Hello, and welcome to this course about sizing LLM inference systems.

In this course you will learn how to estimate the number of GPUs needed for your inference workload.

Before we start, we'd like to remark that you don't need a deep expertise of LLMs to benefit from the course
We've prepared ready-to-run notebooks with all the commands and assets for you. Once you click on the start buttom, you will access the course environment with the notebooks. We recommend you click on "start" now since it takes a few minutes to load.
Before jumping into the notebooks, let me introduce the instructors of this DLI and give you an introduction of the contents of the course.

About Us

Dmitry Mironov, EMEA

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

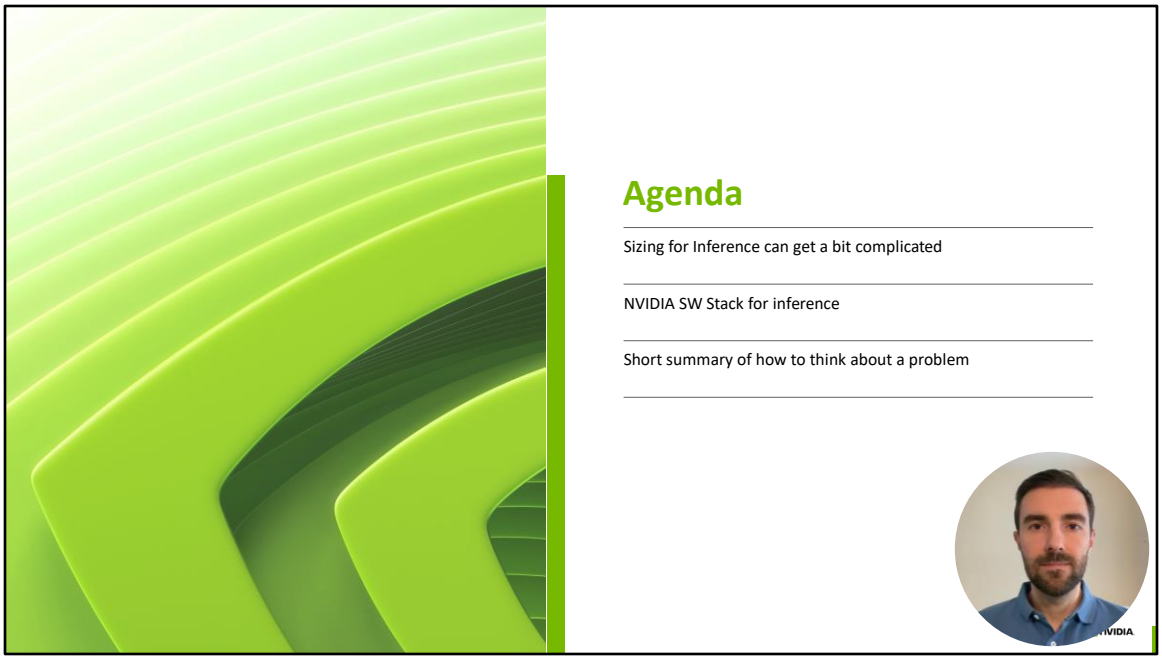
Sergio Perez, EMEA

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

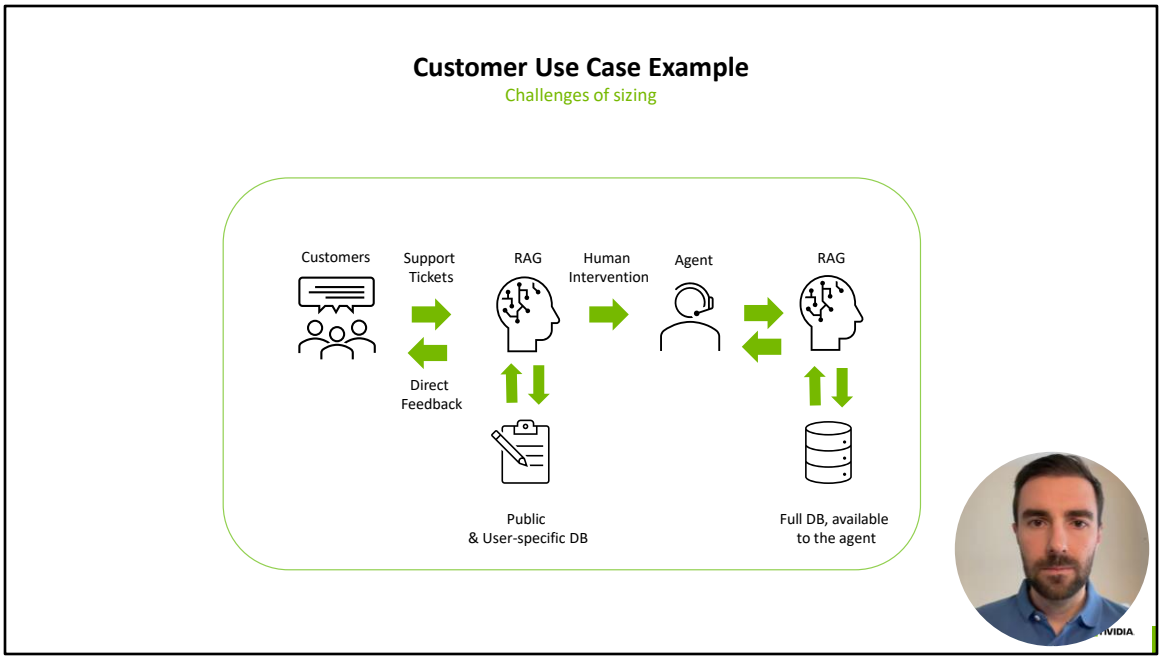
Let me tell you some background about Dmitry and myself, since we'll be your instructors during this course.

Dmitry is senior deep learning solutions architect at NVIDIA, where he supports deployment of AI and deep learning solutions. He's definitely the person to go to if you want to optimize your inference workload for production!

And I am Sergio, also a senior deep learning solutions architect at NVIDIA. I work with companies across sectors to implement Generative AI applications, and my area of expertise is model compression with techniques like quantization.



