# Sizing LLM inference systems

How many GPUs do you need for inference?

# About Us

- Senior Deep Learning Solutions Architect @ NVIDIA - Supporting deployment of AI / Deep Learning solutions
- Focusing on large scale efficient deployment and inference

**Dmitry Mironov, EMEA**

- Senior Deep Learning Solutions Architect @ NVIDIA - Supporting delivery of AI / Deep Learning solutions
- Focusing on quantization in training and inference

**Sergio Perez, EMEA**

NVIDIA.

# Agenda

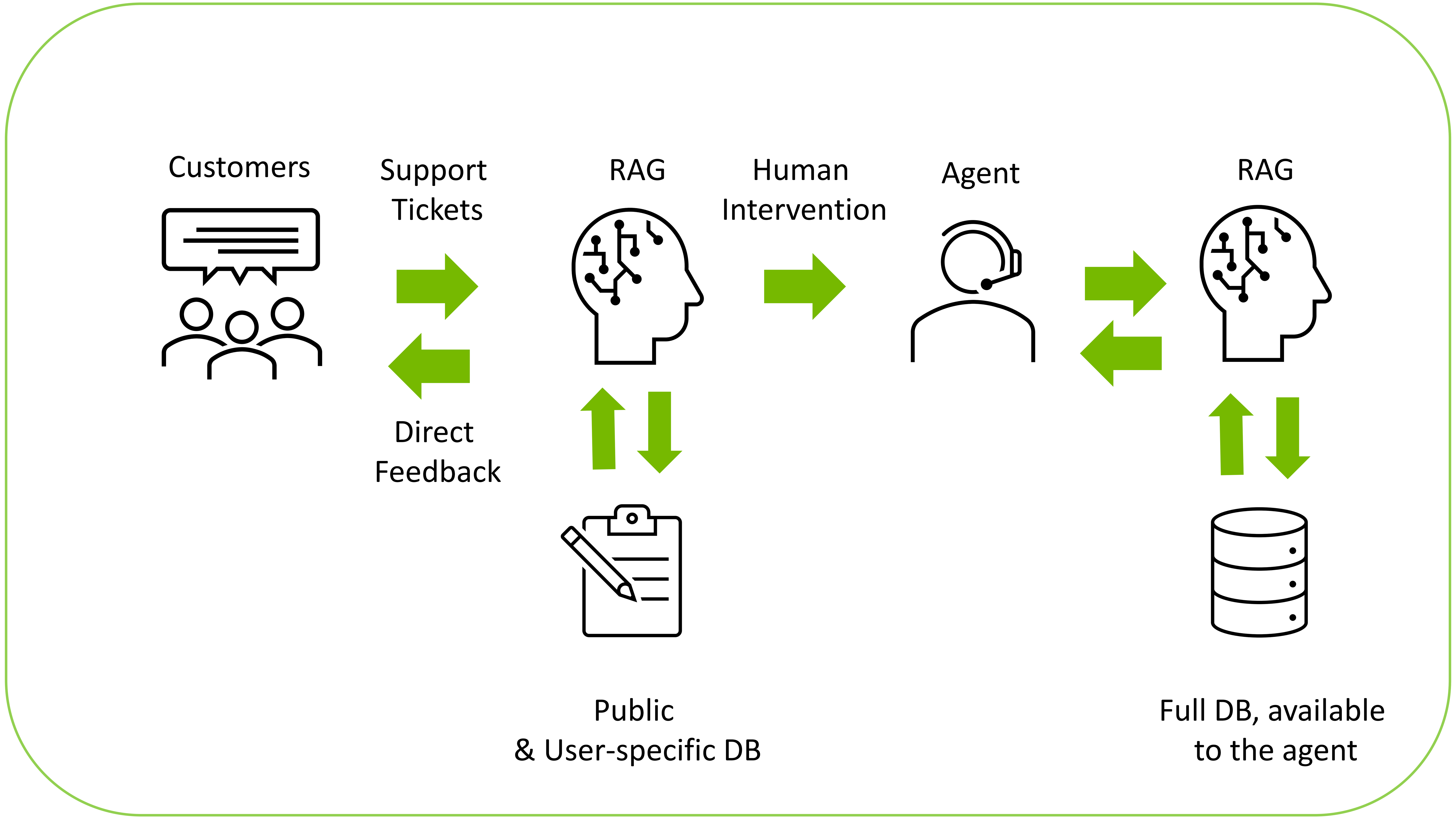
Sizing for Inference can get a bit complicated

NVIDIA SW Stack for inference

Short summary of how to think about a problem

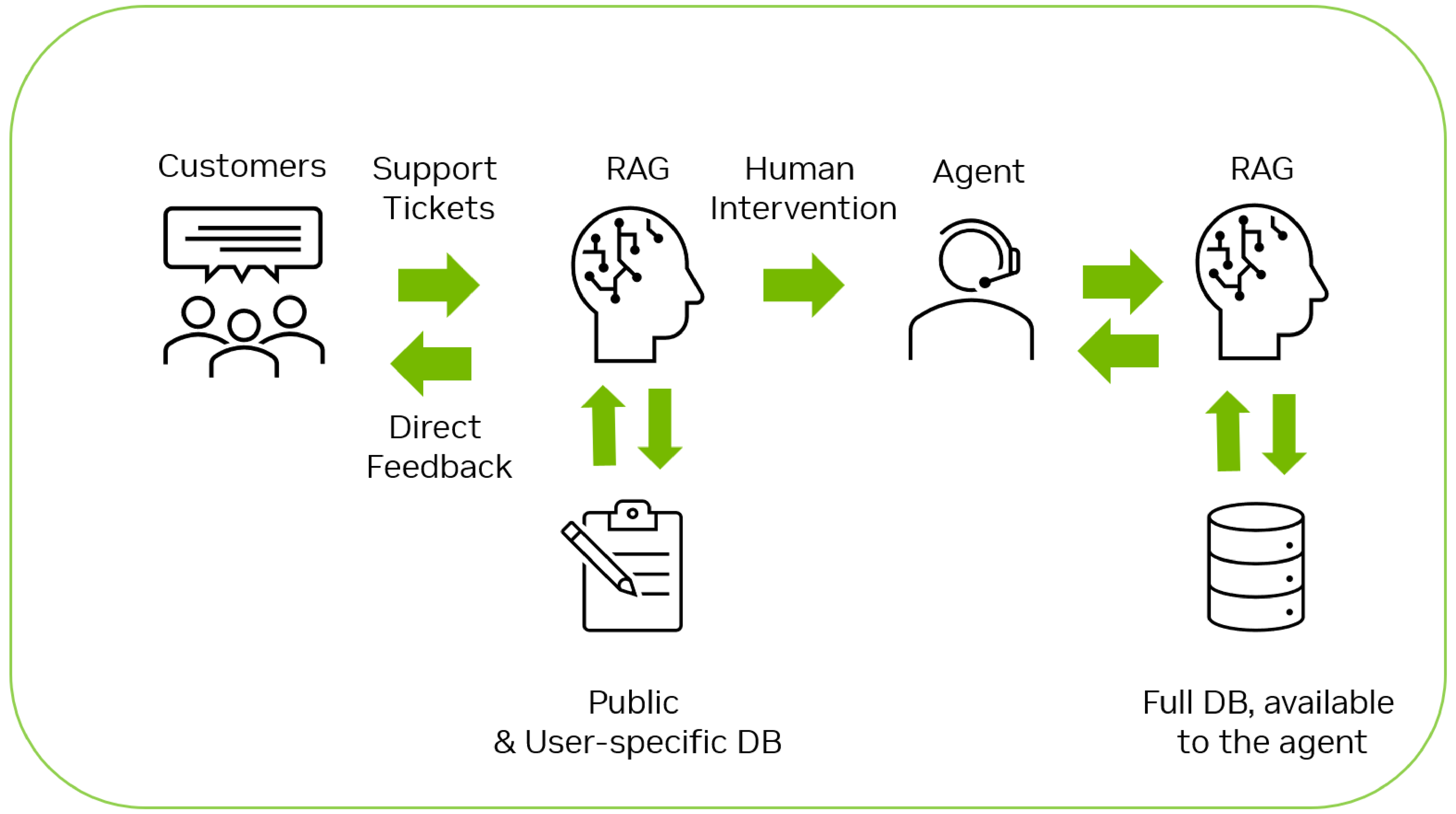
# Customer Use Case Example

## Challenges of sizing




# Customer Use Case Example

## Challenges of sizing




**How many systems do we need to buy for this?**

Gather requirements

- 3500 words in, 500 words out
- NeMo 43B GPT
- First token latency limit 3s
- Max 31 requests (=prompts) per second

