S62797 - LLM Inference Sizing: Benchmarking End-to-End Inference Systems

Dmitry Mironov
Solutions Architect, NVIDIA
Sergio Perez
Solutions Architect, NVIDIA

About Us

Let's coordinate inference conversation

- Senior Deep Learning Solutions Architect @ NVIDIA - Supporting deployment of AI / Deep Learning solutions
- Focusing on large scale efficient deployment and inference
- Co-author of NeMo Inference Sizing Calculator

Dmitry Mironov, EMEA
dmitrym@nvidia.com

- Senior Deep Learning Solutions Architect @ NVIDIA - Supporting delivery of AI / Deep Learning solutions
- Focusing on quantization in training and inference
- Co-author of NeMo Inference Sizing Calculator

Sergio Perez, EMEA
sergiop@nvidia.com

Agenda

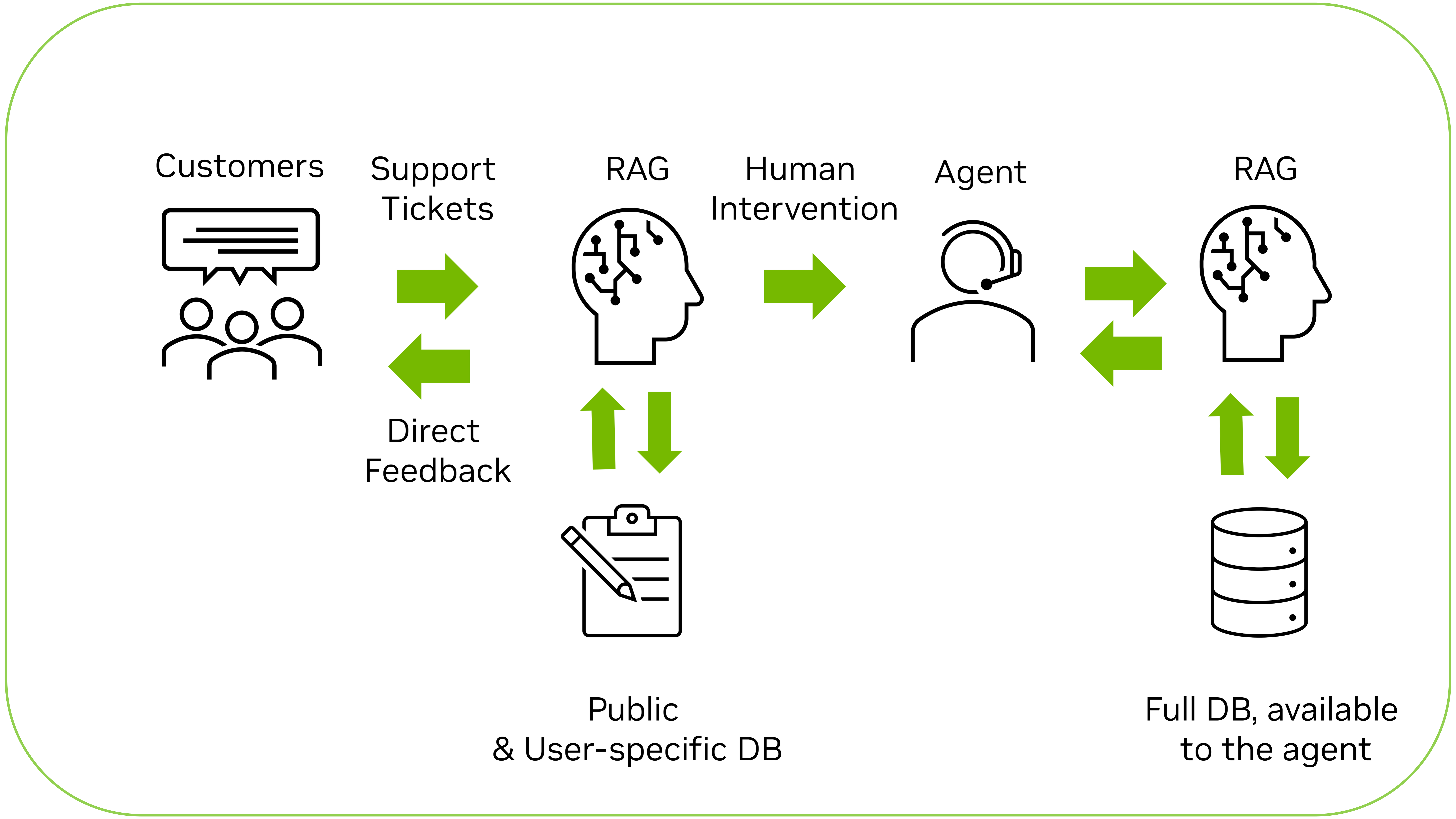
Sizing for Inference can get a bit complicated

Inference performance tools to help you are emerging

Short summary of how to think about a problem and live demo

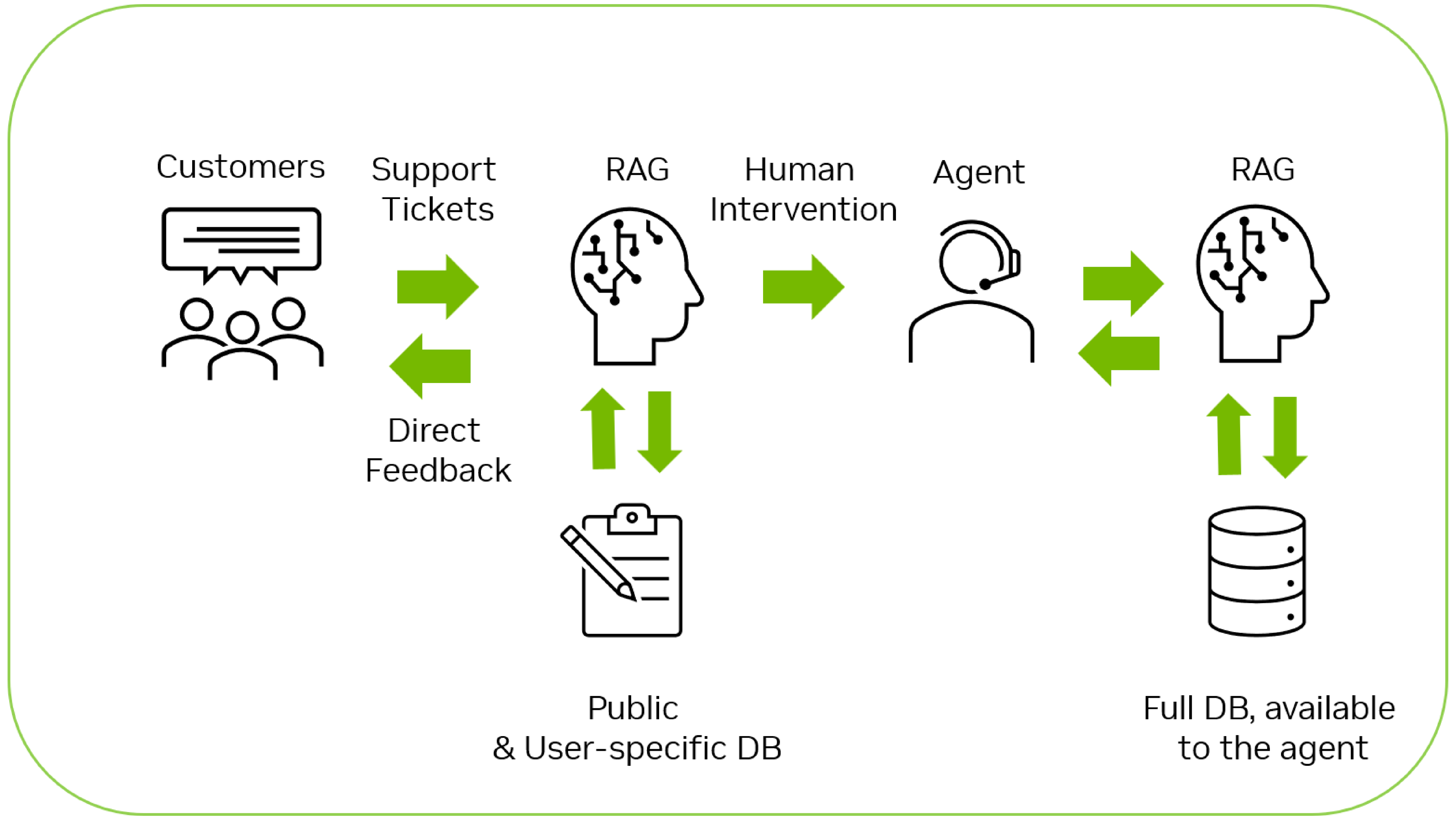
Customer Use Case Example

Challenges of sizing



Customer Use Case Example

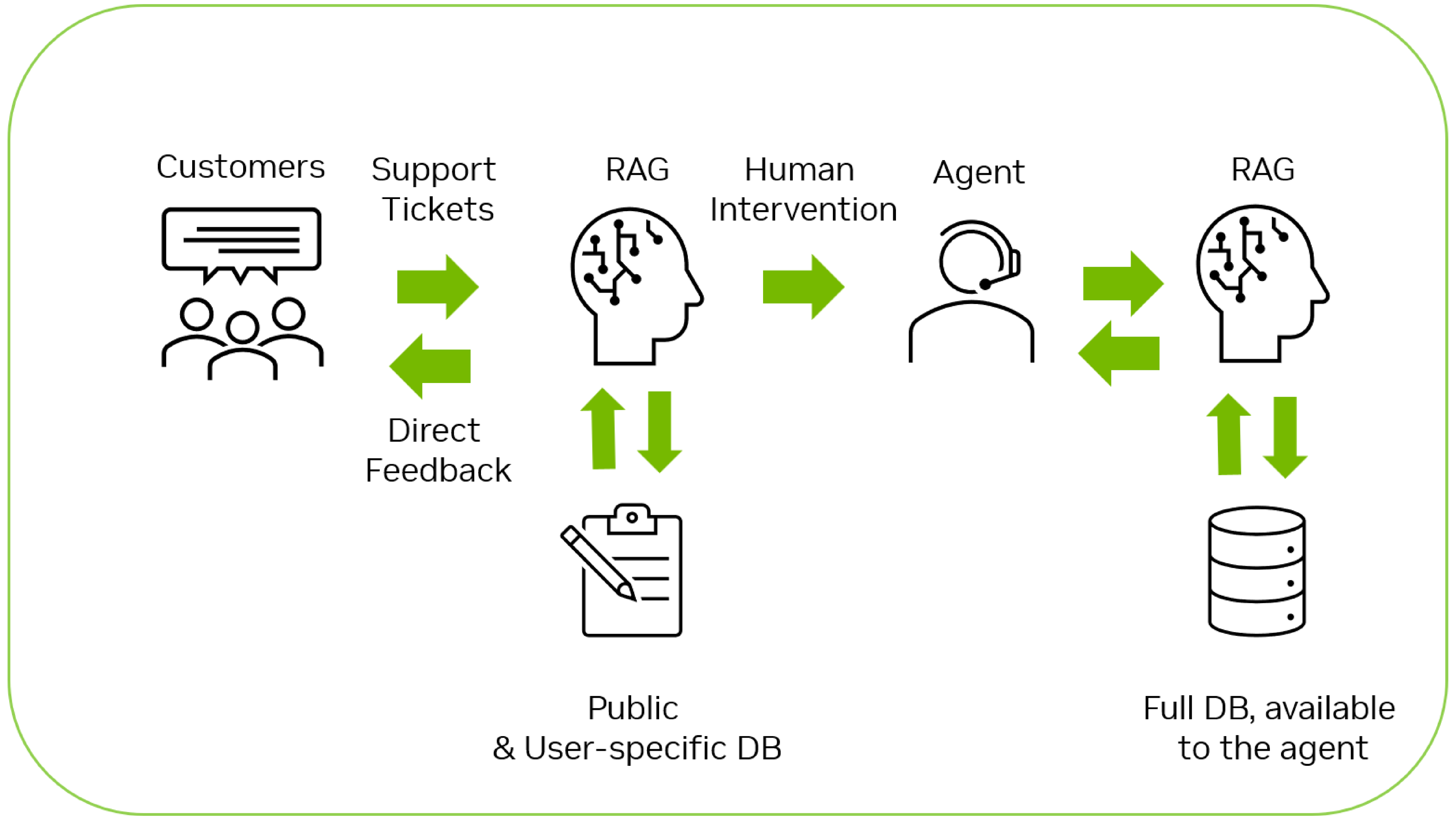
Challenges of sizing



How many systems do we need to buy for this?

