
Goodput Constraints: TTFT ≤ 250ms and ITL ≤ 25ms
ISL/OSL: 300/100
Concurrency: [1, 2, 4, 8, 16, 32, 64]

Slides courtesy of "DistServe: Distributed Serving for Large Language Models" Hao AI Lab Blog, hao-ai-lab.github.io/blogs/distserve

Triton 3.0
Disaggregated Serving

GenAI Models and Workloads Require Data Center Scaling

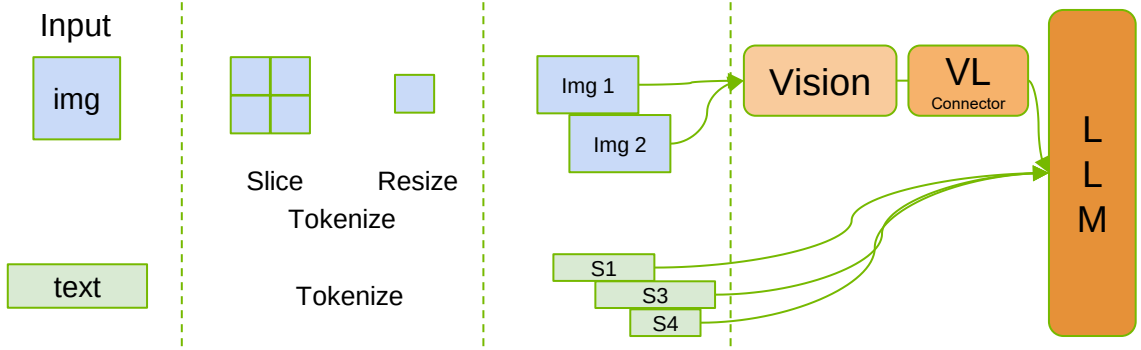
Data center scale



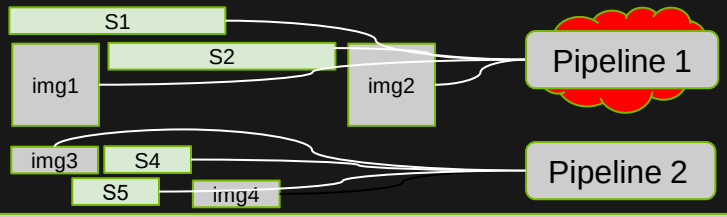
Hetero multistage pipeline

- Multi-node & out of process
- Independent scaling
- Pull (capacity) based load balancing

Simple visual language model (loosely Llava 1.5 HD) example



Bottlenecks & severe under utilization in pipeline (Adobe Firefly, Linkedin)