**Inference sizing**

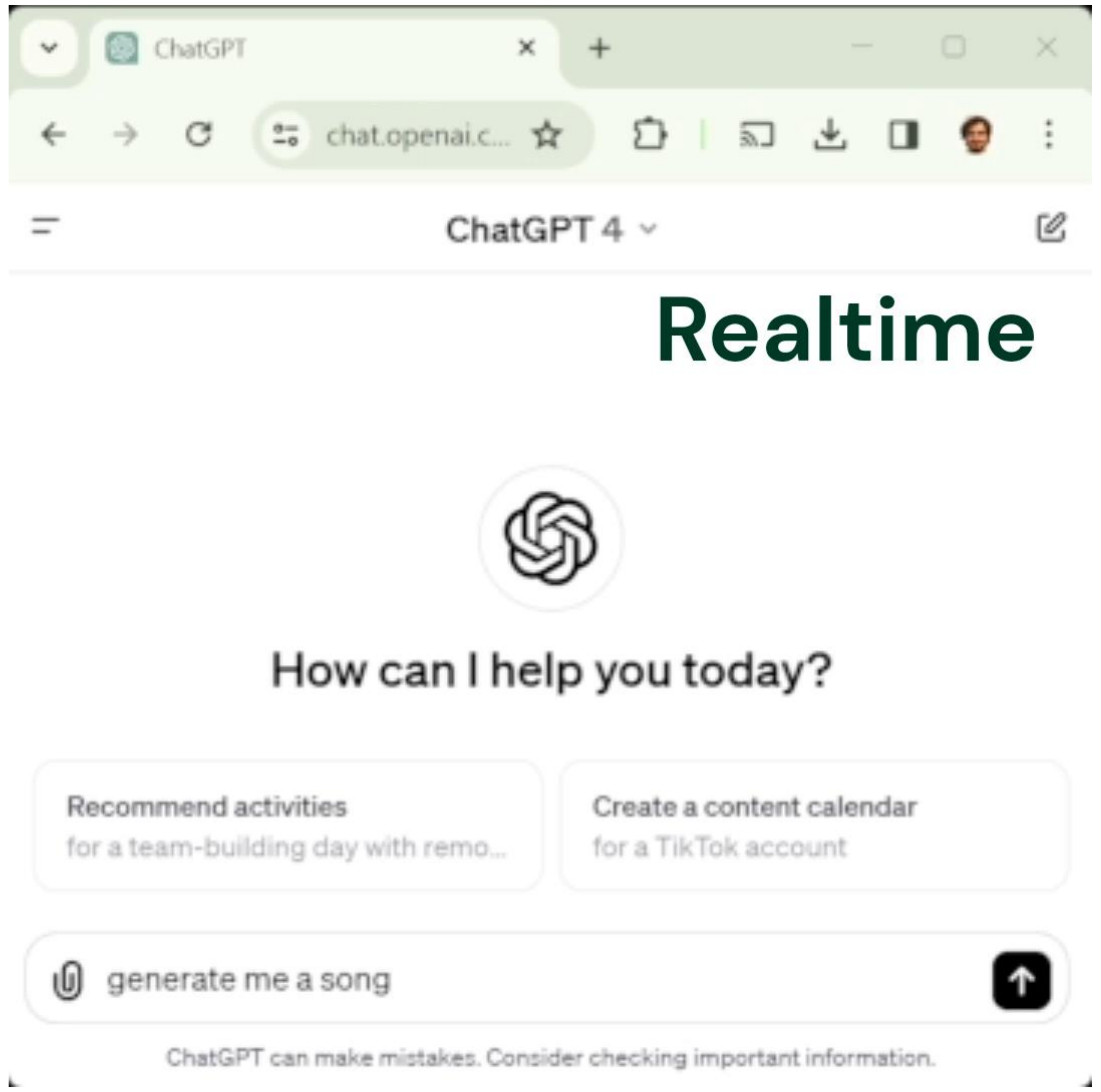
**The customer needs 13 DGX H100 systems**

- Throughput: 2.4 requests per second
- First token latency 2606 ms (prefill) is within the limit specified
  Inter-token latency 21.4 ms/generated token
- Generation latency of 500 tokens = 21.4 * 500 = 10 700 ms = 10.7 s

# 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 sp`

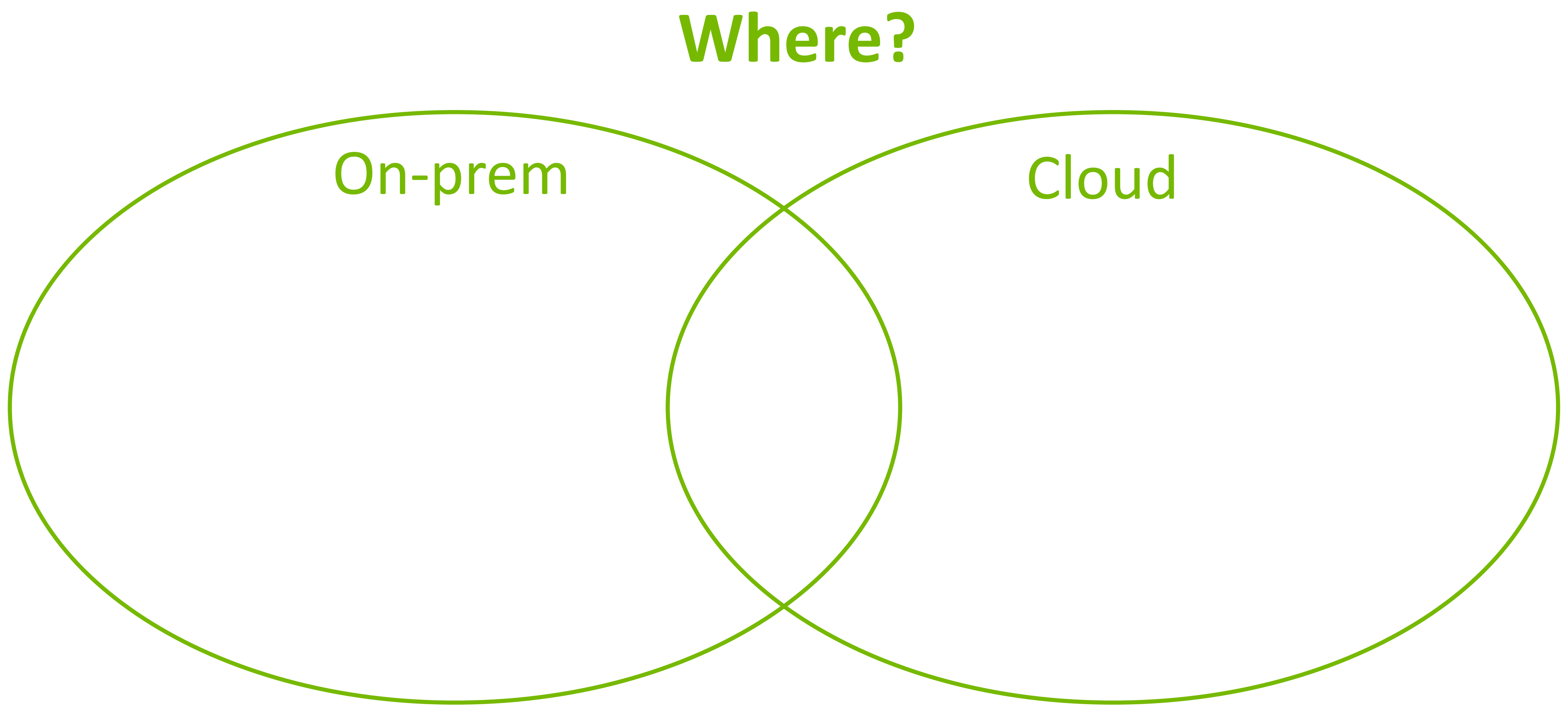
| Prefill, first-response | Decoding | | | | | | |
|---|---|---|---|---|---|---|---|
| User: generate me a song. AI: *Could* | you | please | generate | text | about | GPUs | that |