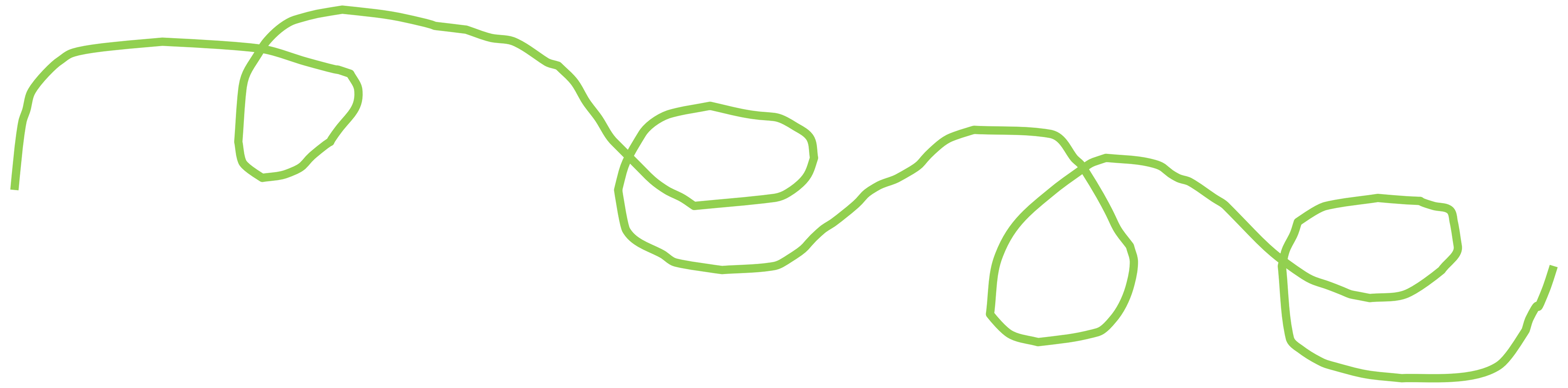
Customer Use Case Example

Challenges of sizing



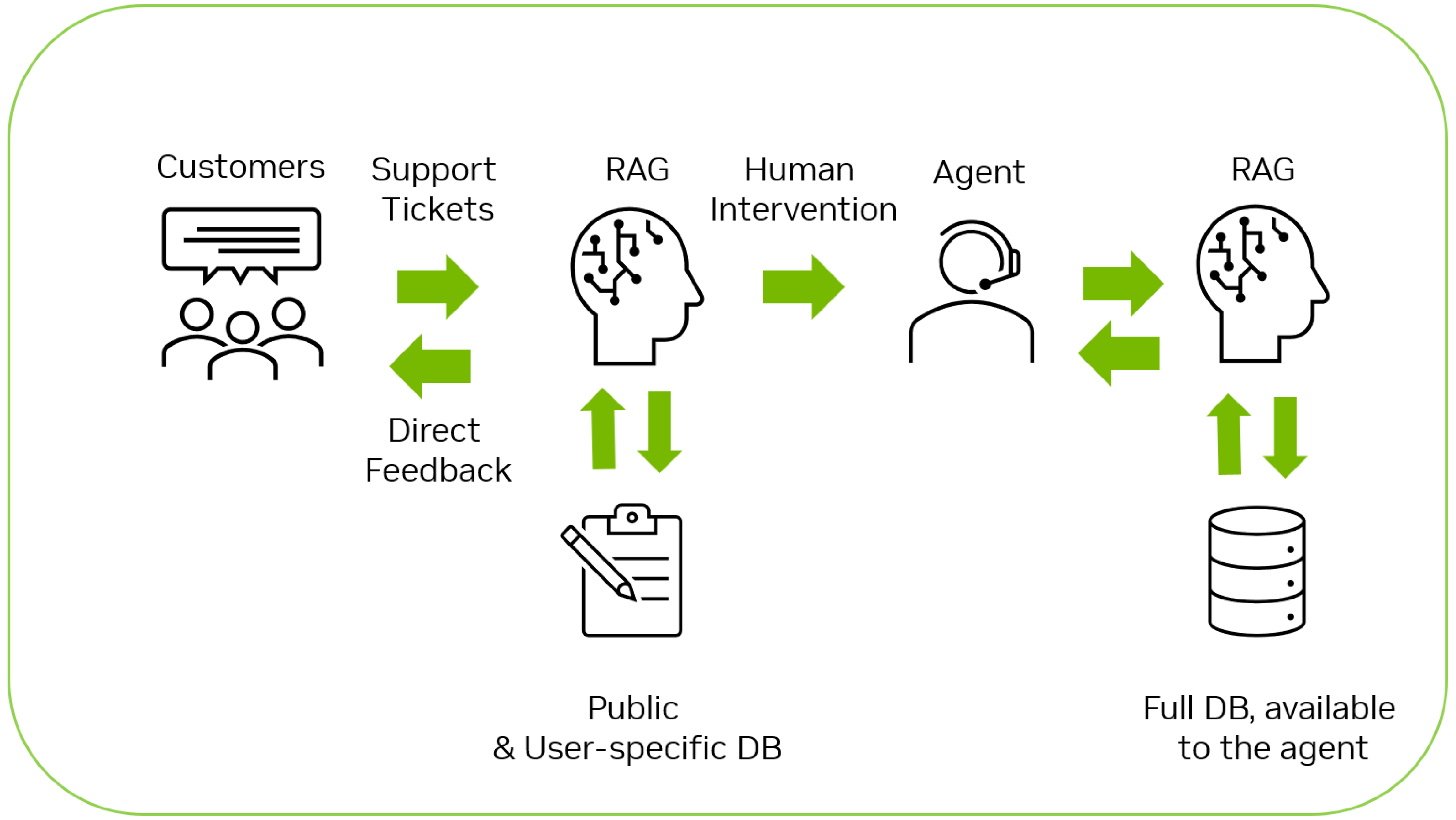
How many systems do we need to buy for this?

How long is a piece of string?



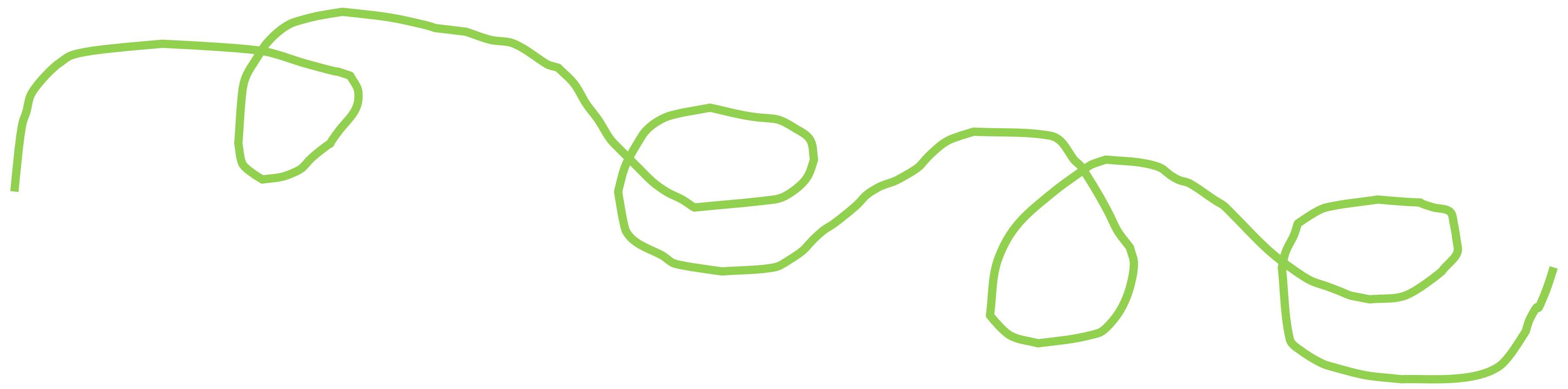
Customer Use Case Example

Challenges of sizing



How many systems do we need to buy for this?

How long is a piece of string?



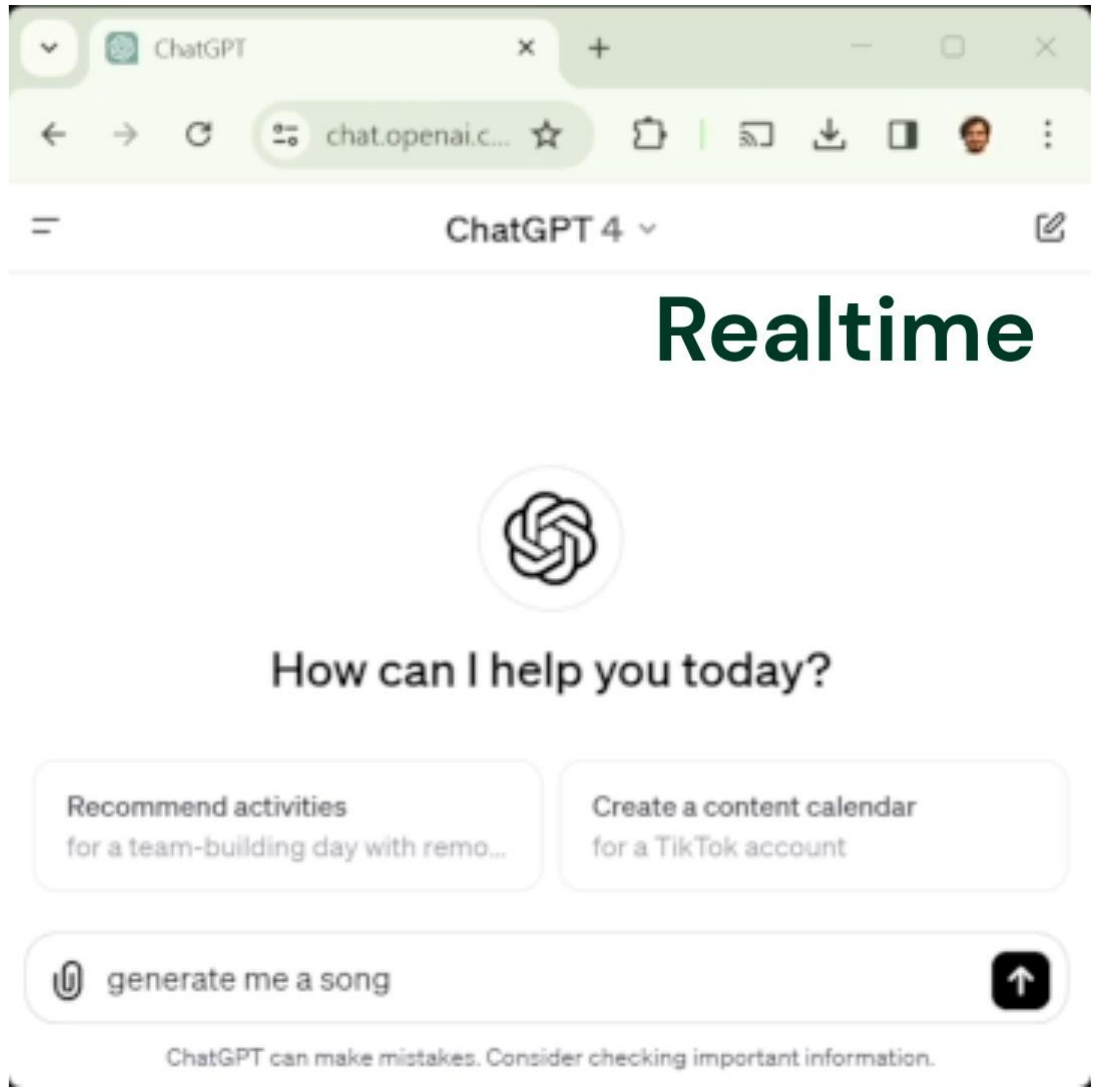
- 3500 words in, 500 words out
- NeMo 43B GPT
- First token latency limit 3s
- Max 31 requests (=prompts) per second

- **The customer needs 13 DGX H100 systems**
  - DGX H100 can serve 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 specifics`