First, I'm going to provide you with an example to understand why inference can get a bit complicated. I will cover some discovery questions so that your inference estimation is as accurate as possible. Afterwards I'll describe the software stack available at NVIDIA for inference. Finally, I'll give you an example of inference sizing and some general tips.



Let’s look at a customer use case example.

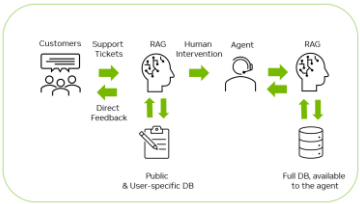
This is a normal workflow found in a customer service center. First, the client submits a support ticket which is received by the AI chat bot. This bot has access to a database with public and user specific data. It tries its best to answer the question but if it can’t solve the user’s problem then the system will escalate the issue to a human agent.

The agent reviews the information from the chatbot and then attempts to solve the end-user's issue.

In this example, the agent has access to a database with public and private info specific to the caller. They use this to help address the user’s question and close the ticket.

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

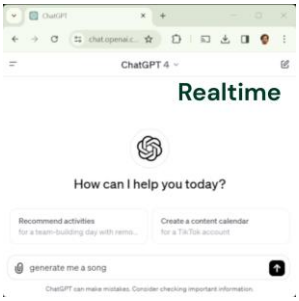
The customer needs 13 DGX H100 systems

- 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



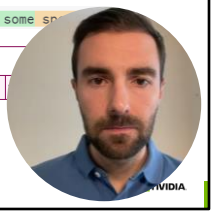
Prefill: 1.14s, 5 input tokens, 1 output token
`generate me a song`