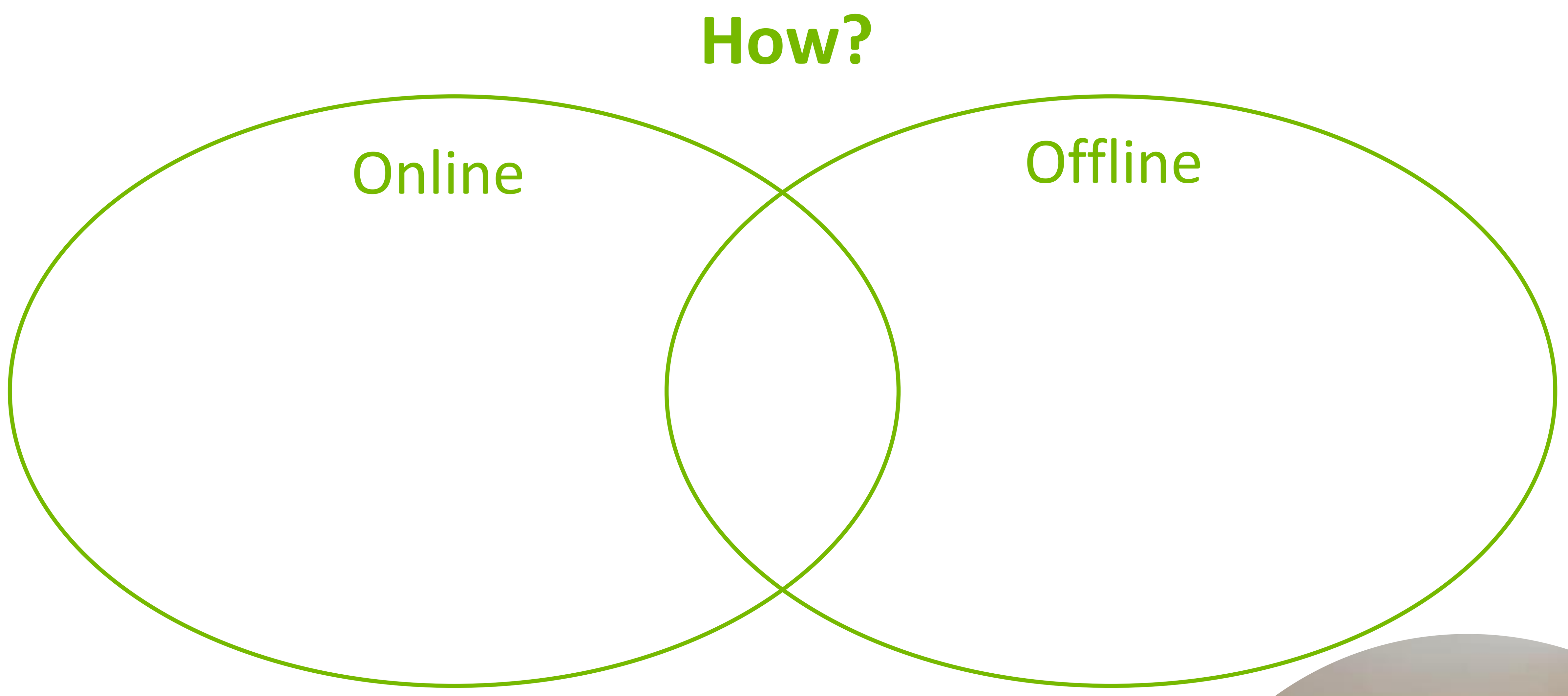
prompt

user prompt

# The Two Things To Care About

Where and how do we execute inference?




# Where?

Significant impact of deployment location

## On-prem

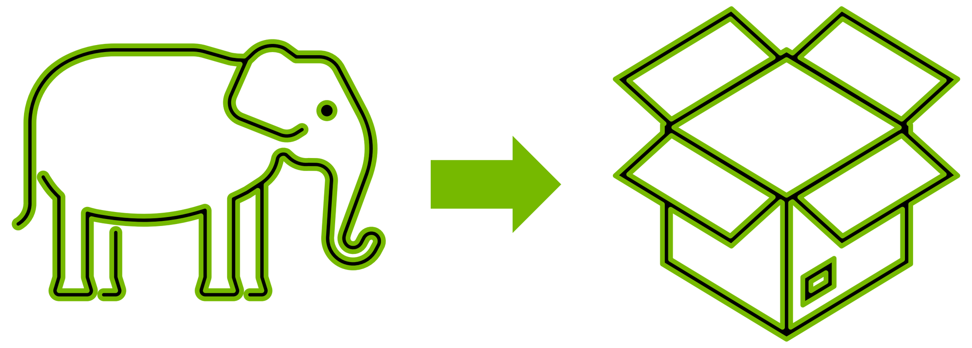
- Fixed Capacity: you need to understand the size for the maximum simultaneous load
- Pricing model: per peak capacity

## Cloud

- Variable Capacity: APIs hide capacity concerns – in reality, similar limitations apply (GPU shortage)
- Pricing model: per token

## On-prem and Cloud

- Minimal Capacity + Autoscaling for bursts



# How?

## Significant impact of inference strategy

### Online — live generation

- Complexity: it matters to people how quickly they will get their response
- Imposing latency requirement significantly decreases available throughput. Need to balance between throughput and latency

### Offline — postponed computation

- Simplest execution model
- Throughput, throughput, throughput: maximum GPU utilization, maximum batch size

**Fun fact:** Fast human reading speed is 90 ms/token (=500 words/minute at 0.75 tokens/word) (avg is 200 ms/token)

# Online Streaming vs Sequential

## Two facets of latency

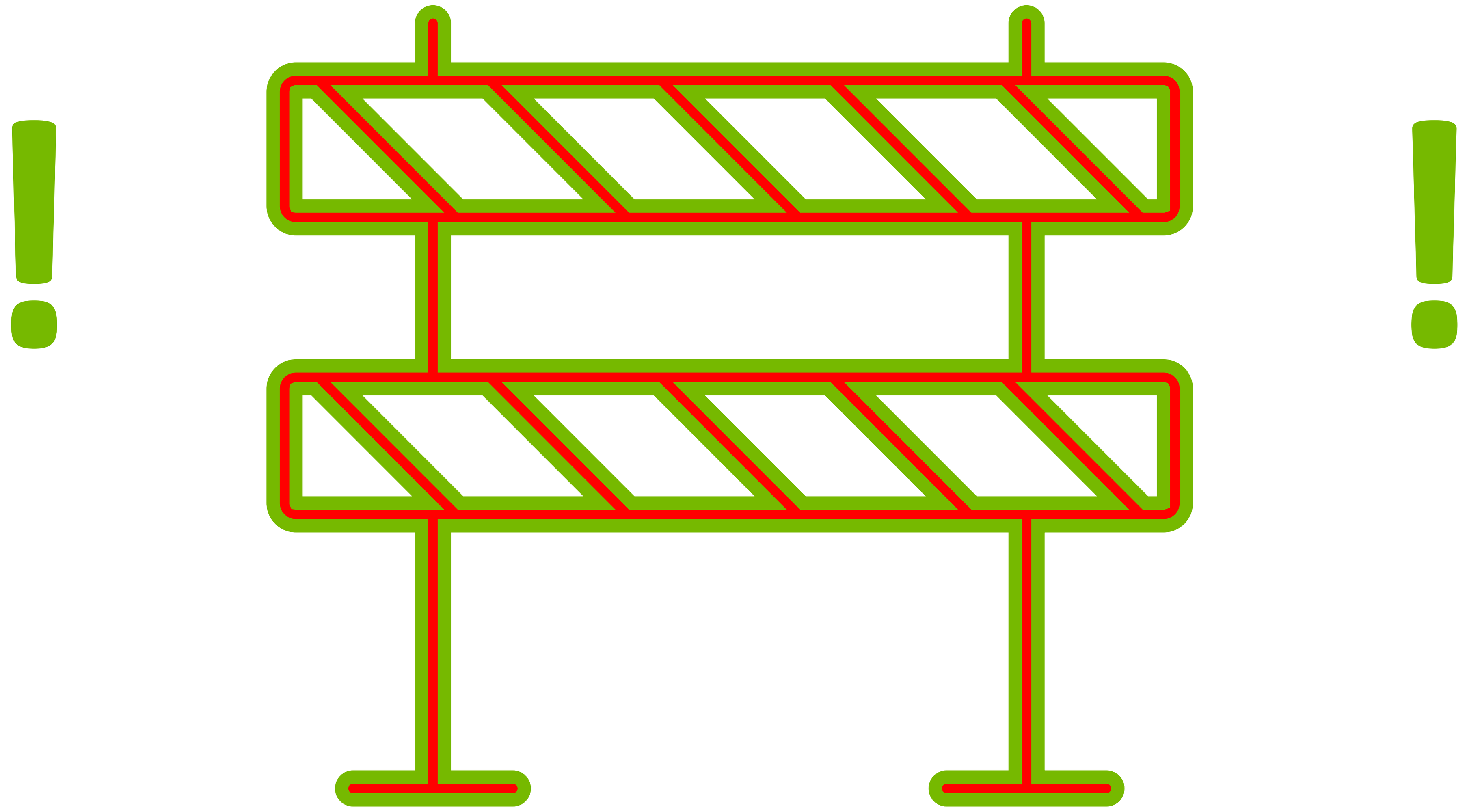
- **Streaming**: one token at a time
  - In this situation only the **TIME-TO-FIRST-TOKEN** matters (as we generate text faster than people can read)
  - One needs to develop the app streaming capabilities
  - Simpler to satisfy real-time latency requirements
  - Can be implemented only in the last step of the pipeline

- **Sequential:** waits for the full response
  - Say you want to check whether the user question is not toxic BEFORE you start answering
  - In this case **END-TO-END** latency/time to last token matters
  - Legacy apps can be simply updated with sequential mode
  - Latency requirements are too restricting for throughput

# Questions to start sizing for inference

# Questions for a Sizing Use Case

1. ✅ What model are you planning to use?
2. ✅ What is the average number of tokens in the prompt to your LLM (Length of input)?
   - For English one token is approximately 0.75 of a word.
   - Make sure to include system prompt.
3. ✅ What is the average number of tokens in your LLM output?
4. ✅ How many requests per second should your system process at its peak?
5. ✅ What is your latency limit? First-token? Last-token?
6. ✅ What GPUs are you considering?