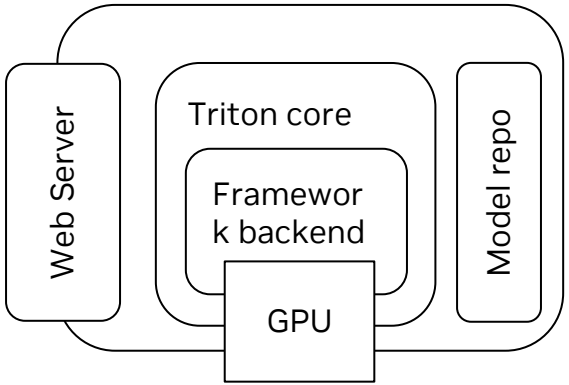


No longer can duplicate pipeline or statically define (push based) load balance

Triton Inference Server

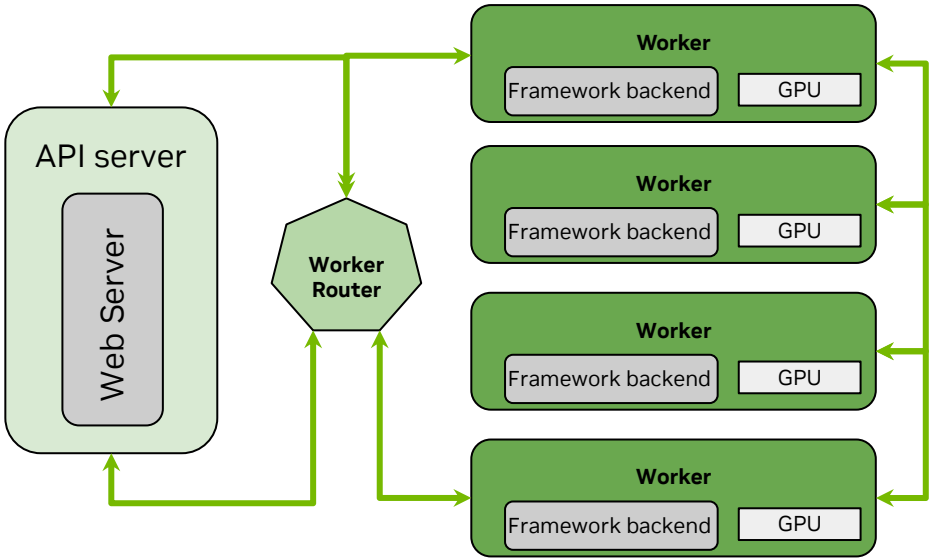
Optimized for Data Center Scale Distributed Workflows

Triton 2.0



- **Optimized for Single Node Performance**
- **Optimized for Web Service Dev Cycle: Develop, Optimize, Deploy**

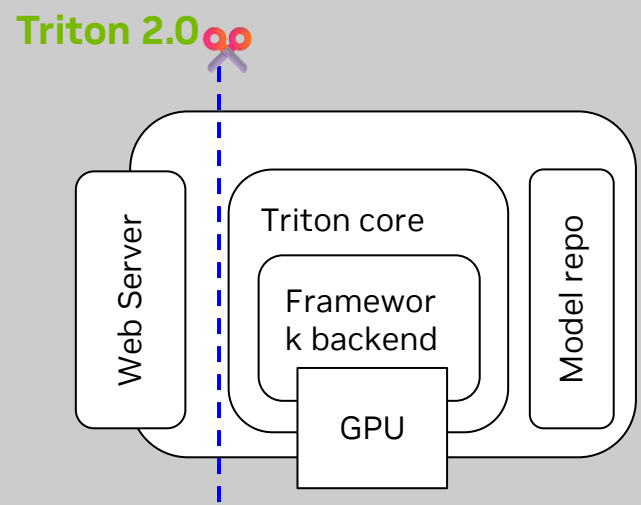
Triton 3.0



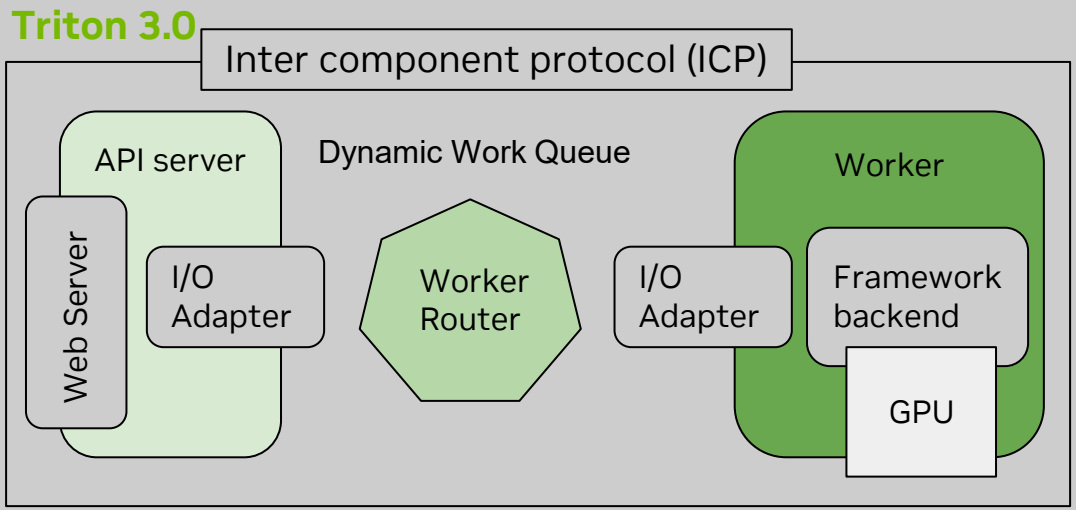
- **Optimized for Gen AI**
- **Optimized for Data Center Scalability**
- **Optimized for Distributed Workflows**
- **Optimized for Quick Iteration**