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

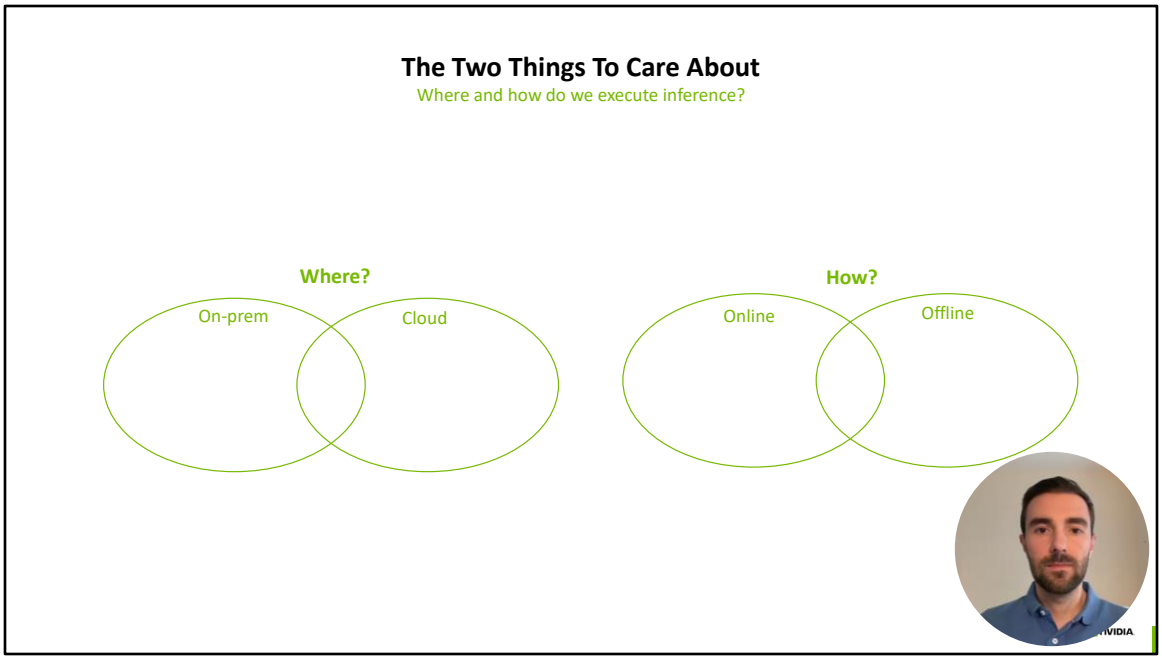
user prompt



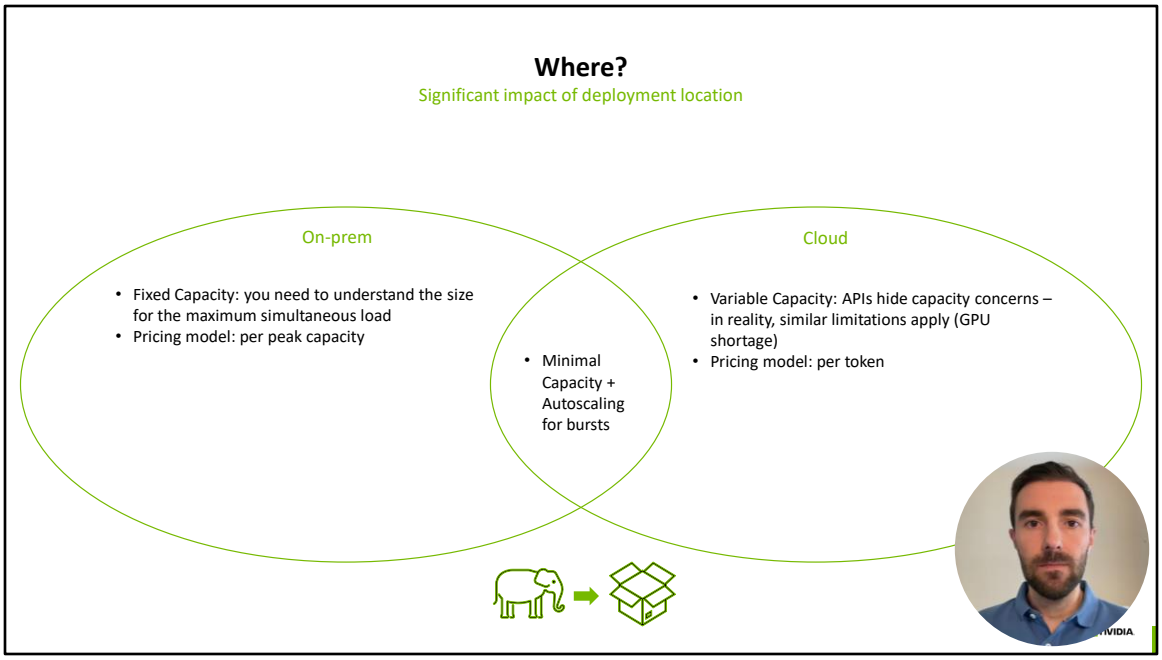
There are two stages of an LLM that you need to be aware of. The first is Prefill and the second is Decoding.