# Which Model?

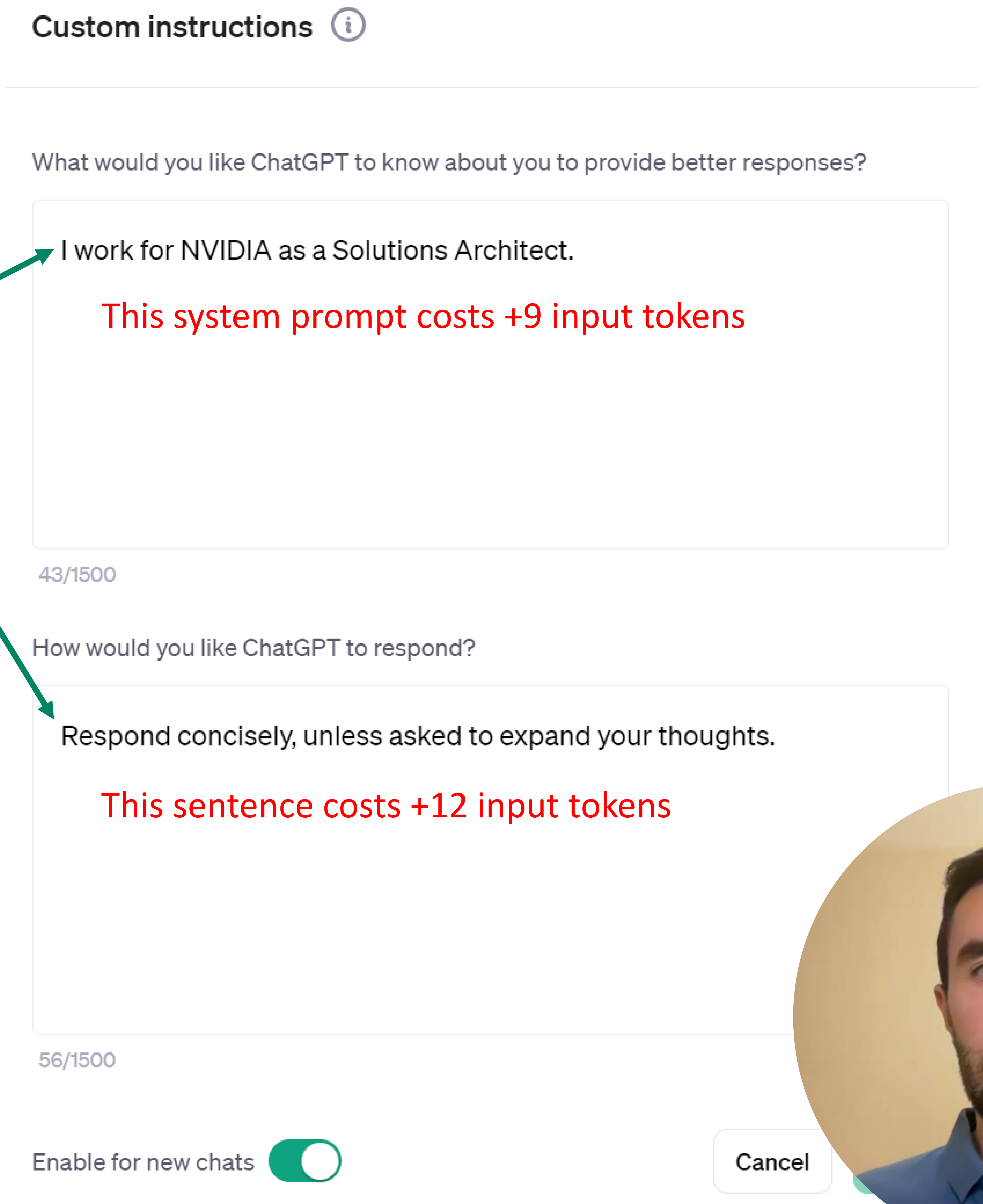
## The most popular requests

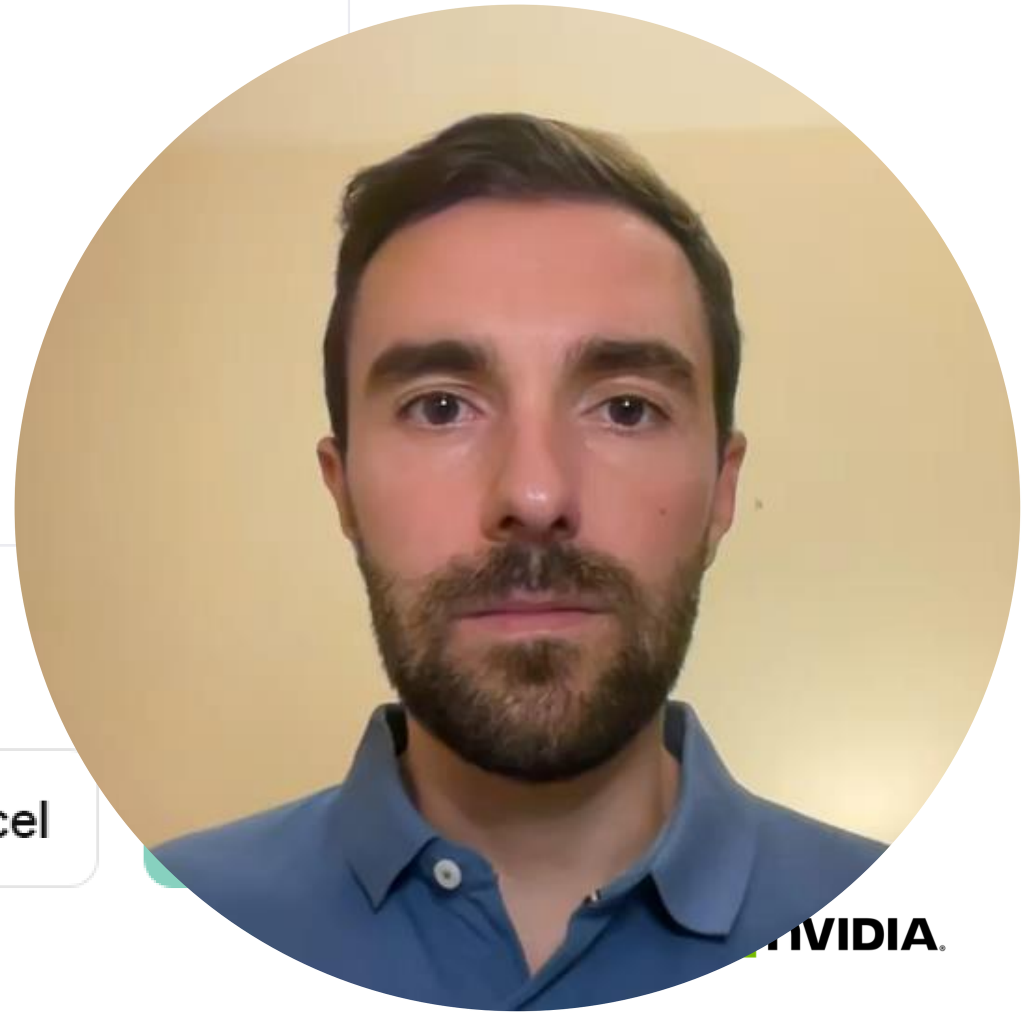
- Typically, we get asked about Llama 3 family of the models
  - Free for research and commercial use
  - Supported by NVIDIA SW stack, including NeMo, NIM, TRT-LLM and Triton
- The bigger the model, the more resources it needs for inference
  - The bigger the model the better the accuracy
  - Very roughly, the resource amount scales with the model size
- If considering Mistral 7B or Llama-3 8B parameters, see also NVIDIA Nemotron-3 8B Family of models: blog