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

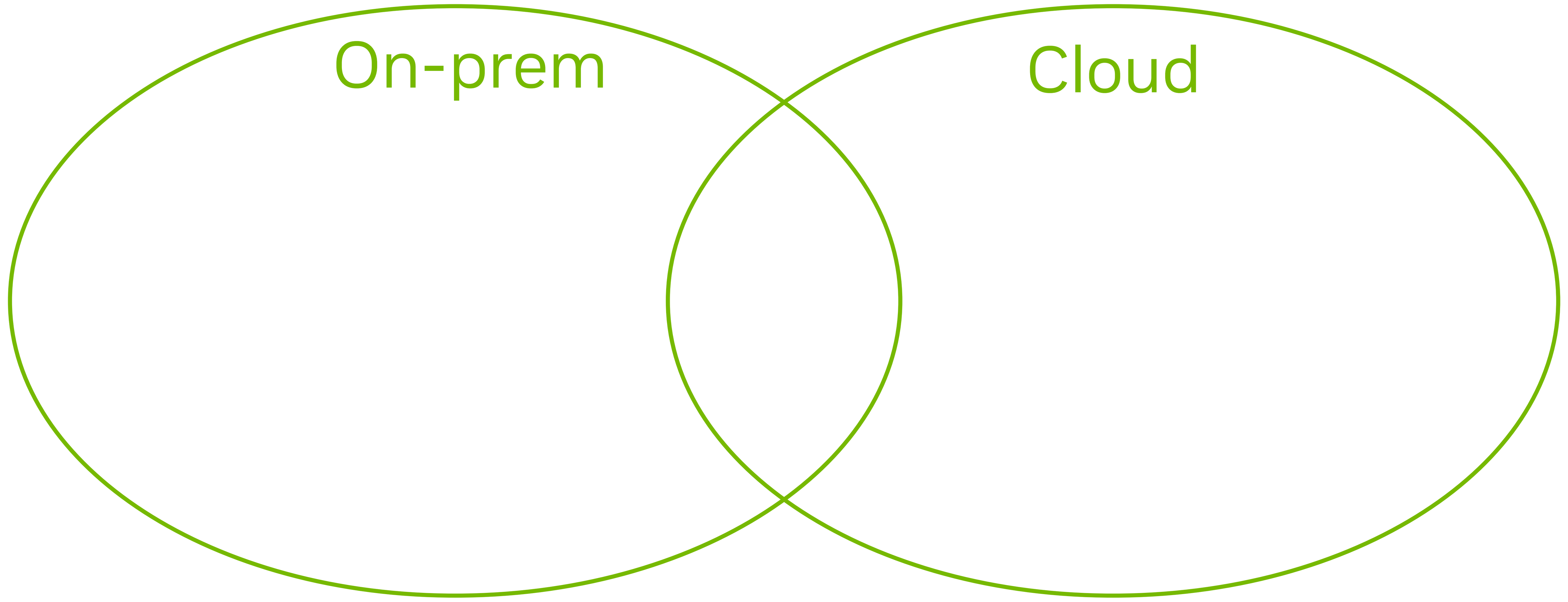
prompt

user prompt

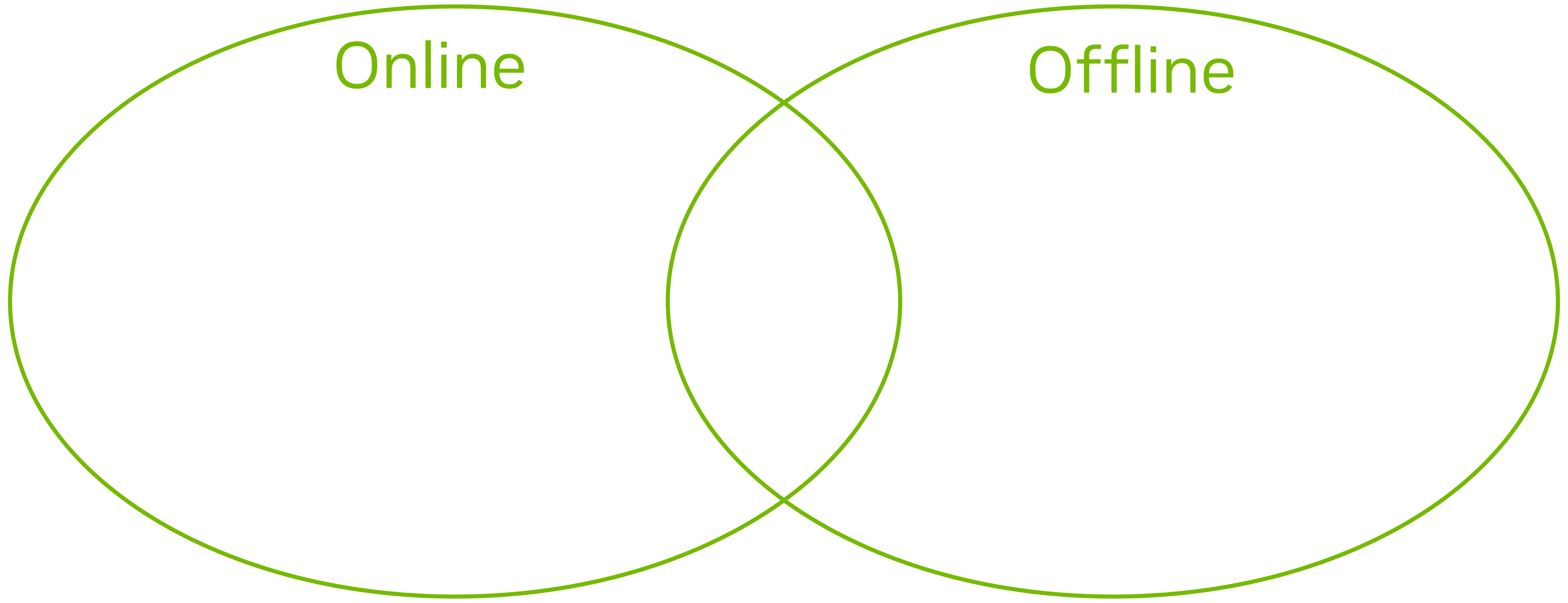
The Two Things To Care About

Where and how do we execute inference?

Where?



How?



Where?

Significant impact of deployment location

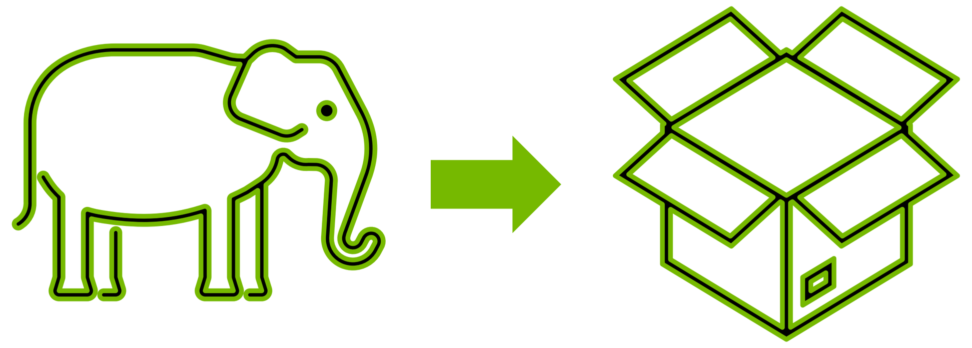
On-prem

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

- Minimal Capacity + Autoscaling for bursts

Cloud

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



How?

Significant impact of inference strategy

Online

- 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

- 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**, when it is OK to give people answer 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**, when one 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

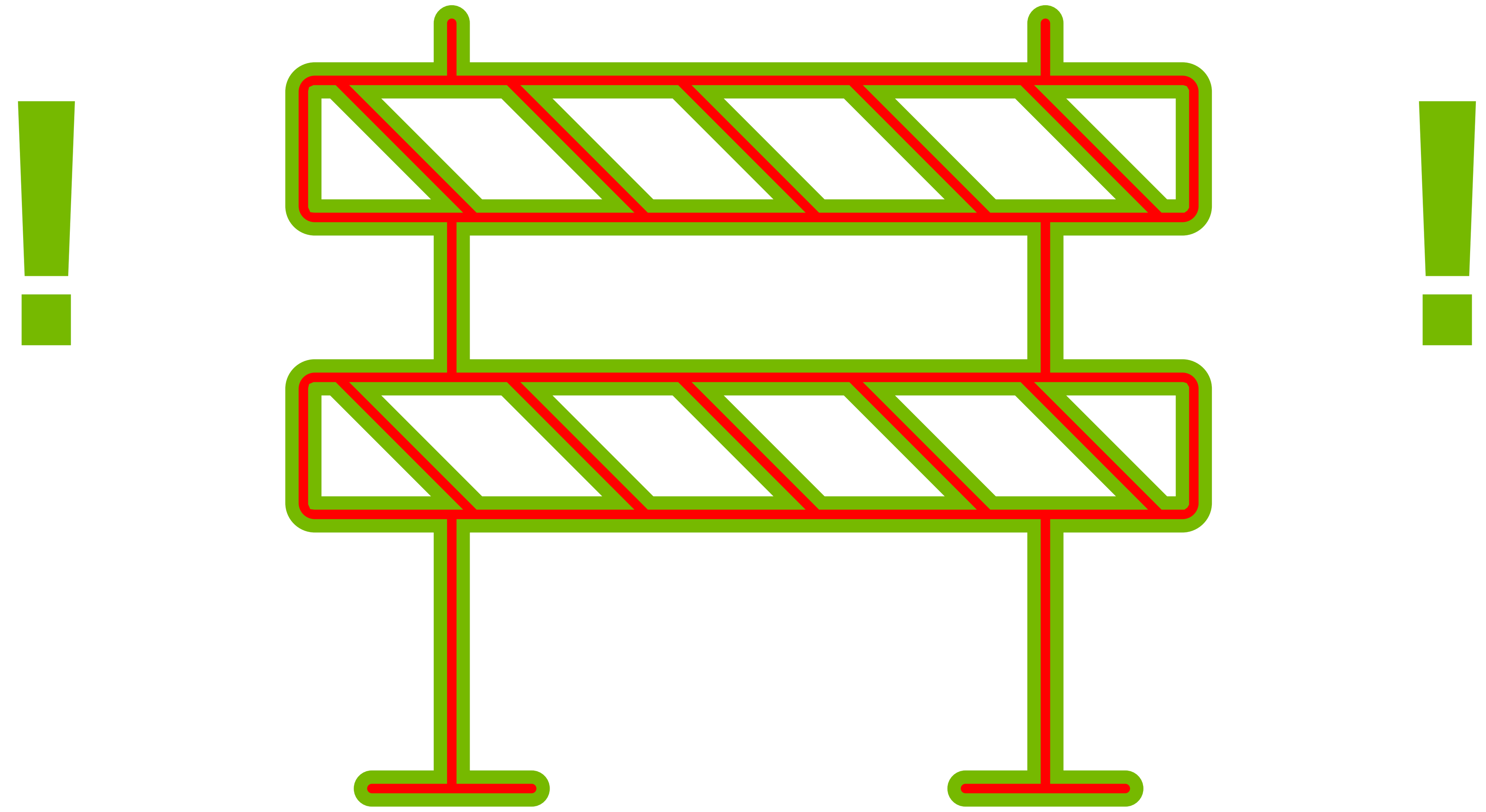
In This Presentation

We focus on the most complicated part of the problem

Sizing One Use Case

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?



Which Model?

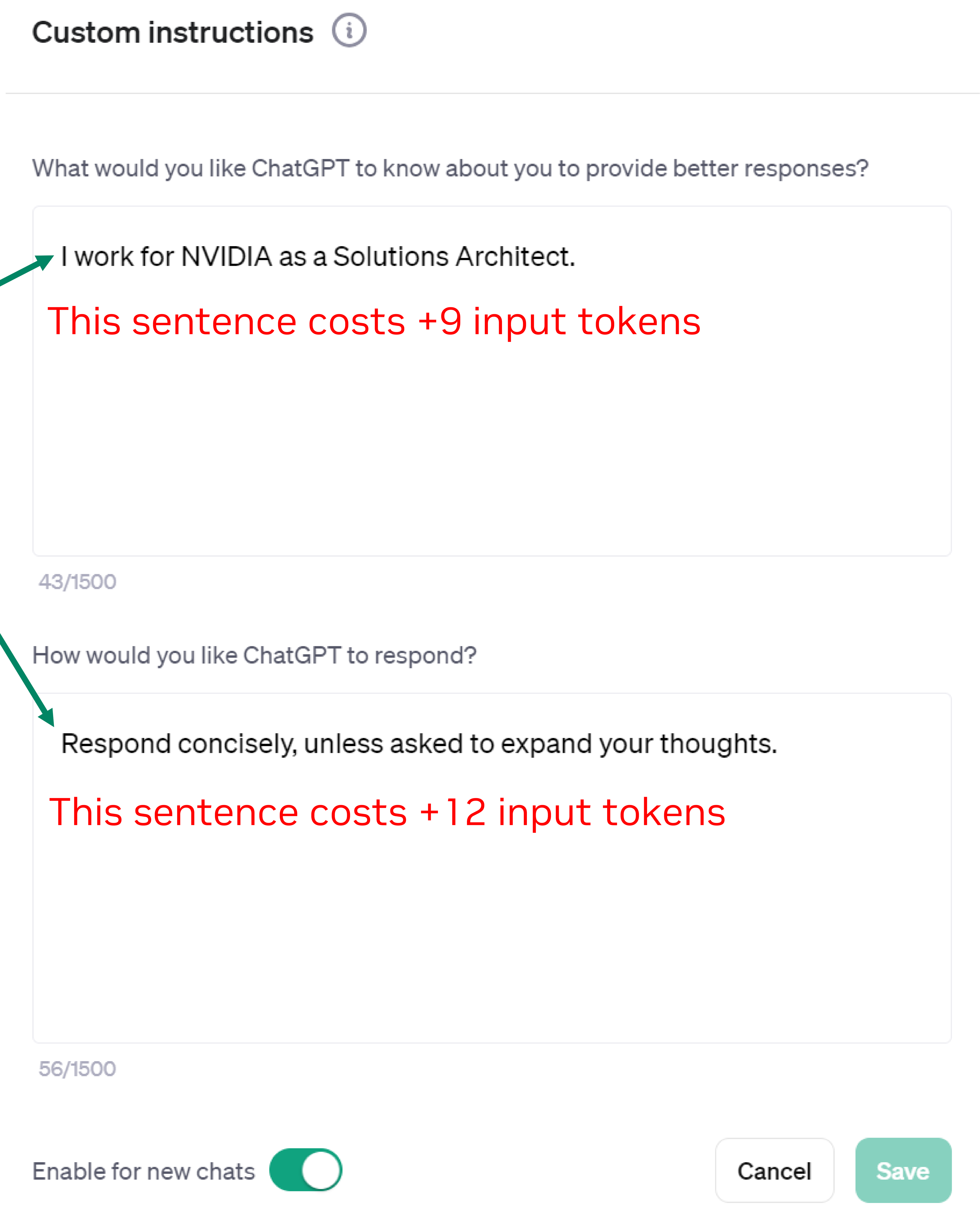
The most popular requests

- Typically, we get asked about LLaMa 2 family of the models:
  - Free for research and commercial use
  - Supported by NVIDIA SW stack, including NeMo
- 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 LLaMa 7B and 13B 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