Triton 3.0 Component Architecture

Split Triton Into Reusable Components



- Just enabled K8 multi-node autoscaling
- But needs new foundation to improve flexible scalability

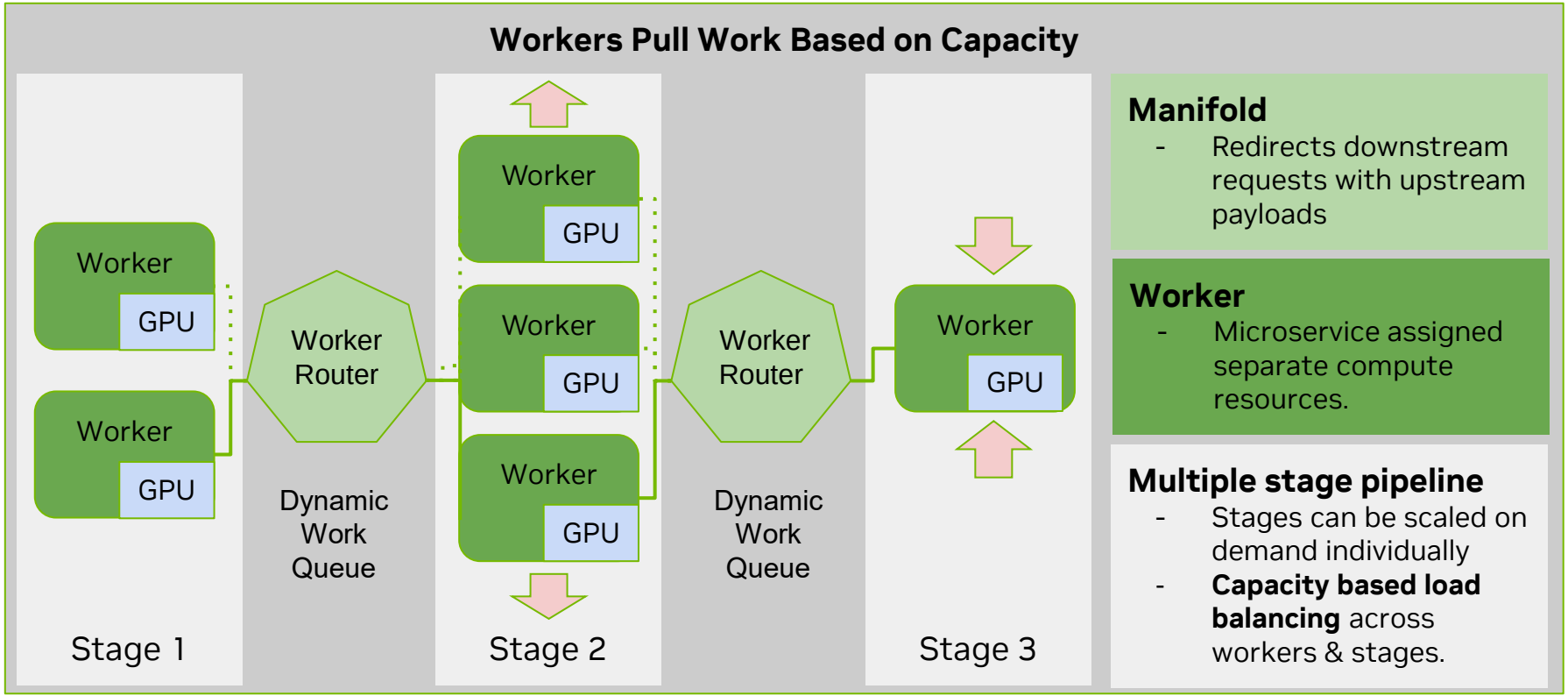


Interchangeable API Server (NIM, RayServe, etc)