# Input Length

## There's a maximum budget of tokens to pass into the model

- Most of the models support up to 4096 tokens.
  - Context window = input tokens + output tokens
  - Llama2 supports 4096 context window
  - New models support even larger context windows
- Everything counts so be careful:
  - **System prompt** (a.k.a custom instructions): instructions you give to the model for every "dialogue". Make sure to include them into the input token count as shown in example on the right
  - **Retrieved documents** (a.k.a Retrieval Augmented Generation, RAG). For RAG pipelines key paragraphs from the internal document storage are added to the prompt, before the user requests. Typically RAG systems target to use full available context length
    - For 4K context typical 3500 input tokens, 500 output tokens
    - What is RAG — NVIDIA blog
  - **Chat history**: previous exchange of messages in the conversation

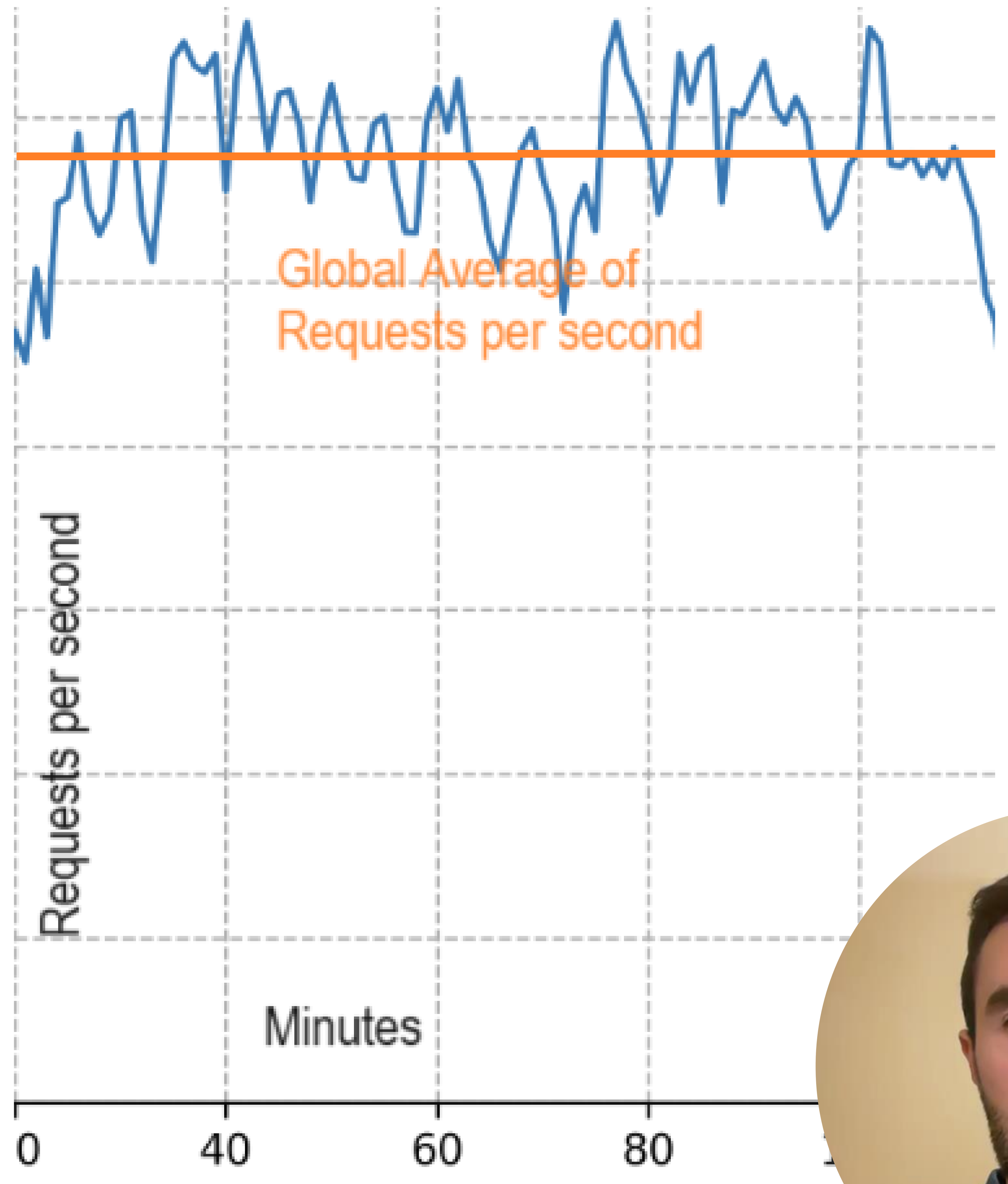


# Peak Requests Per Second

- Poisson distribution approximation
  - One knows the average, but would like to know the peak
- Find 95th percentile: [ChatGPT dialogue]

```
from scipy.stats import poisson

# Parameters
lambda_ = 64  # average number of requests per second
percentile = 0.95  # 95th percentile

# Calculate the 95th percentile value
k_95th_percentile = poisson.ppf(percentile, lambda_)
print(k_95th_percentile) # 77, 20% difference
print(poisson.ppf(0.95, 7)) # 12, 71% difference
```



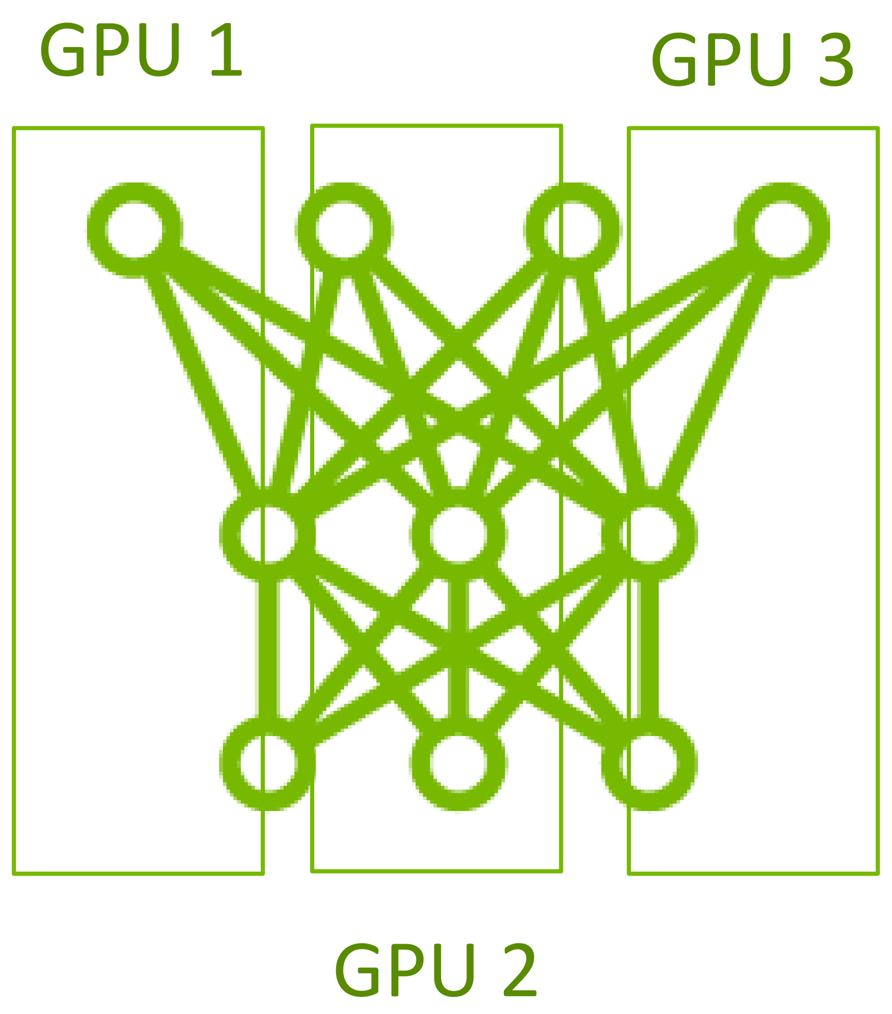

# LLM Inference Requires Multiple GPUs

## Tensor Parallelism (TP) – so how to split your neural network

- Tensor Parallelism (TP) can be used for LLM Inference. One model gets split across several GPUs, which heavily relies on data exchange between GPUs
  - Lower latency, but lower throughput
  - TP >= 2 required for bigger models like LLaMa-70B
- If TP>2 we strongly recommend NVLink-enabled servers for inference, such as HGX and DGX systems (in contrast to PCIe servers)
- We normalize all the results for servers with 8 GPUs (even for L40S)
  - An instance is the group of GPUs forming a data replica of the model
  - (# of instances) * TP = 8
  - 8 instances with TP1, 2 instances with TP4