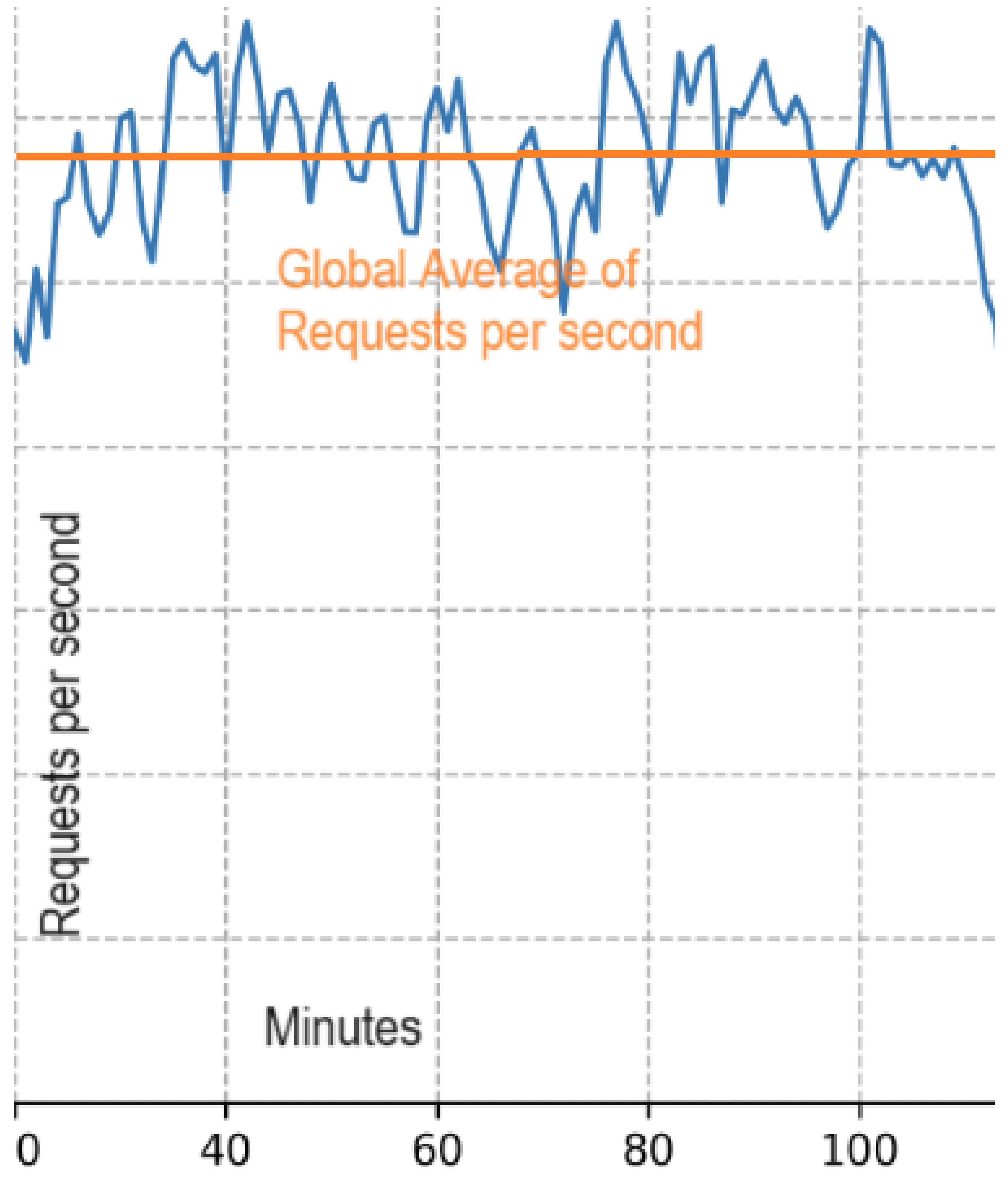
Peak Requests Per Second

- 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
```



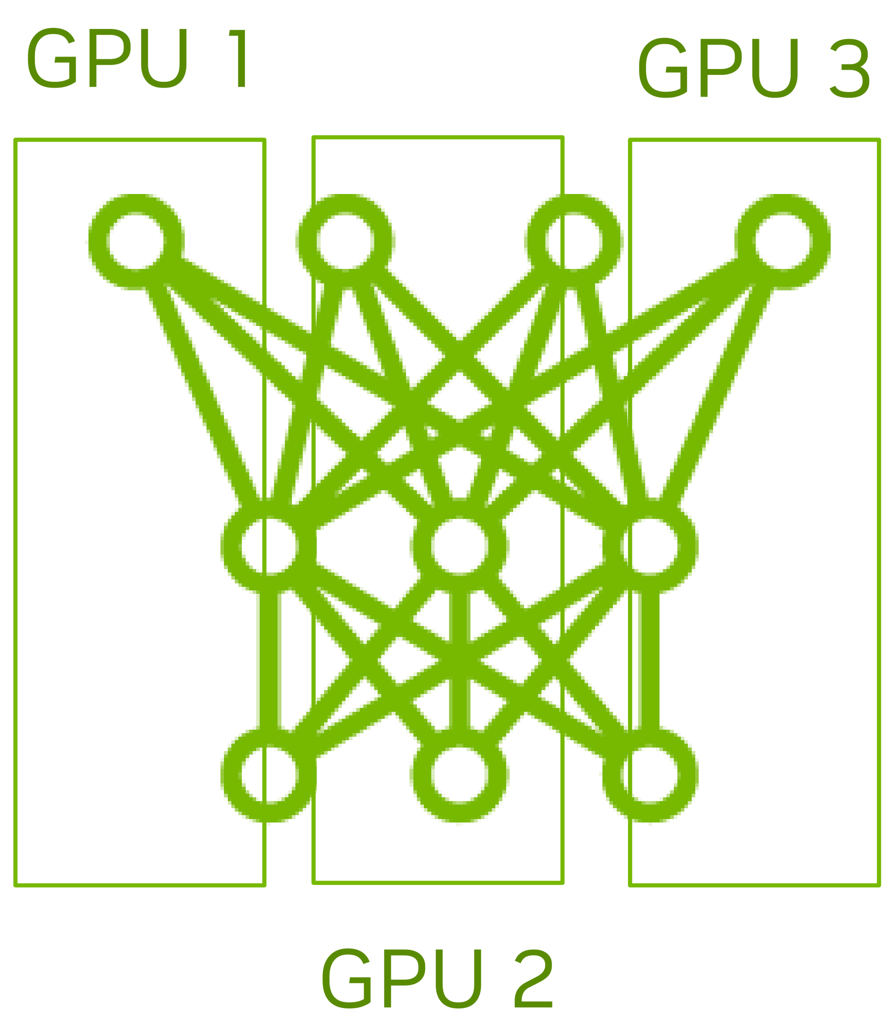
LLM Inference Requires Multiple GPUs

Tensor Parallelism (TP) – so how to split your neural network across several GPUs

- Tensor Parallelism (TP) can be used for LLM Inference. One model gets split across several GPUs. 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
- We normalize all the results for servers with 8 GPUs (even for L40s)
  - (# 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

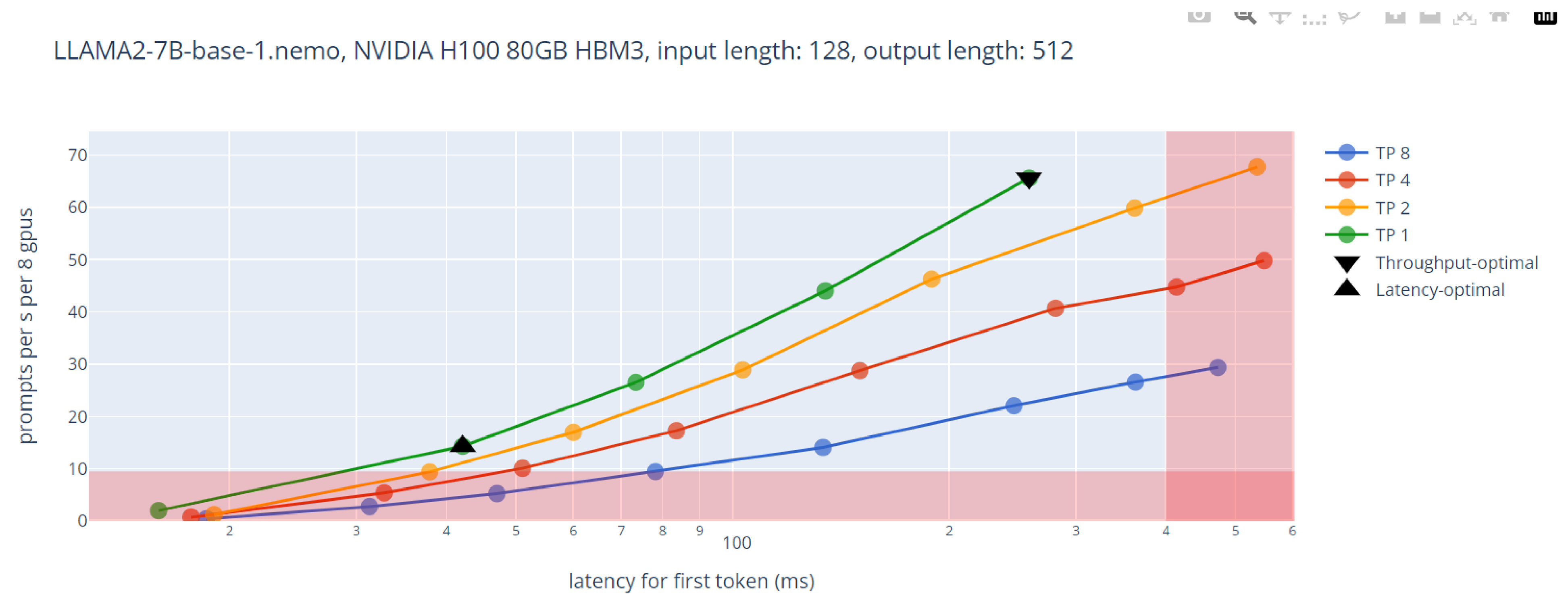
Publicly Available Performance Benchmarking

- 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
- https://github.com/triton-inference-server/client/blob/main/src/c%2B%2B/perf_analyzer/docs/llm.md
  - Triton Performance Analyzer is CLI tool which can help you optimize the inference performance of models running on Triton Inference Server by measuring changes in performance as you experiment with different optimization strategies.

Inference Performance Exploration Tools

- Simpler, less precise, benchmarks-based https://nemo-inference-sizing.nvidia.com/ (to be published)
- Very precise, more complex, benchmarks + simulation: Nemo Inference Microservice (available Early Access)

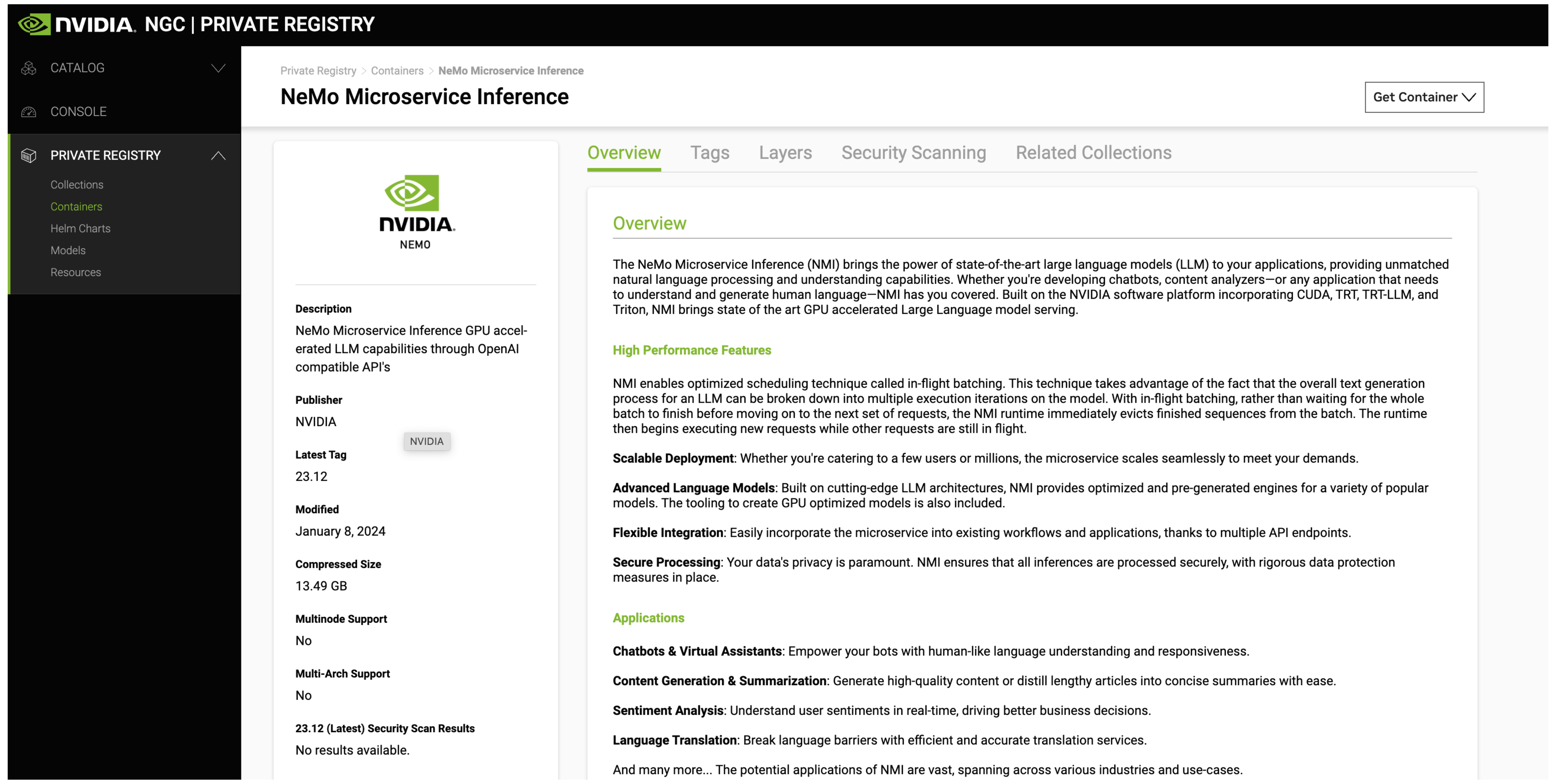
https://nemo-inference-sizing.nvidia.com/



Recommended Configurations Within Limits

Metric	Throughput-optimal ▼	Latency-optimal ▲	Clicked data
Latency for first token (ms)	258.3	42.2	[Click on a point]
Latency per generated token (ms)	14.8	8.7	[Click on a point]
Prompts per second per 8 GPUs	65.6	14.3	[Click on a point]
Tensor Parallelism	1	1	[Click on a point]
Batch size	64	8	[Click on a point]

Nemo Inference Microservice



*name on the page will be updated

Demo of NeMo inference sizing

Time to first token (Prefill)	Inter token latency (Decoding)	End to End time (Prefill + Decoding)

Model

LLAMA2-13B-base-1.nemo

Input length (tokens)

128

Output length (tokens)

1

GPU

NVIDIA H100 80GB HBM3

Inference requirements