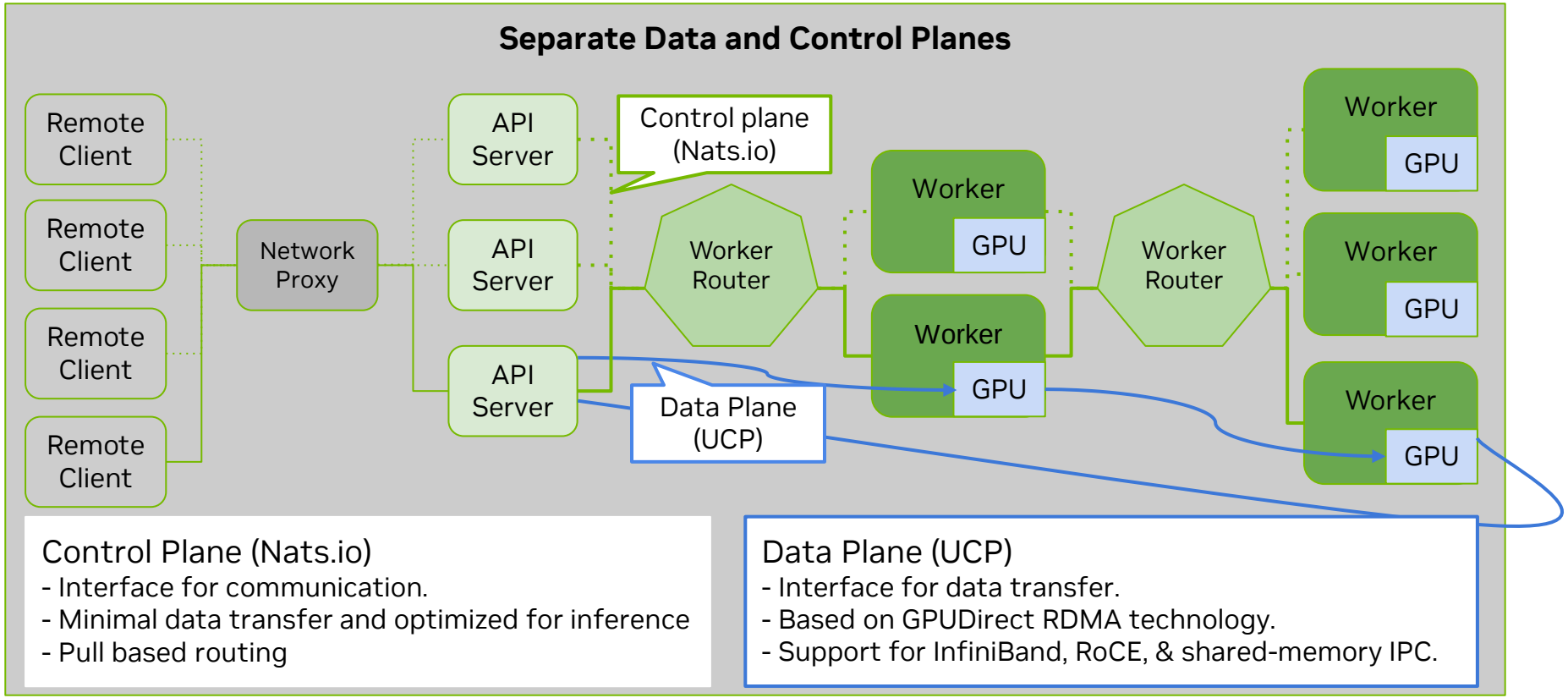
Coordinate data to workers w.r.t capacity

Create/destroy on demand
Scale up for throughput
Single model or ensemble.