**Time = $**

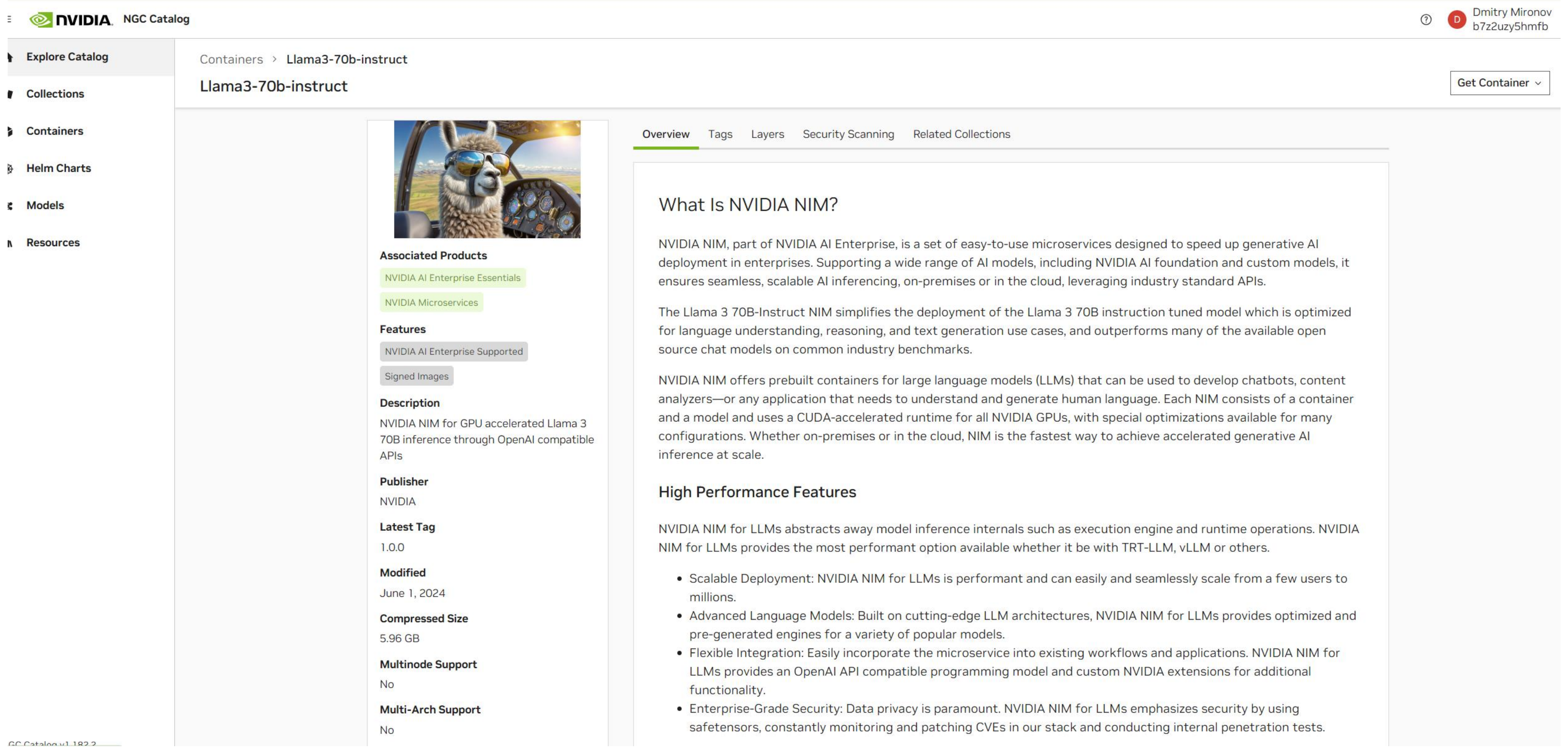
| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| TP8 | Instance 1 | | | | | | | |
| TP4 | In. 1 | | | | In. 2 | | | |
| TP2 | In. 1 | | In. 2 | | In. 3 | | In. 4 | |
| TP1 | In. 1 | In. 2 | In. 3 | In. 4 | In. 5 | In. 6 | In. 7 | In. 8 |

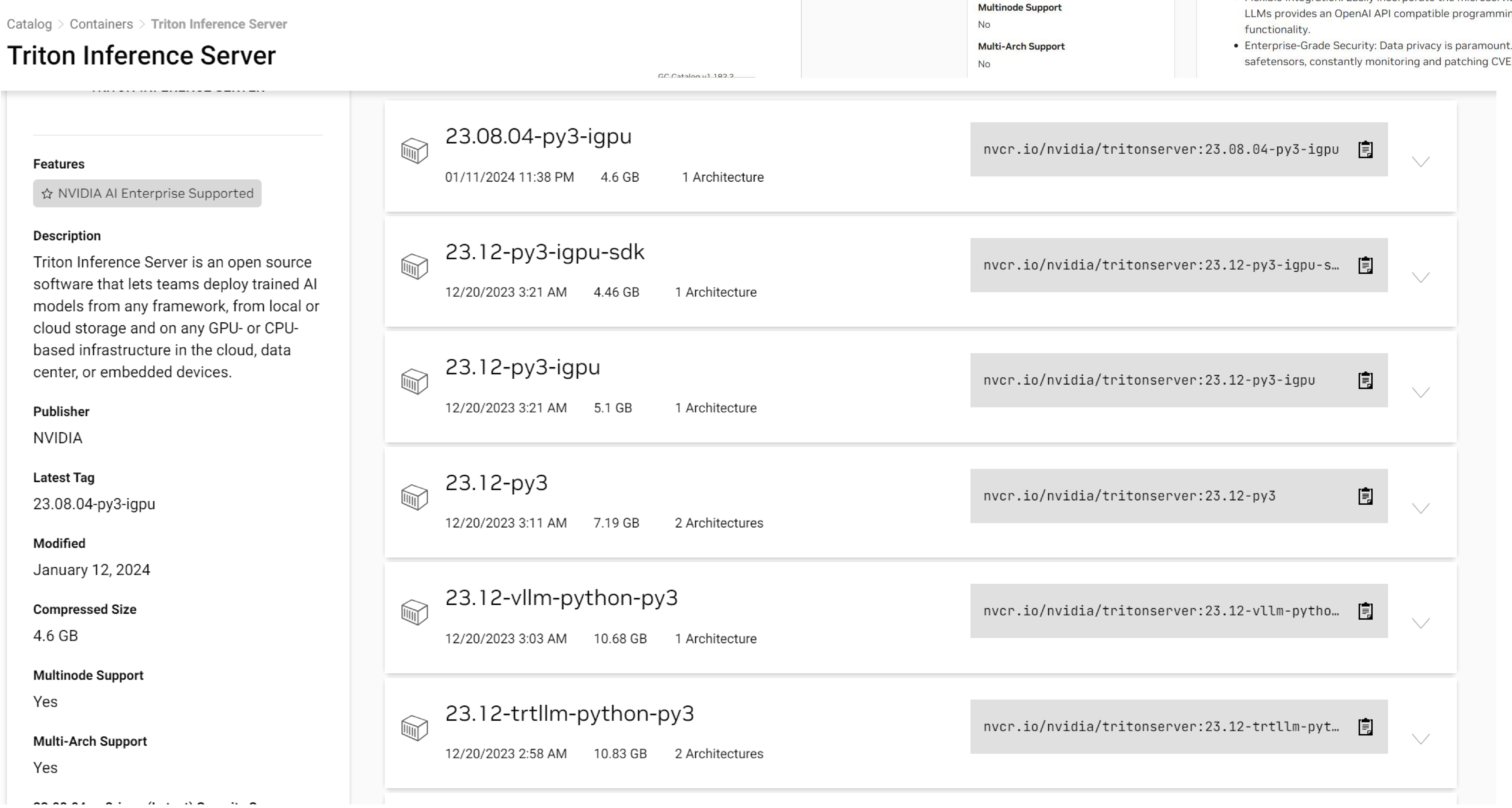



# Tools Available

# Inference Containers

- Triton + TensorRT-LLM
  - Open Source hands-on tools
  - TensorRT-LLM optimizes a model for inference
  - Triton is an inference server
- NVIDIA NIM
  - Deploy a LLM within minutes
  - Supports OpenAI-compatible API
  - Accelerated by TRT-LLM