Prefill is the time between pressing enter on your device and the first output token appearing on your screen. Decoding occurs when the other words in my search are generated.

Usually, in most requests Prefill takes less than 20% of the end-2-end latency, while decoding takes more than 80%. That shows how important it is to send the tokens back to the client as soon as they're generated. Such implementation is called streaming.



For inference sizing, it’s important to understand where and how you are executing inference.
Concerning the ”where”, you could have your GPUs on-prem or on the cloud.
And about “how to execute inference”, you could have an online application responding live to a customer, or an offline application without latency requirements.
Let’s dive deeper into these questions.

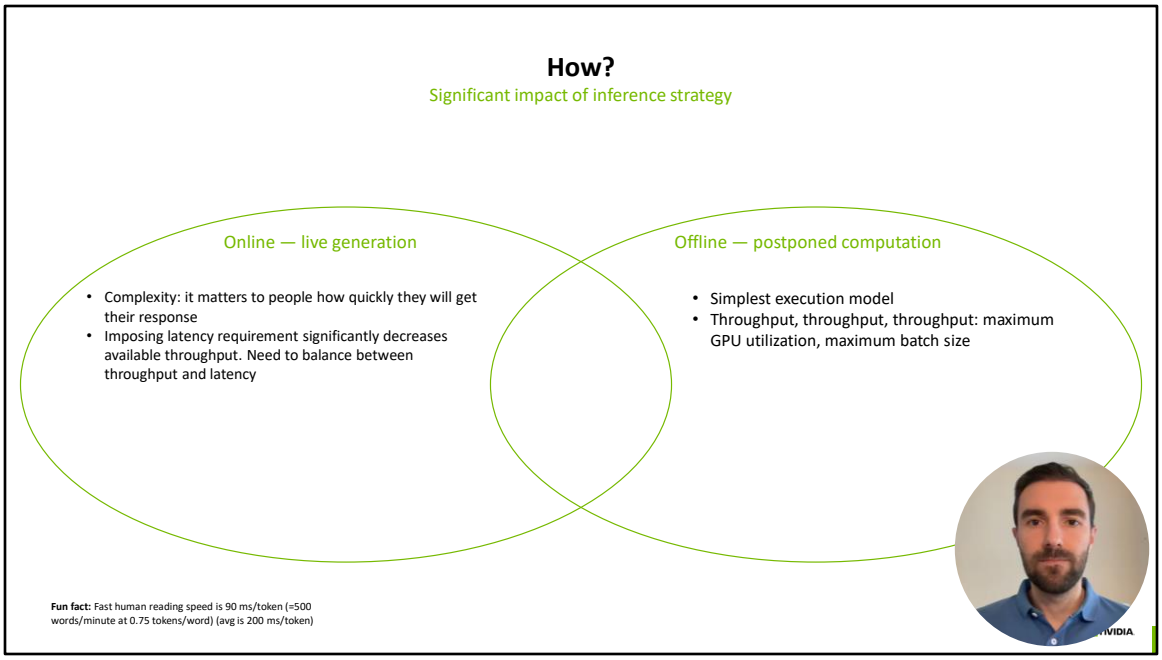


Where you deploy your GPUs can have a significant impact.

In an on-prem scenario, you have a fixed peak capacity equal to the number of GPUs available in your datacenter. The pricing model is per peak capacity, and this means that you are paying for all the GPUs even if you are not using them.

In the cloud, there is a variable capacity and you have some margin to scale the number of available GPUs if needed. In addition, Generative AI services on the cloud usually charge per token.

For both approaches, our recommendation is to start with a minimal capacity of GPUs and leave some room to autoscale for bursts of demand.



Concerning the how to execute, let's start with offline scenarios.

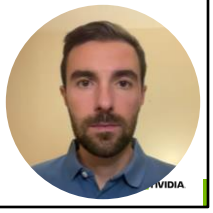
This case is rather simple. You know your inputs, you know your output sizes, and then you just need to optimize for throughput. Latency is not relevant here. Firstly, check if your LLM fits on one GPU, and if not apply tensor or pipeline parallelism to make it fit in the minimum number of GPUs. Once it fits, just increase the batch size to be as large as possible. That's the technique to achieve the highest throughput.

But for the online use case, there is a big caveat. And this caveat is the trade off between throughput and latency. This will be a topic across this course. But in short, if you impose latency requirements for your inference, then you have to trade throughput. latency requirements are not for free. Actually, there are two distinct flavors of latency requirements, let's examine them.