Maximum first token latency: 1200 ms

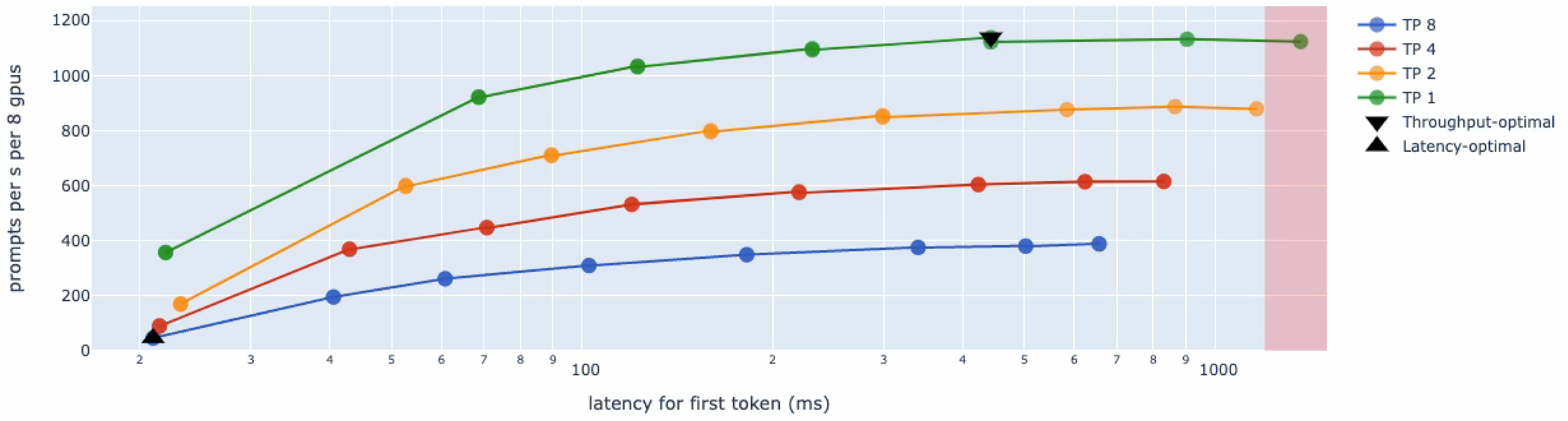
0 500 1.0k 1.4k

Minimum prompts per second per 8 GPUs: 0

0 380 760 1.1k

This area does not satisfy the inference requirements

LLAMA2-13B-base-1.nemo, NVIDIA H100 80GB HBM3, input length: 128, output length: 1



Recommended Configurations Within Limits

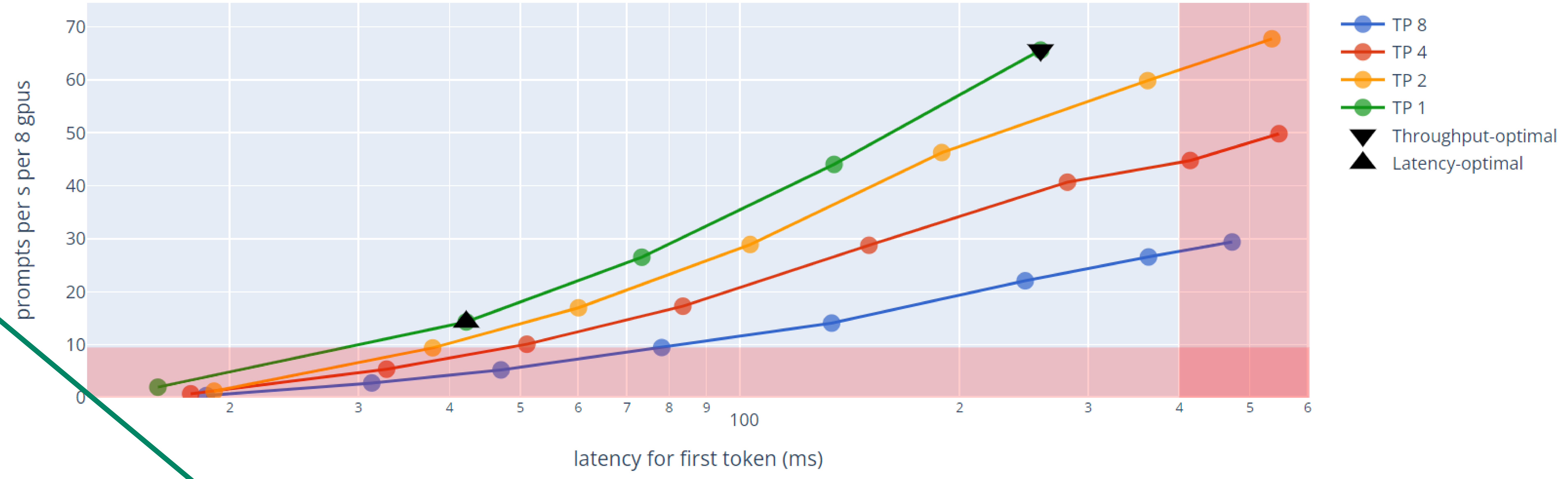
Metric	Throughput-optimal ▼	Latency-optimal ▲	Clicked data
Latency for first token (ms)	442.9	21.0	[Click on a point]
Latency per generated token (ms)	N/A	N/A	[Click on a point]
Prompts per second per 8 GPUs	1138.5	47.1	[Click on a point]
Tensor Parallelism	1	8	[Click on a point]
Batch size	64	1	[Click on a point]
Model	LLAMA2-13B-base-1.nemo	LLAMA2-13B-base-1.nemo	[Click on a point]
GPU	NVIDIA H100 80GB HBM3	NVIDIA H100 80GB HBM3	[Click on a point]

Example 1

Smaller model – for auxiliary task

- We are looking for a sizable use case of **Llama-7B**. 128 in, 512 out.
- For input 128, output 512 we have **65.6** peak prompts per second per **one** DGX H100
- That's 53.2 requests per second on average
- That's 1.5M requests per working day (8 hours)
- 3 requests per person → **500k daily active users**
- 192M input, 768M output tokens per day

LLAMA2-7B-base-1.nemo, NVIDIA H100 80GB HBM3, input length: 128, output length: 512



Recommended Configurations Within Limits

Metric	Throughput-optimal ▼	Latency-optimal ▲	Clicked data
Latency for first token (ms)	258.3	42.2	[Click on a point]
Latency per generated token (ms)	14.8	8.7	[Click on a point]
Prompts per second per 8 GPUs	65.6	14.3	[Click on a point]
Tensor Parallelism	1	1	[Click on a point]
Batch size	64	8	[Click on a point]

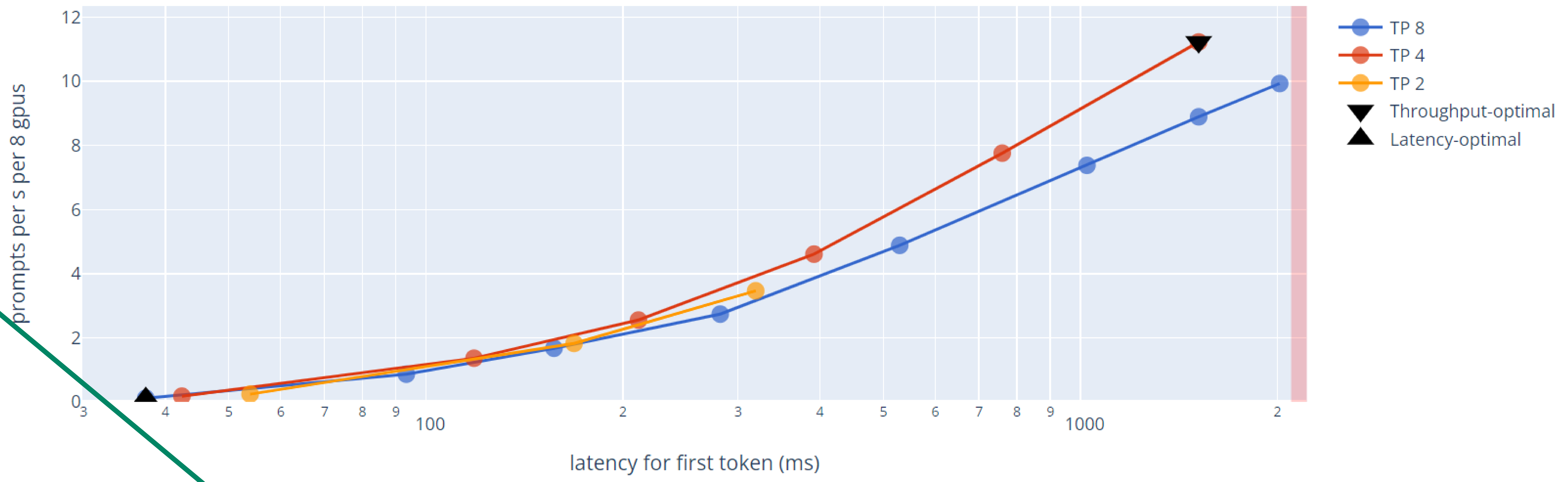
https://nemo-inference-sizing.nvidia.com/

Example 2

Larger model (ChatGPT like)

- We are looking for a sizable use case of **Llama-70B.** 128 in, 512 out.
- For input 128, output 512 we have **11.2** peak prompts per second per **one** DGX H100
- That's 6.92 requests per second on average
- That's 200k requests per working day (8 hours)
- 3 requests per person → **66k daily active users**