# AI Inference Workflow


# TensorRT-LLM Optimizing LLM Inference

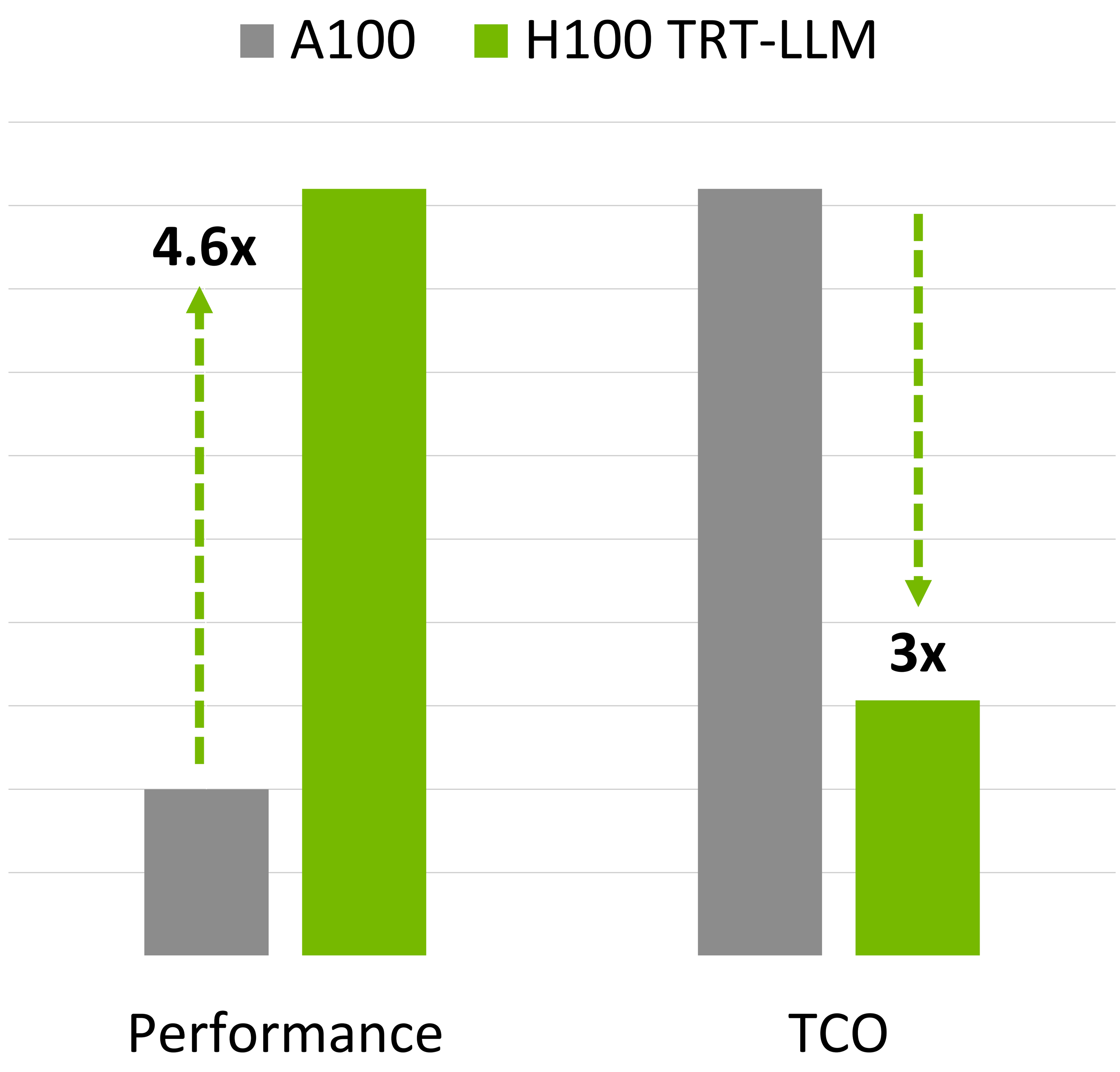
## SoTA Performance for Large Language Models for Production Deployments

***Challenges***: LLM performance is crucial for real-time, cost-effective, production deployments. Rapid evolution in the LLM ecosystem, with new models & techniques released regularly, requires a performant, flexible solution to optimize models

***TensorRT-LLM*** is an **open-source** library to **optimize inference performance** on the latest **Large Language Models** for NVIDIA GPUs. It is built on FasterTransformer and TensorRT with a simple Python API for defining, optimizing, & executing LLMs for inference in production