Triton 3.0 Components

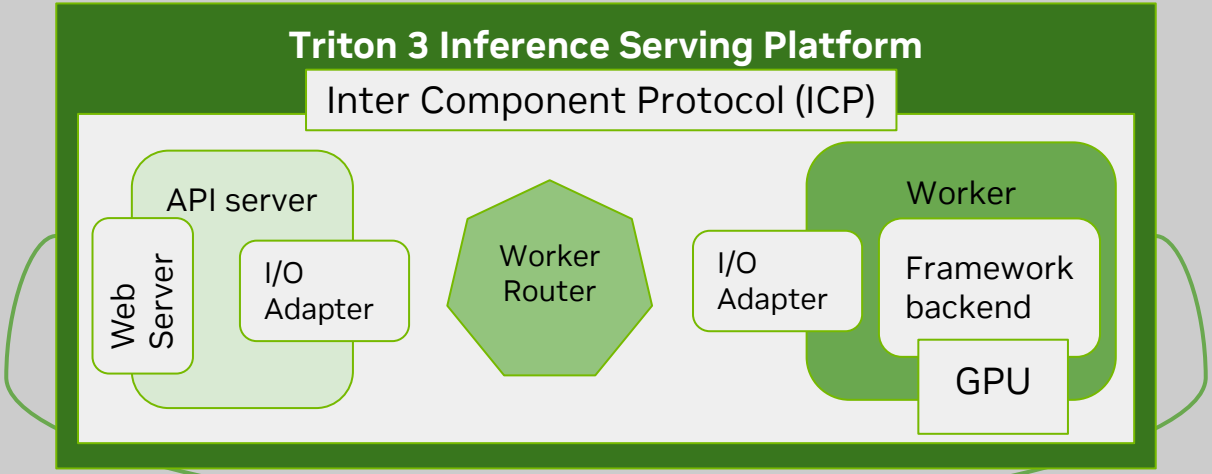


Triton 3.0 Inter Component Protocol



Triton 3.0 Component Reuse

Components can be adopted in modular fashion and will have first class Python support.

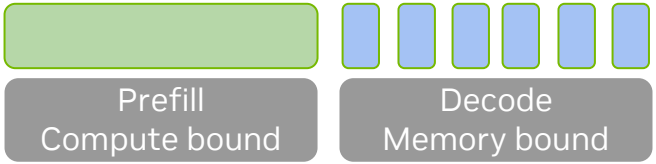


Domain Specific APIs:
In addition to inference APIs, include LlamaStack, SGLang, LangGraph, etc

KV cache Transfer & Model Management
- Disaggregated serving
- Domain specific KV cache routing

Perf Benchmarking & Tuning
- GenAI Perf & Model navigator
- Worker to worker traffic

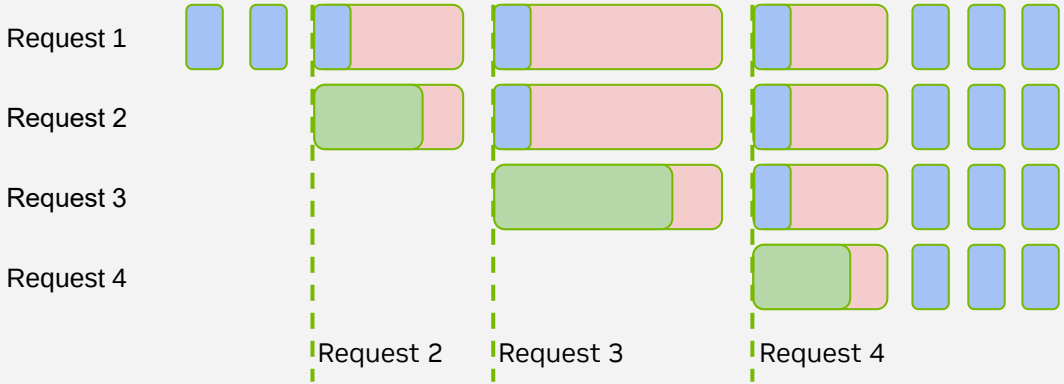
Use Case: Disaggregated Serving



Colocating prefill & decoding causes interference

Lost Generation

4 requests processed in single GPU with continuous batching



Colocating prefill & decoding couples resource allocation and parallelism

- Different optimal strategies for prefill & decode
- **Prefill:** tensor parallelism for low latency
- **Decode:** data or pipeline parallelism for high throughput