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



When discussing about latency requirements, you need to understand the difference between streaming and sequential mode.

In streaming mode, you can generate tokens one at a time and return them to the end client, like you saw in the ChatGPT example. In this mode, you care about time to first token, because this is the time during which the customer is waiting for the first token. Afterwards, the following tokens are generated much faster, and the rate of generation is usually faster than the average human reading speed.

In sequential model, you cannot stream the tokens. You have to wait for the end result, and as a result you care about the end to end latency. This is the time to produce all the tokens in the output sequence.

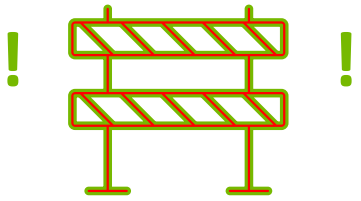
In the notebooks, you will learn to optimize for both time to first token and end to end latency.

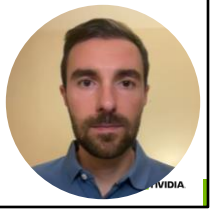
Questions to start sizing for inference

Let’s review now some of the questions to ask if you are starting to size an inference application.

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?





We recommend that you gather the following information before starting to estimate the inference performance:

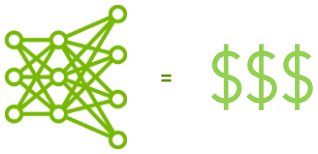
- The models intended to use for this use case.
- The average number of tokens per prompt.
- How many tokens in their LLM output.
- The request per second
- Latency requirements
- Finally, the GPUs you intend to use.

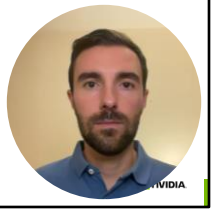
Let's cover some of those categories in the following slides.

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



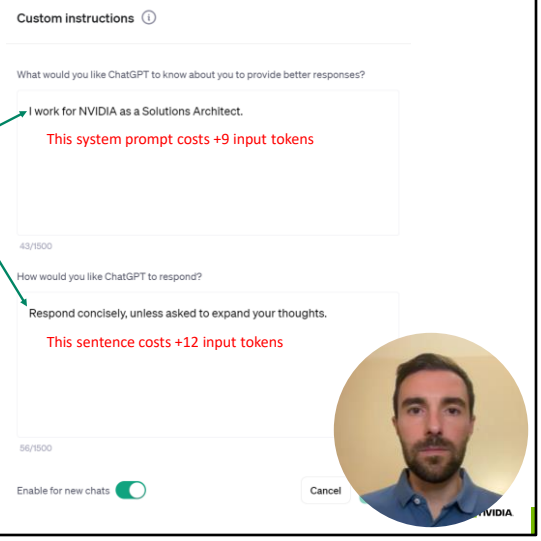


The size of the model can have a huge impact in the inference performance. At a high level, the bigger the model, the slower and more expensive the inference is. Smaller models can have really good quality for particular tasks while reducing the inference cost, so we recommend that you explore them too.

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



The input length, sometimes denoted as context window, can impact the inference performance. Bear in mind that words are converted to tokens. Every model runs on tokens in and tokens out. Usually, models like Llama run on about 4k-8k tokens or roughly 3000-6000 words in English.

In particular applications like RAG pipelines, the input length could be particularly large since you are adding chunks of documents into it. In those situations, the first-token latency is large since the input length is significant.