- 25.6M input, 102M output tokens per day
- $38.4 + $204 GPT 3.5 turbo per day (fair comparison) = $7.2K/month on OpenAI

LLAMA2-70B-base-1.nemo, NVIDIA H100 80GB HBM3, input length: 128, output length: 512



Recommended Configurations Within Limits

Metric	Throughput-optimal ▼	Latency-optimal ▲	Clicked data
Latency for first token (ms)	1516.7	37.3	[Click on a point]
Latency per generated token (ms)	41.6	17.2	[Click on a point]
Prompts per second per 8 GPUs	11.2	0.1	[Click on a point]
Tensor Parallelism	4	8	[Click on a point]
Batch size	128	1	[Click on a point]

https://nemo-inference-sizing.nvidia.com/

Example 3

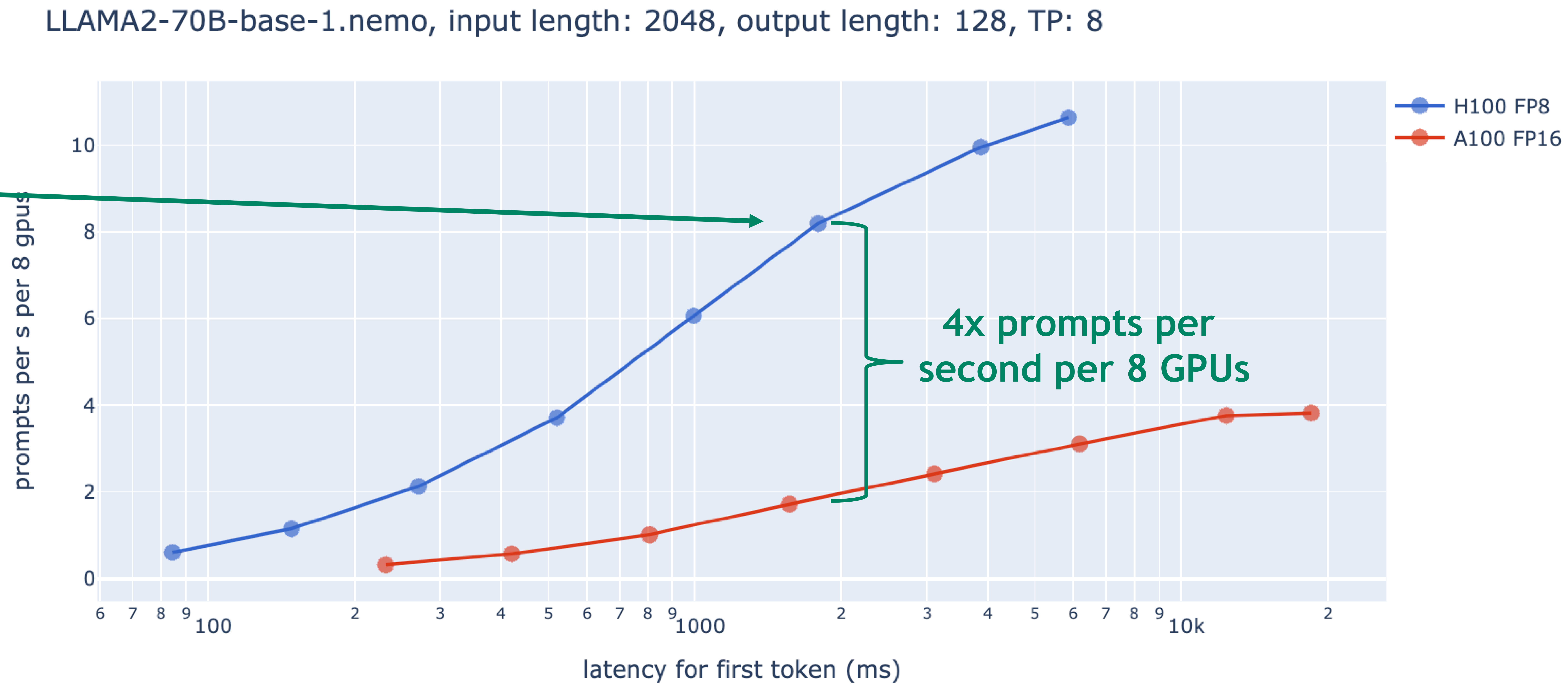
A100 FP16 vs H100 FP8

- H100 features a Transformer Engine with FP8 precision support

- For input 2048, output 128 and TP8, H100 with FP8 delivers **x4 prompts per second** compared to A100 with FP16, for the same latency

- Part of that increase is due to the FP8 format and part due to the better performance of H100 vs A100

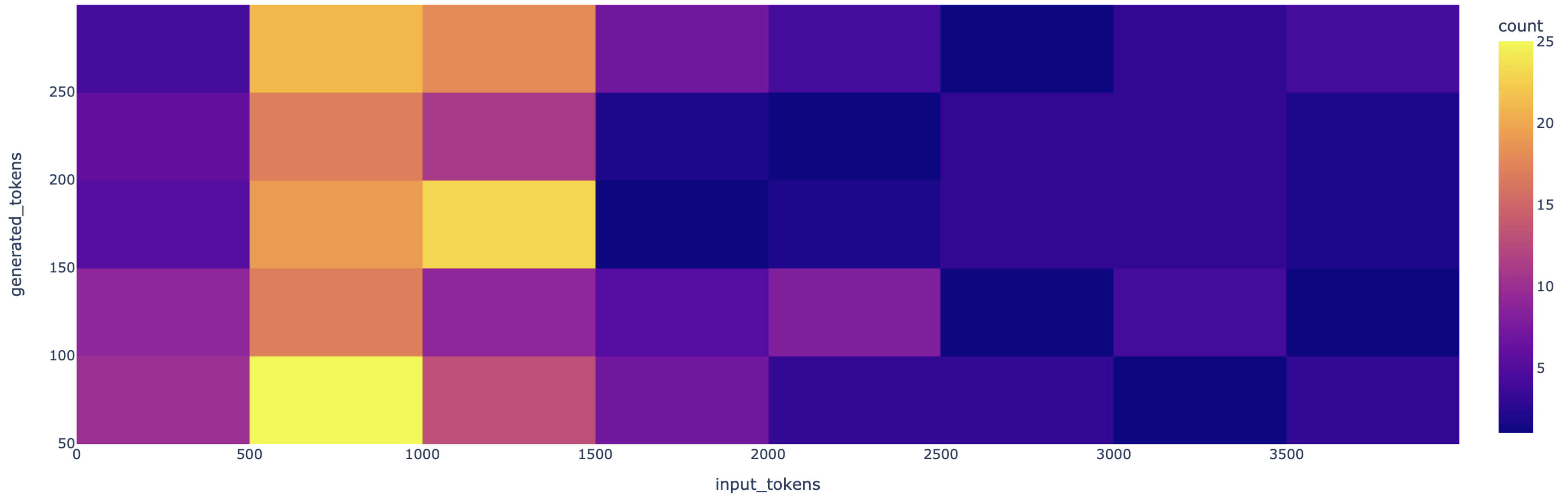
- Other efficient techniques like pruning, distillation or sparsification can increase performance

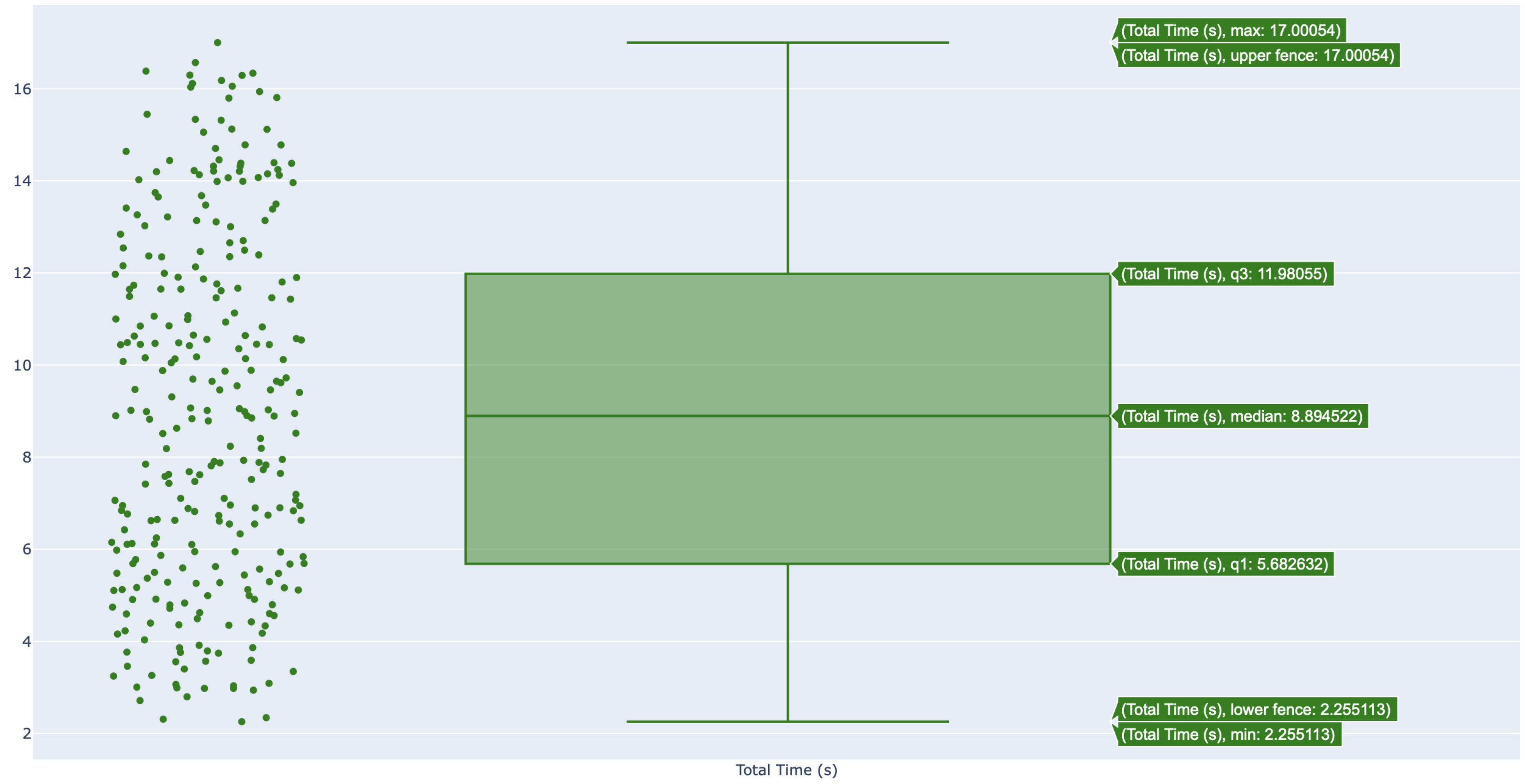


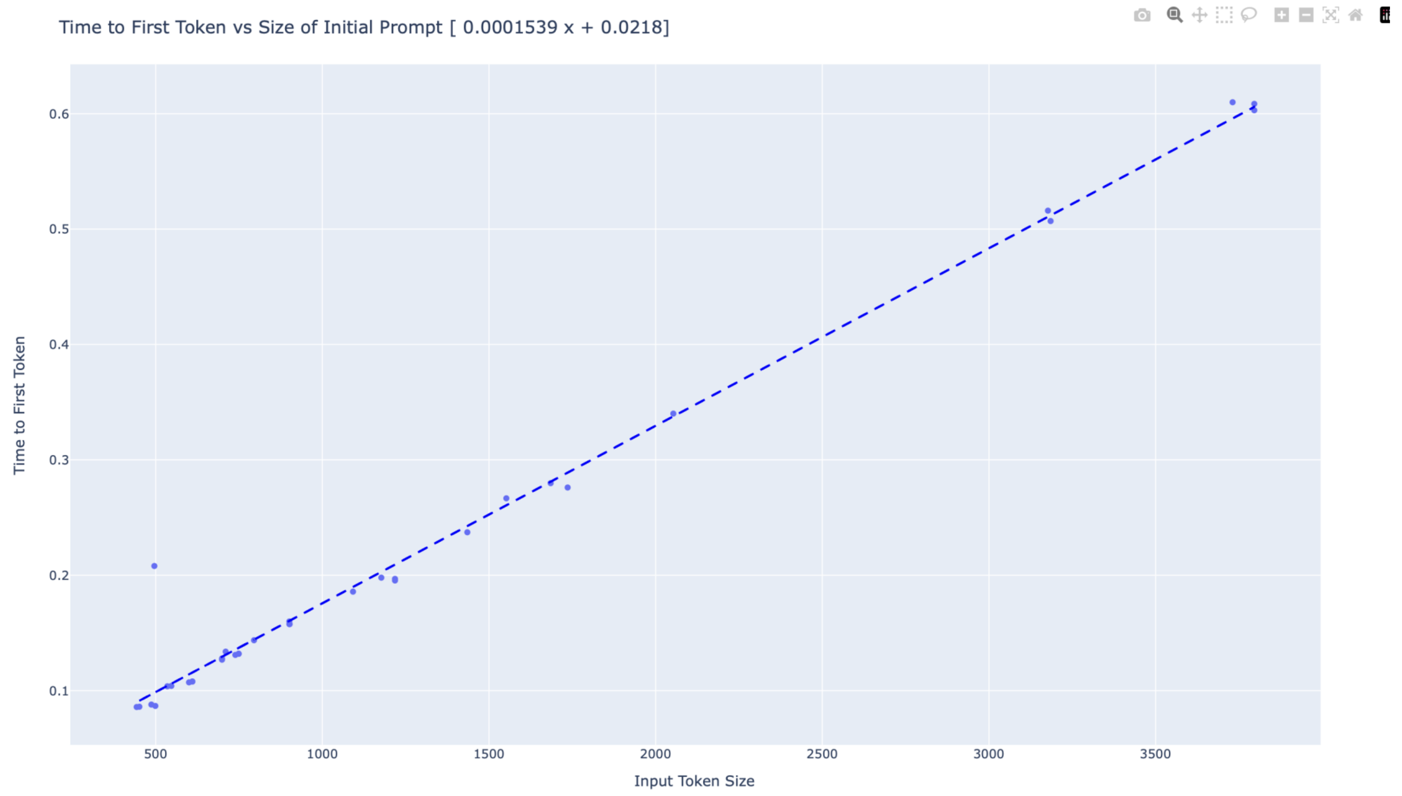
https://nemo-inference-sizing.nvidia.com/

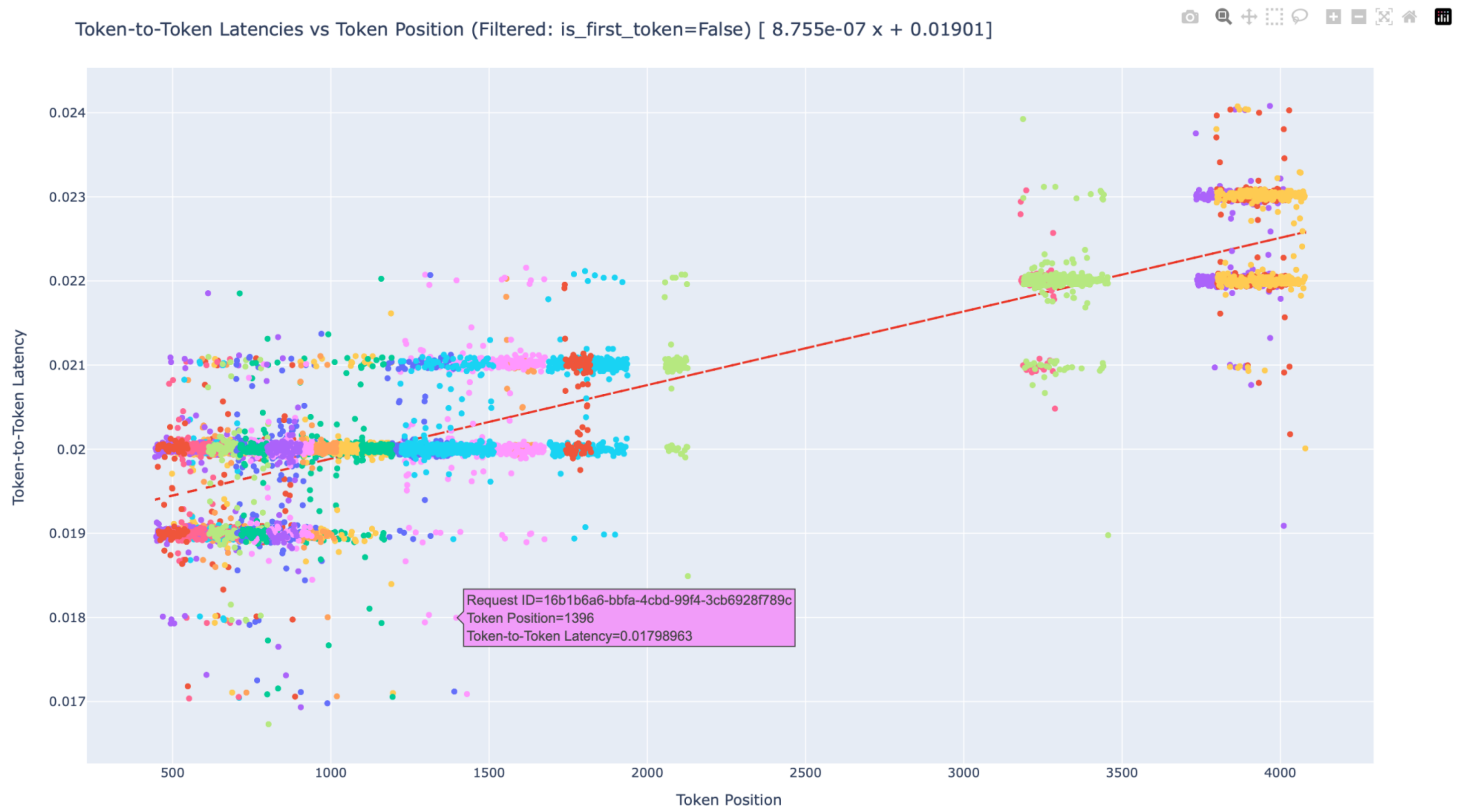
NeMo Inference Microservice Performance Tools

- Measure, Plan, Deploy
  - Load Generators
    - Concurrent Loadgen = MLPerf MultiStream Scenario or LLM Perf tool
    - Poisson Loadgen = MLPerf Server Scenario
  - Trace Analysis
    - Collect runtime statistics from Load Generators
    - Generate Visualizations of the collected statistics
  - Performance Models
    - Use collected statistic to build a predictive performance model
    - Digital Twin of the Production Deployment
  - Evaluate Scenario in Simulation
    - Tweak arrival rates and/or input/output and evaluate the performance of the virtual deployment.
    - Simulation is accurate to within 10-15% for small models and to within 3-5% for large models









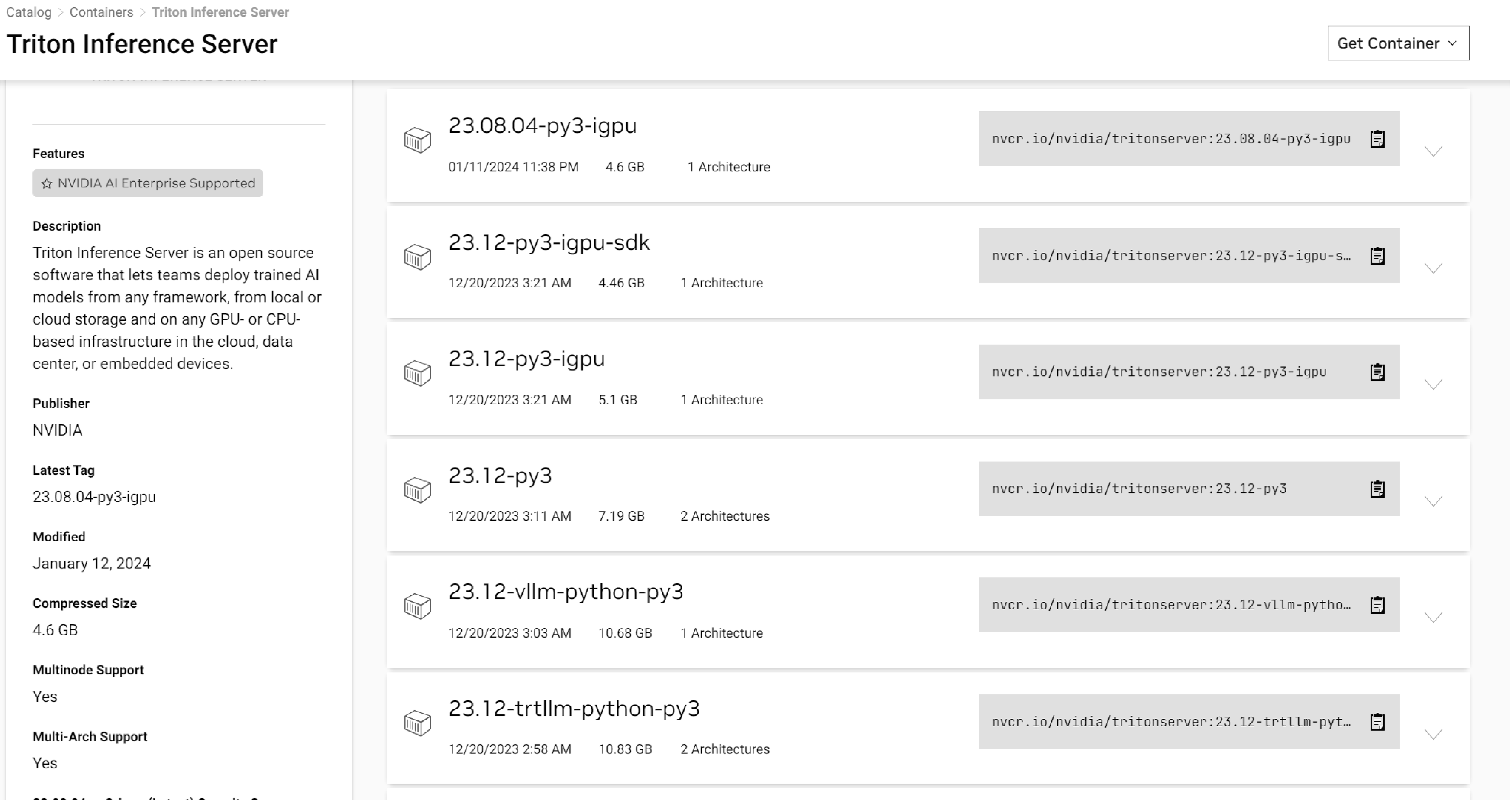