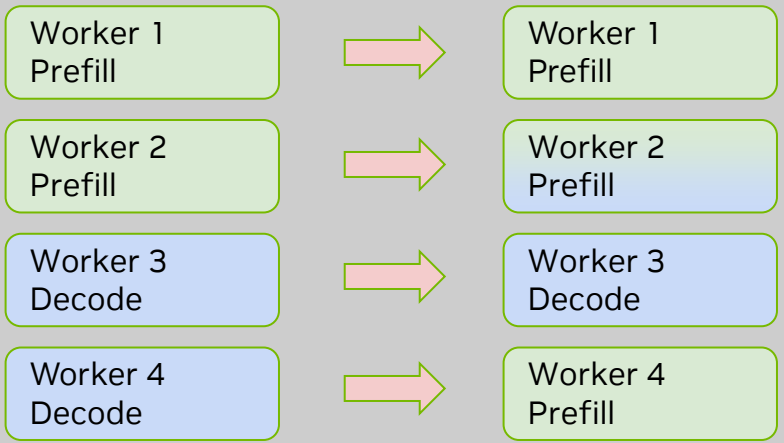
Disaggregate prefill and decode

Leverage high BW NVLink to minimize the KV cache transfer overhead

Use Case: Disaggregated Serving

Dynamic Disaggregated Serving

Dynamically change prefill and decode based on ISL/OSL or TTFT/ITL requirements



Disaggregated Serving on Mixed SKUs

- Simulated results on H100 (prefill) and H20 (decode) show cost benefits

Disaggregated Mixture of Experts (1.8B)

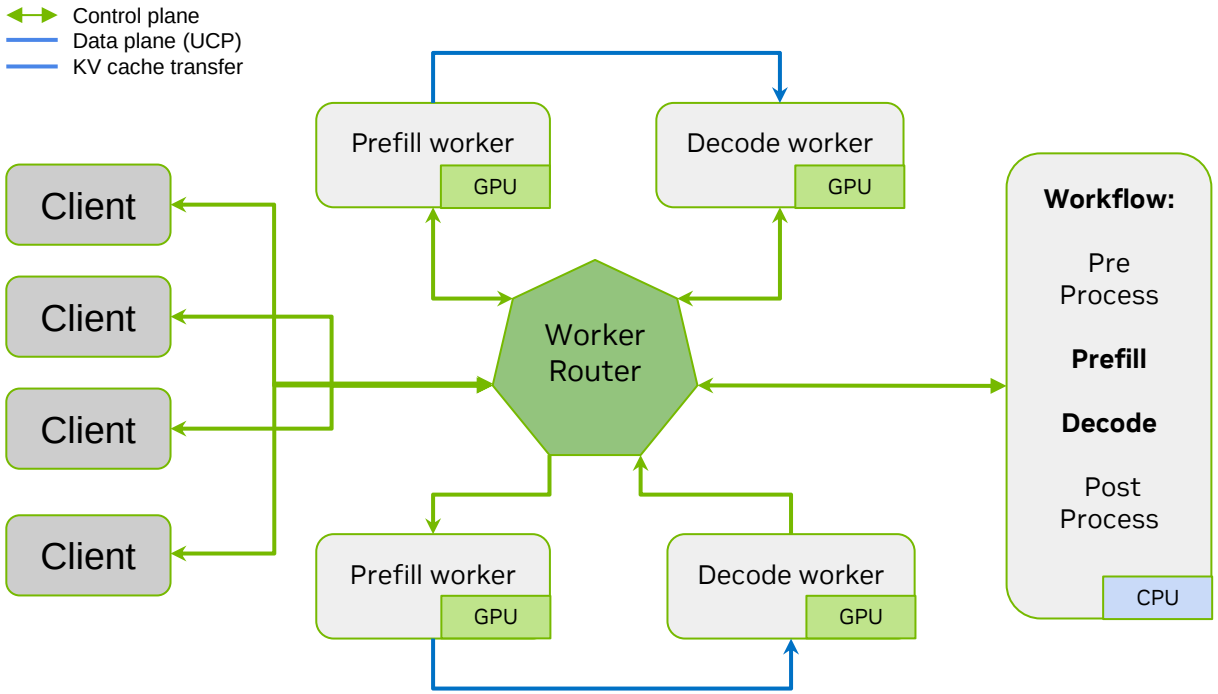
- Can make bigger models more performant

NVLink/NVSwitch => High BW KV cache transfer across workers & nodes

- Enables easier programmability for placing disaggregation.