### SoTA Performance

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



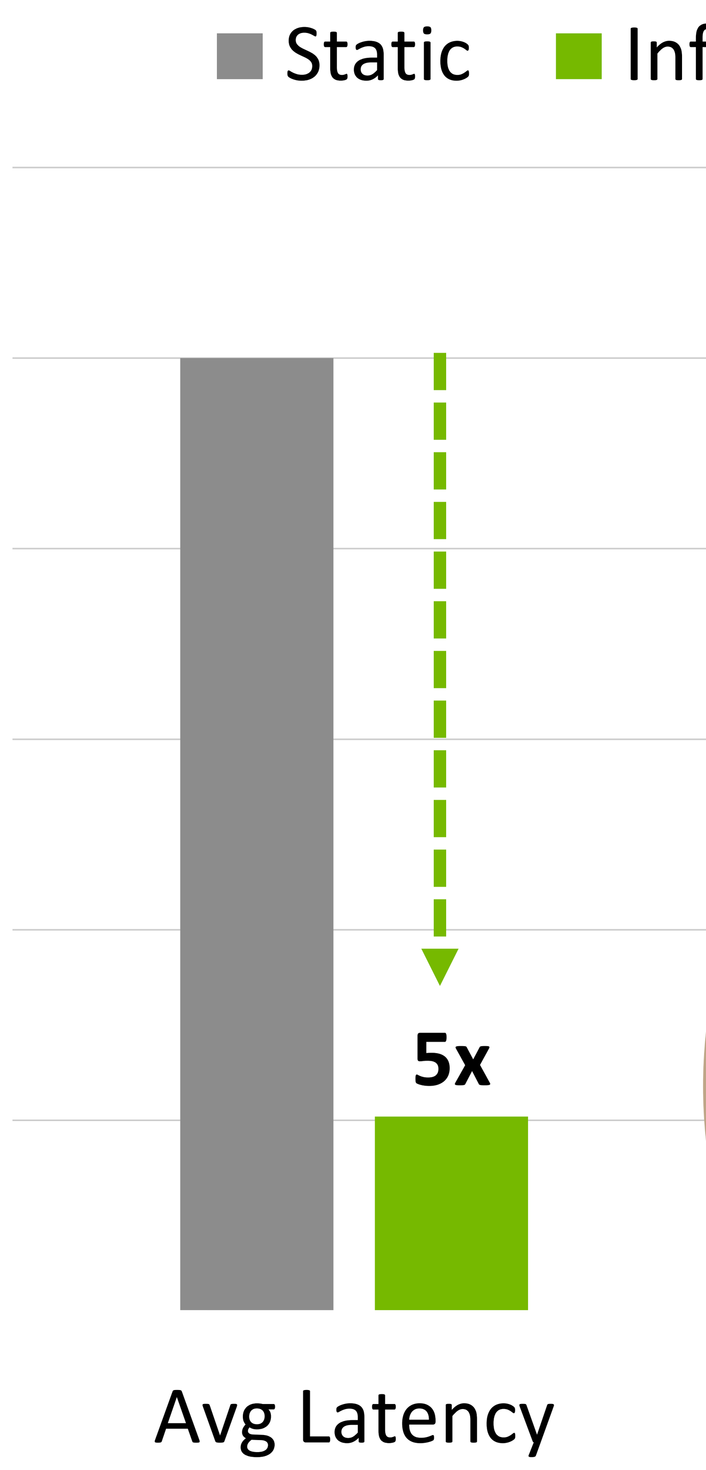

### Ease Extension

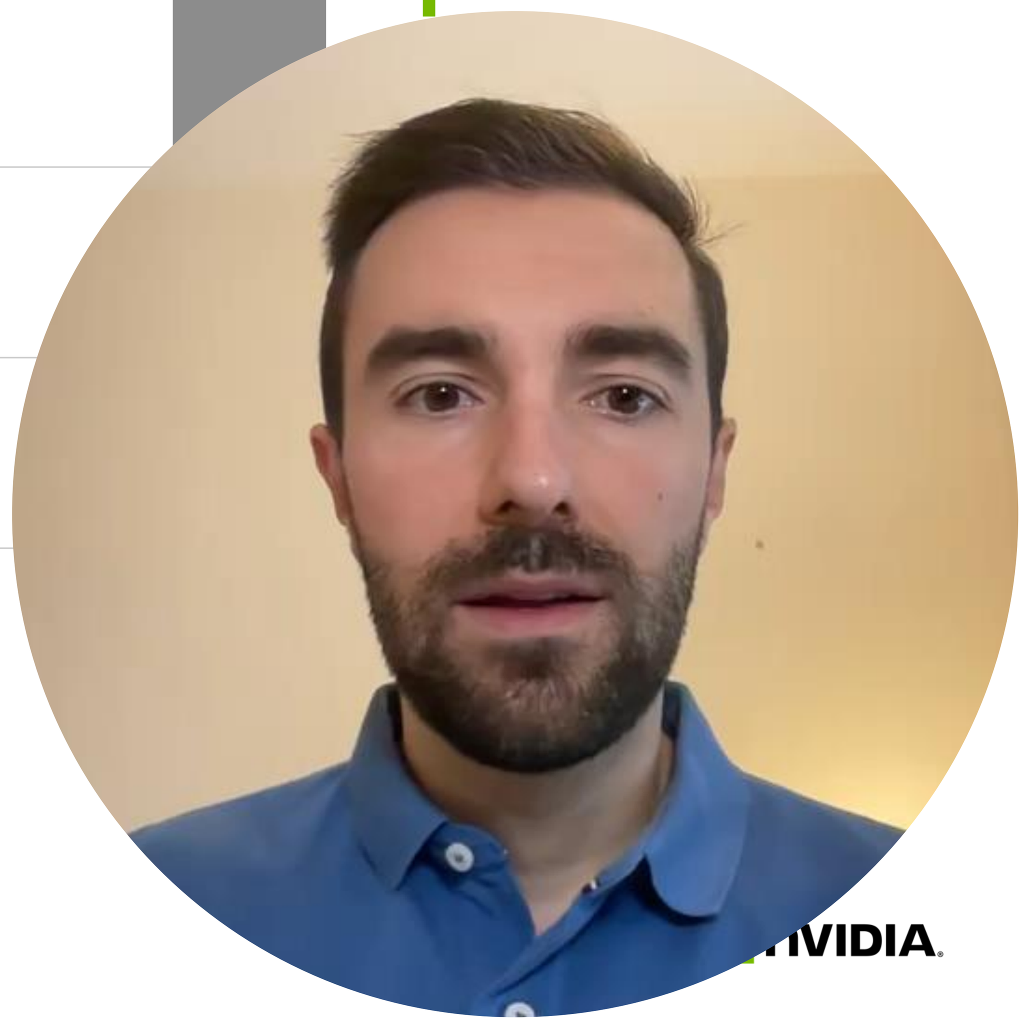
Add new operators or models in Python to quickly support new LLMs with optimized performance

```
# 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



Numbers are preliminary based on internal evaluation on Llama 7B on H100

# 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 Configur

Opt
GPU/
Thro

# 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 with security and control** of AI applications and data

**Speed time to market** with prebuilt, continuously maintained microservices

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

**Optimize throughput** and latency to maximize token generation and responsiveness

**Boost accuracy** by tuning custom models from proprietary data sources

**Deploy in production** with API stability … quality assurance and enterprise su…

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

# Publicly Available Performance Benchmarking

- Most recommended: GenAI-Perf from Triton team
  - https://github.com/triton-inference-server/client/tree/main/src/c%2B%2B/perf_analyzer/genai-perf
  - Triton GenAI-Perf is a CLI tool which can help you optimize the inference performance of models running on Triton Inference Server and OpenAI endpoints by measuring changes in performance as you experiment with different optimization strategies.
  - Used in NIM for LLMs Performance Guide https://docs.nvidia.com/nim/benchmarking/llm/latest/index.html

- https://github.com/NVIDIA/TensorRT-LLM/tree/main/benchmarks/cpp — TensorRT-LLM C++
  - TensorRT-LLM provides users with an easy-to-use Python API to define Large Language Models (LLMs) and build TensorRT engines that contain state-of-the-art optimizations to perform inference efficiently on NVIDIA GPUs. TensorRT-LLM also contains components to create Python and C++ runtimes that execute those TensorRT engines.
  - Some results: https://github.com/NVIDIA/TensorRT-LLM/blob/main/docs/source/performance.md

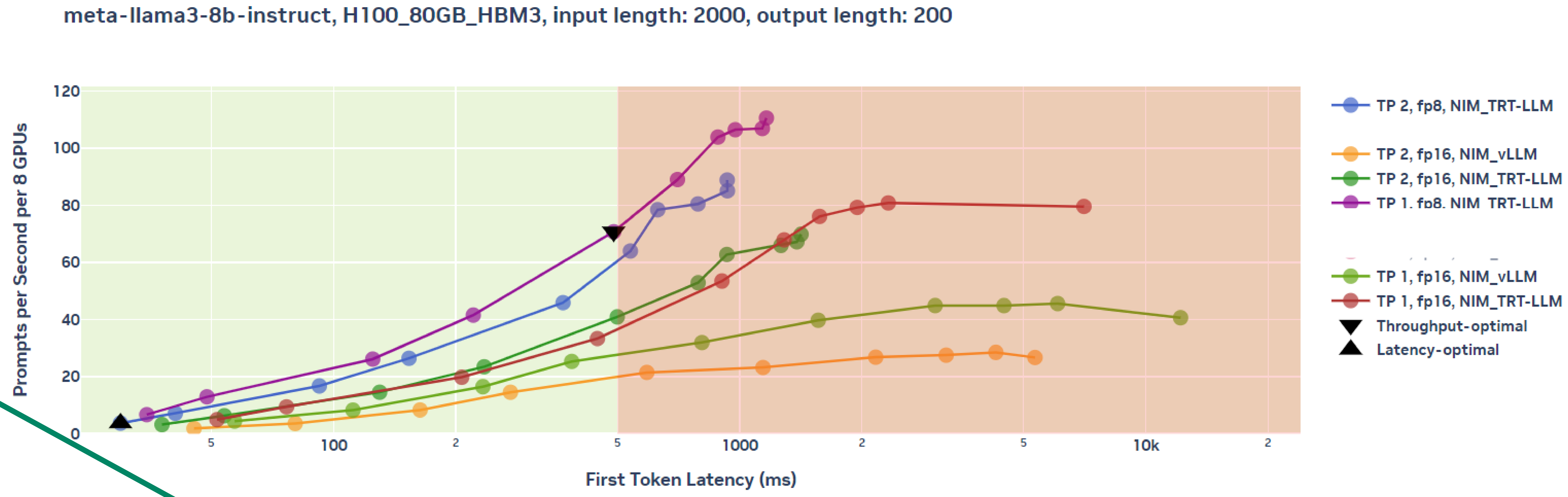
- Triton CLI for limited experimentation: https://github.com/triton-inference-server/triton_cli

# Example of benchmarking

# Example with Llama 3 8B

## Smaller model – for auxiliary task

- We are looking for a sizable use case of **Llama3-8B**. 2000 in, 200 out, **TTFT < 500ms**
- For input 2000, output 200 we have **70.7** peak prompts per second per **one** DGX H100
- That's 2M requests per working day (8 hours)
- 3 requests per person → **679k daily active users**
- 4.2B input, 261M output tokens per day

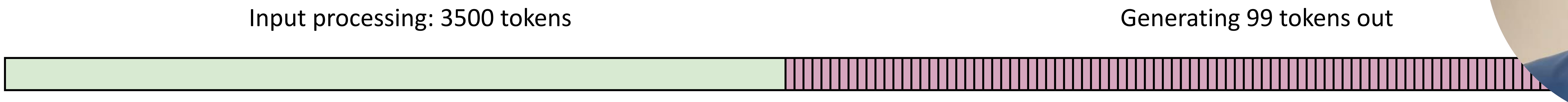



Best performance per optimal scenario

| Metric | Throughput-optimal ▼ | Latency-optimal ▲ |
|---|---|---|
| First Token Latency (ms) | 489.5 | 29.9 |
| Prompts per Second per 8 GPUs | 70.7 | 3.7 |
| Latency per Generated Token (ITL, TPOT) (ms) | 11.7 | 5.24 |
| Prompts per Second per 1 GPU | 8.8 | 0.47 |
| Cost Per 1000 Prompts (USD) | 0.053 | 1 |
| Cost Per 1M Input Tokens (USD) | 0.021 | 0.39 |
| Cost Per 1M Output Tokens (USD) | 0.062 | 1.2 |

# Rules of Thumb for Sizing

- We estimate the sizing based on NVIDIA SW stack: NIM or TensorRT-LLM (=TRT-LLM) and Triton Inference Server
- For models greater than 13B, that need more than 1 GPU, prefer NVLink-enabled systems
- In the streaming mode, when the words are returned one by one, first-token latency is determined by the input length
- The cost and the latency are usually dominated by the number of output tokens
  - Example below: H100 SXM, Llama 70B, BS 8, TP 4, FP 16.
    Input of 3500 tokens takes the same amount of time as generating 99 tokens
    (2.6 seconds each stage, 26.8 ms/generated token)
  - However, generating is almost always faster than human reading speed
  - Thus, input tokens are much cheaper
- Introducing latency limit can significantly decrease available throughput
- Larger models require more memory and have higher latency, scaling approximately with the model size




# Inference Resources

- NIM for LLM Benchmarking Guide https://docs.nvidia.com/nim/benchmarking/llm/latest/index.html
- NVIDIA NIM: https://docs.nvidia.com/nim/index.html
- GTC session about LLM inference sizing: https://www.nvidia.com/en-us/on-demand/session/gtc24-s62797/
- Mastering LLM Techniques: Inference Optimization— NVIDIA Blog https://developer.nvidia.com/blog/mastering-llm-techniques-inference-optimization/

NVIDIA