```
In [25]: print(f"actuall walltime: {full_loadgen_report.walltime:.2f}; simul
actuall walltime: 163.49; simulated walltime: 170.15; ratio: 0.96
```

Rules of Thumb for Sizing

- We estimate the sizing based on NVIDIA SW stack: NeMo, 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.
- New apps should be developed in streaming mode . To introduce LLMs into the older apps, one may use sequential mode.
- Locality of compute is not too important for the cloud deployments of the LLM.
  - Consider cheapest deployment across the world due to latency in seconds

Inference Containers

- NeMo Inference Microservice — fresh release
  - Supports OpenAI-compatible API — killer feature
  - Supports NeMo LLM Service compatible API
  - Accelerated by TRT-LLM
- Triton + TRT-LLM
  - Part of NVAIE and can be supported
  - Works with HF models
  - Open Source



Paper to Understand Inference

Chunked Prefill

- https://arxiv.org/pdf/2308.16369.pdf

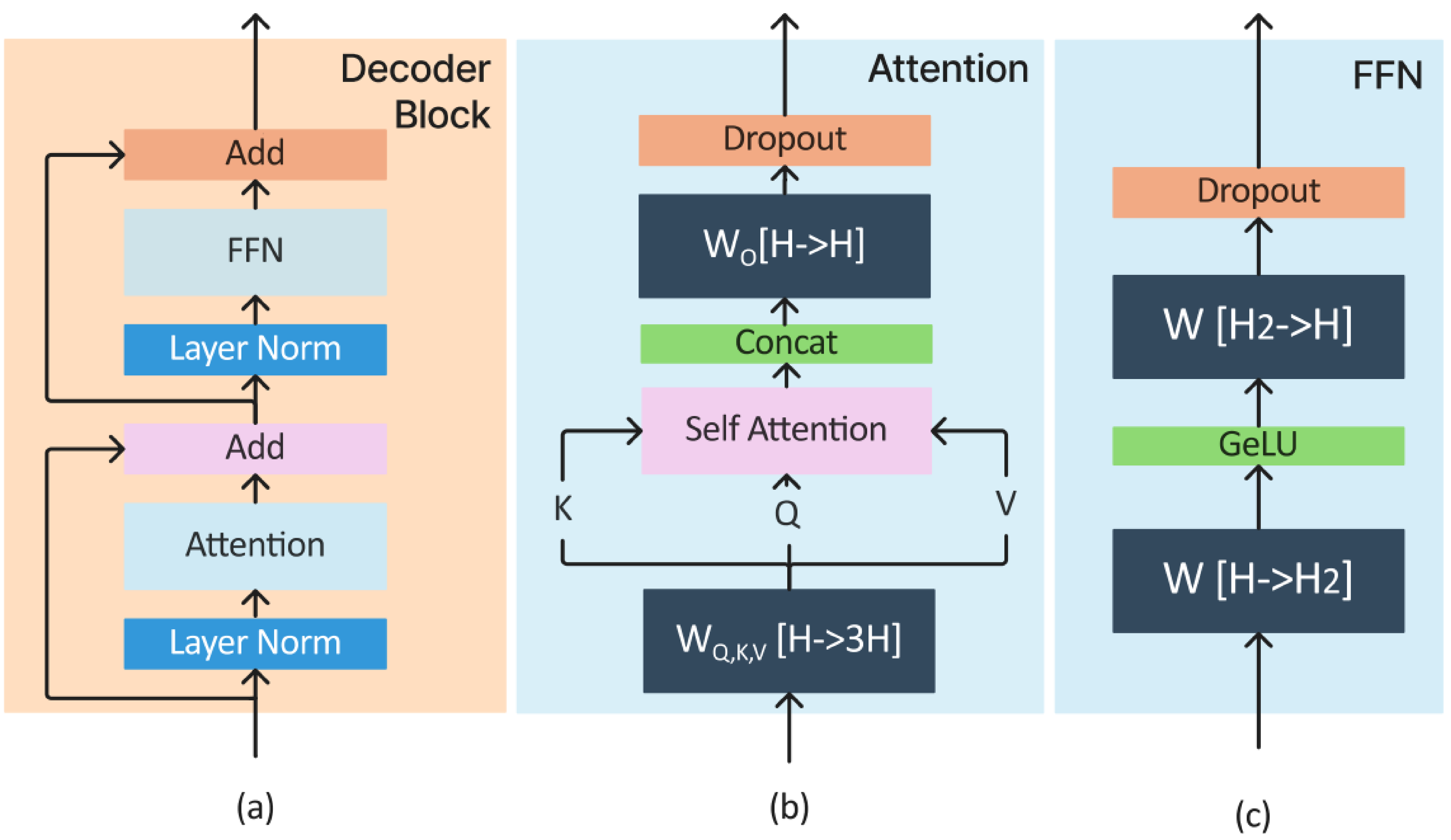


Figure 2: High-level architecture of a decoder block.

Operation	Shapes of tensors		
	Input(s)	Weight(s)	Output(s)
preproj	$[B,L,H]$	$[H,H]$	$[B,L,H]$
attn	$[B,L,H]$	-	$[B,L,H]$
postproj	$[B,L,H]$	$[H,H]$	$[B,L,H]$
ffn_ln1	$[B,L,H]$	$[H,H_2]$	$[B,L,H_2]$
ffn_ln2	$[B,L,H_2]$	$[H_2,H]$	$[B,L,H]$

Table 1: Shapes of the input, weight, and output tensors in a transformer decoder block. B, L and H denote batch size, embedding (aka hidden) size and sequence length (L=1 during decode, except for attention).

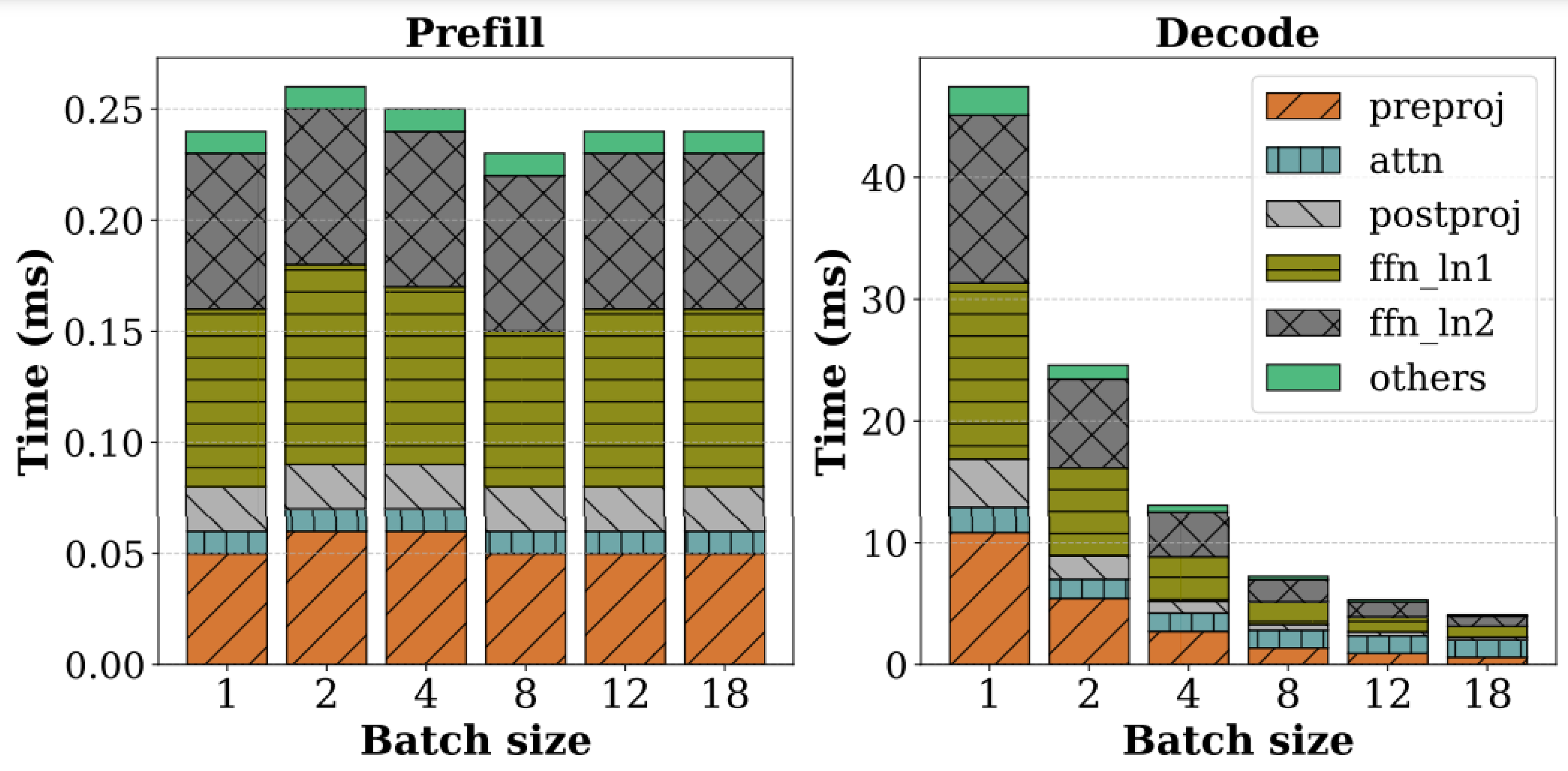
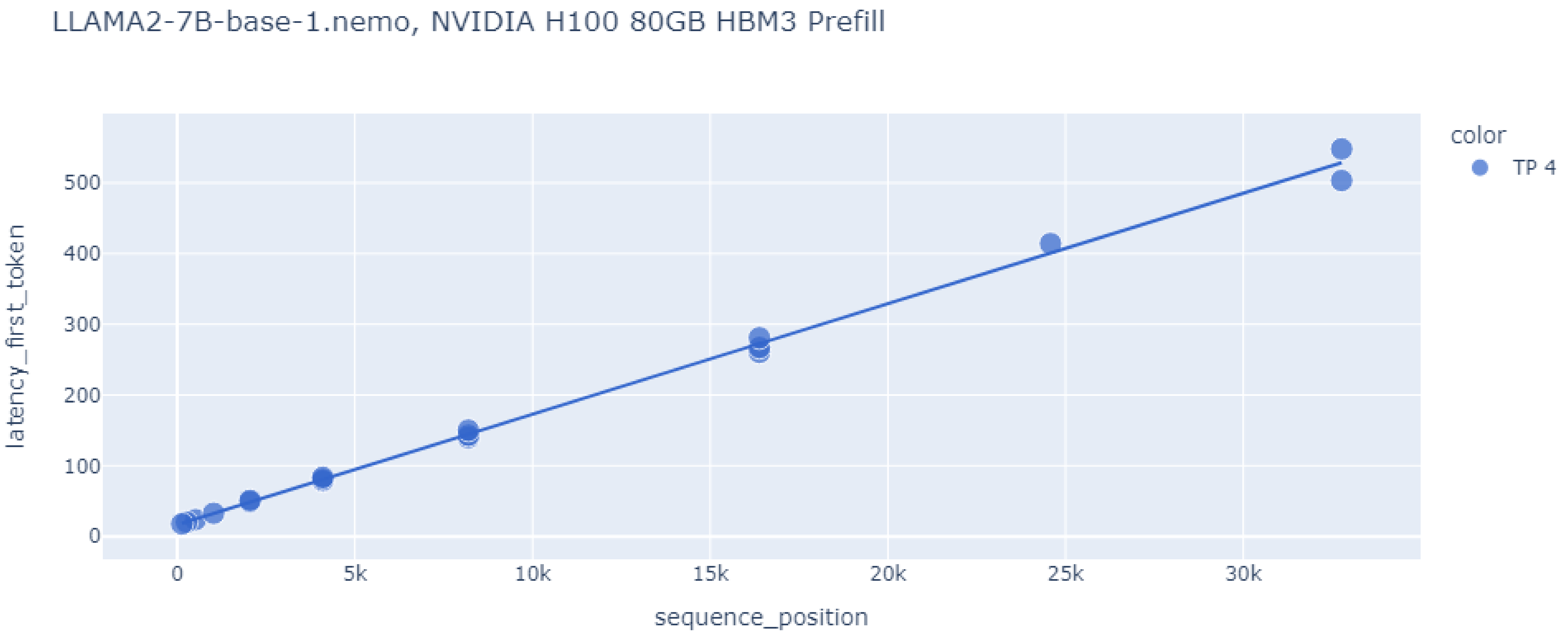


Figure 3: Per-token prefill and decode time with different batch sizes (sequence length = 1024) for LLaMa-13B on A6000 GPU. Prefill saturates GPU compute even at batch size of 1 and results in almost constant per-token time across batch sizes. Decode under-utilizes GPU compute and costs as much as 200× prefill for batch size 1. The incremental cost of linear operators for decode is almost zero as batch size increases. The attention cost does not benefit from batch size as it is memory-bound.

Prefill Is Compute Bound in FFN

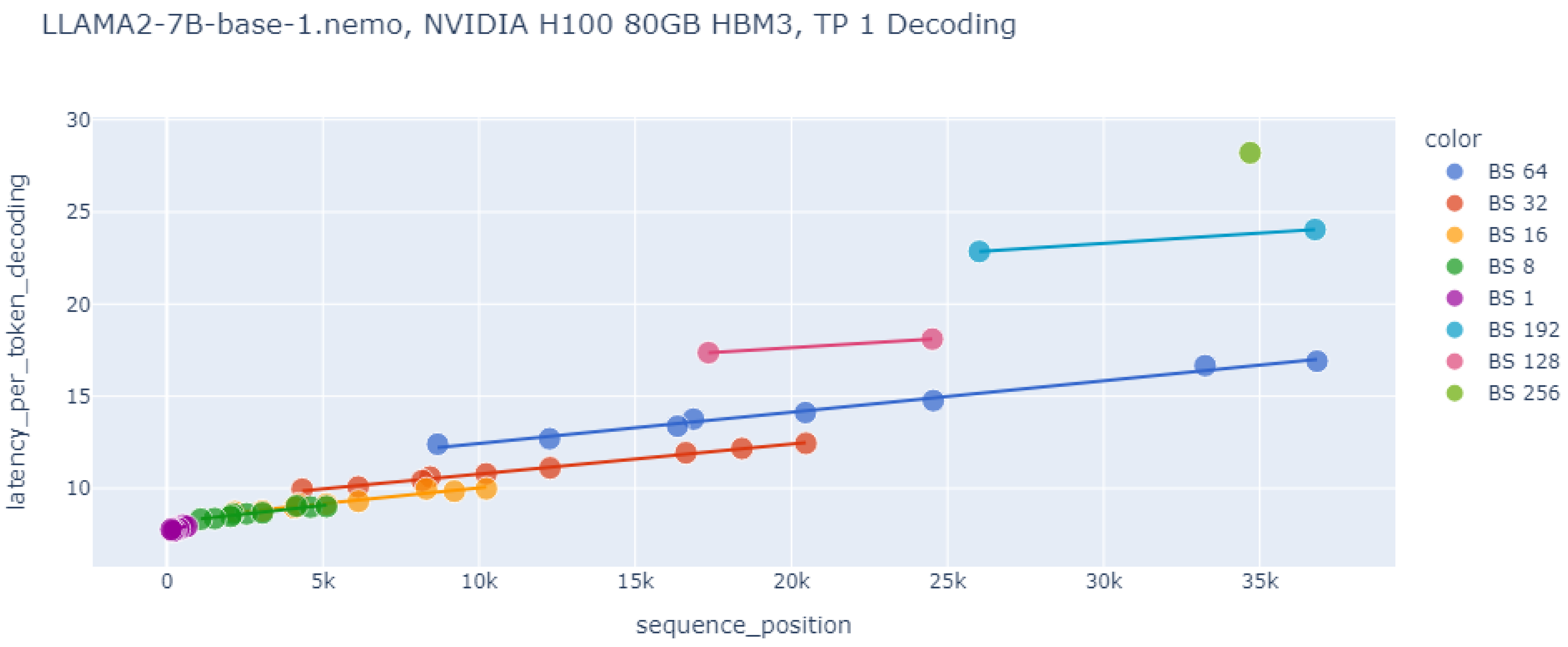
```
df_measured['sequence_position'] = df_measured['input_len']*df_measured['batch_size']
```



Decoding Is Memory Bound in KV Cache

But not so straightforward ☹