Better utilization of one gigantic GPU system (Blackwell)

Disaggregated Serving Prototype



TRT-LLM

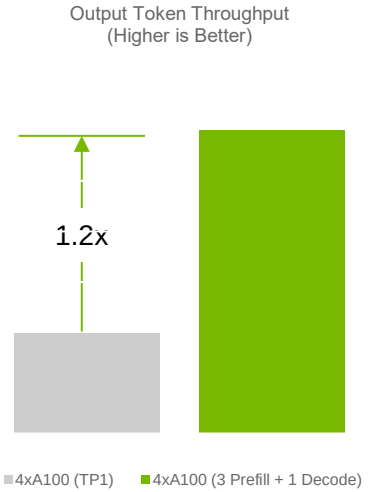
- Llama 3 8b Instruct
- TP 1
- 4x A100 GPUs
- Pull Based Routing

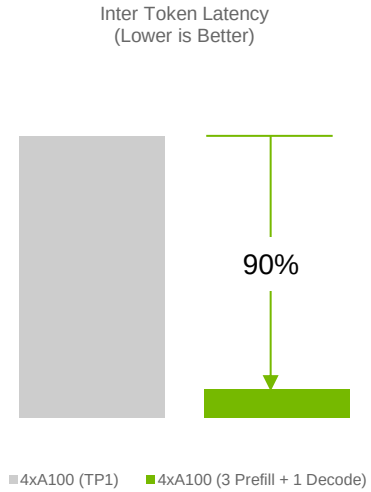
vLLM

- Llama 3.1 70B Instruct
- TP 2
- 8x H200 GPUs
- Pull Based Routing

Disaggregated Serving Prototype - Triton 3 with TRT-LLM backend

Llama3 8B, ISL/OSL: 4096/512, TRT-LLM 0.12, Concurrency:10



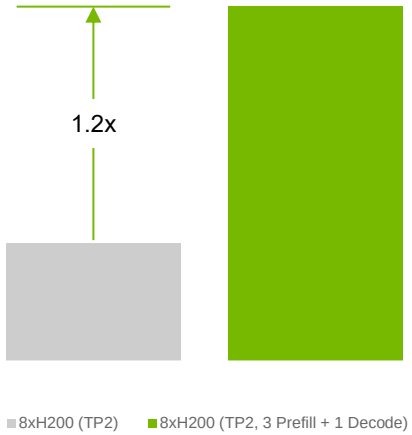


Preliminary results. Baseline is default (non-optimized settings). Results for Disaggregated Workflow based on non-streaming TTFT (ms) 4A100 (TP1):1x, 4A100 (3 Prefill + 1 Decode) = 2.5x

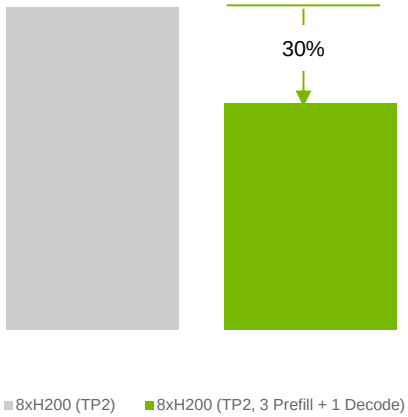
Disaggregated Serving Prototype - Triton 3 with vLLM Backend

Llama3.1-70B, vLLM 0.5.3, 8x H200, ISL/OSL: 2K/128, Concurrency:10

Output Token Throughput
(Higher is Better)



Inter Token Latency
(Lower is Better)



Preliminary results. Baseline is default (non-optimized settings). Results for Disaggregated Workflow based on non-streaming TTFT: 8xH200 (TP2):1x, 8xH200 (3 Prefill + 1 Decode) = 1.4x

NVIDIA

Thank You

Getting started with NVIDIA AI Inference

Bring inference to production with performance, ease, and cost savings

NVDIA Triton Product Page

NVIDIA Launch Pad

Download Triton on NGC

Explore More Resources for Development

For more information on NVIDIA Triton Inference Server

Open-source GitHub repository:

Latest release information

Quick start guide:

NVIDIA Deep Learning Institute. Self-paced online course. 4 hours.
Deploying a Model for Inference at Production Scale