```
df_measured['sequence_position'] = (df_measured['input_len'] + (df_measured['output_len'] - 1)/2 )*df_measured['batch_size']
```



Current Limitations and Extrapolations

- Extensive measurement are currently available only for NeMo Inference Container
  - PyTriton + TRT-LLM Python Backend
  - Only BF16 (expect approx. 30% improvement from switching to FP8 on H100 and L40s)
  - Client-side batching
- We expect the in-flight batching results to be compatible with the results of the NeMo Inference Sizing Calculator benchmarks
  - We expect the throughput to be the same
  - We expect the decoding latency to be the same
  - We expect the prefill latency to be close to the measured latency of prefill batch size 1
  - Results in significant improvement in first token latency

Call to Action

Use the Personal Checklist to understand your requirements

- Use the sizing checklist for your use cases
- Get a ballpark estimate of required HW using the calculator: https://nemo-inference-sizing.nvidia.com/ (when it gets published)
- Need clarifications or help with sizing? Drop message to me and my colleague: Dmitry Mironov dmitrym@nvidia.com, Sergio Perez sergiop@nvidia.com

Dmitry Mironov
dmitrym@nvidia.com

Sergio Perez
sergiop@nvidia.com

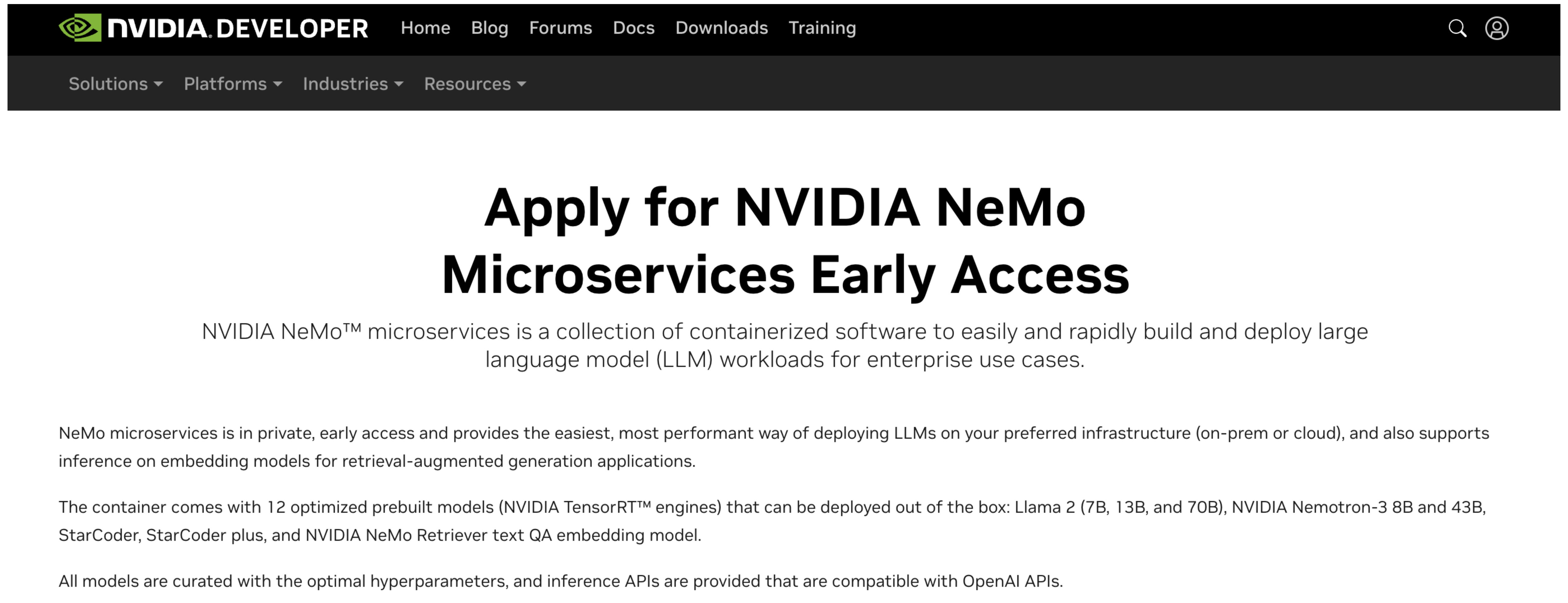
Inference Resources

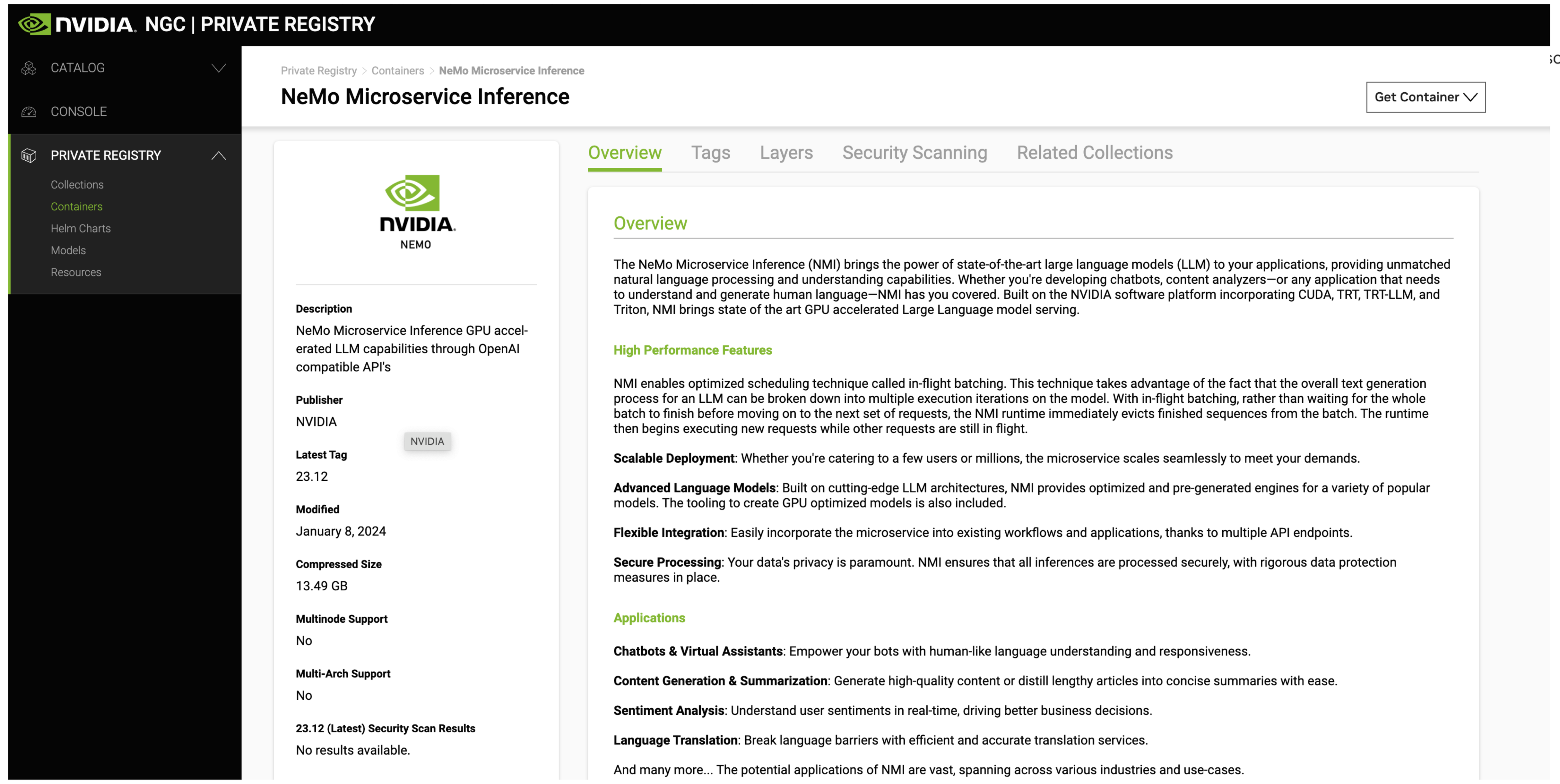
- http://nemo-inference-sizing.nvidia.com/
- NVIDIA NeMo Microservices Early Access
- OpenAI pricing + token counter
- What is RAG — NVIDIA blog
- Mastering LLM Techniques: Inference Optimization — NVIDIA Blog

LLAMA2-70B-base-1.nemo, NVIDIA H100 80GB HBM3, input length: 128, output length: 512

Recommended Configurations Within Limits

Metric	Throughput-optimal ▼	Latency-optimal ▲	Clicked data
Latency for first token (ms)	1516.7	37.3	[Click on a point]
Latency per generated token (ms)	41.6	17.2	[Click on a point]
Prompts per second per 8 GPUs	11.2	0.1	[Click on a point]
Tensor Parallelism	4	8	[Click on a point]
Batch size	128	1	[Click on a point]





NVIDIA