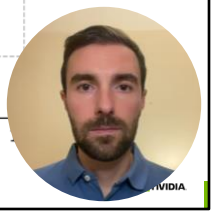
Peak Requests Per Second

- 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
```



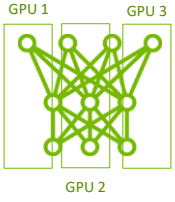
Sizing for on-prem is always sizing for the peak. If the system is not running at peak, the effective cost per token grows. This is especially important, if peaks are very high, relative to the average.

LLM Inference Requires Multiple GPUs

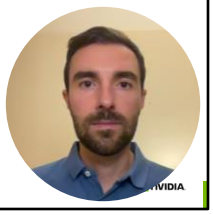
Tensor Parallelism (TP) – so how to split your neural network

- 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

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



Time = $



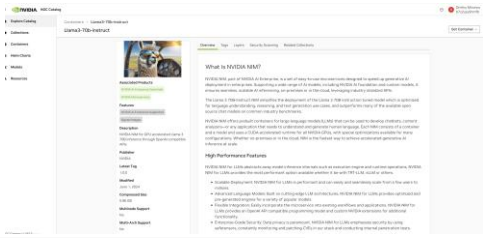
For larger models, the available memory within 1 GPU may not be enough. In those cases, you can use tensor parallelism to split the model across several GPUs. We recommend using GPUs with NVLink in these scenarios. There are other types of parallelism, like pipeline or sequence parallelism, that can also alleviate the memory footprint of LLMs during inference.

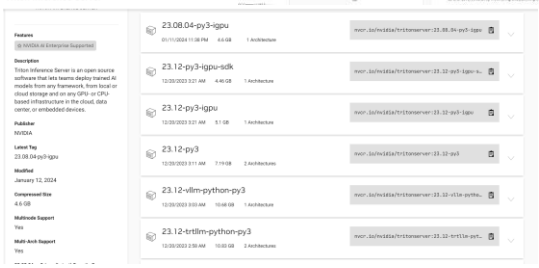
Tools Available

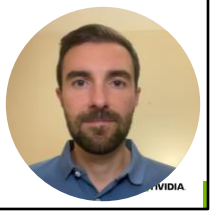
Let’s now move into the tools available for LLM inference.

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





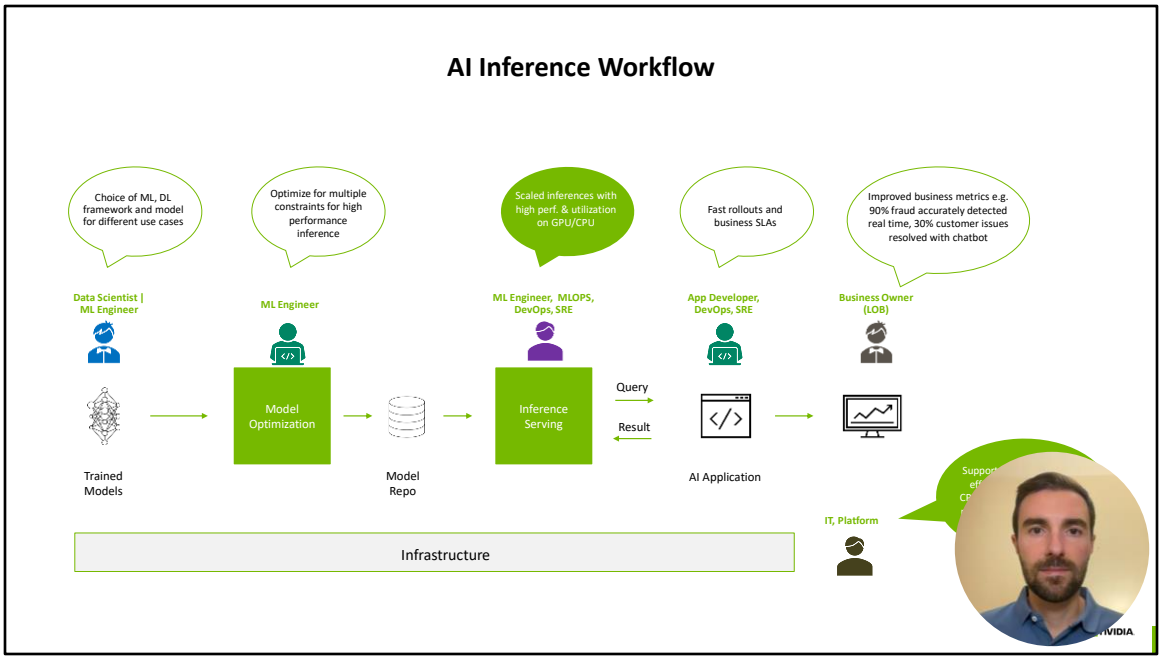


At NVIDIA, we provide containers to developers so that you can start using GPUs as quick and easy as possible. In particular, let me tell you about two containers that you can try if you are thinking about running inference workloads.

The first isTriton plus TensorRT-LLM. You can find these two libraries on NVIDIA's Github. Many developers prefer a more hand-on approach and leverage these tools to optimize inference workloads.

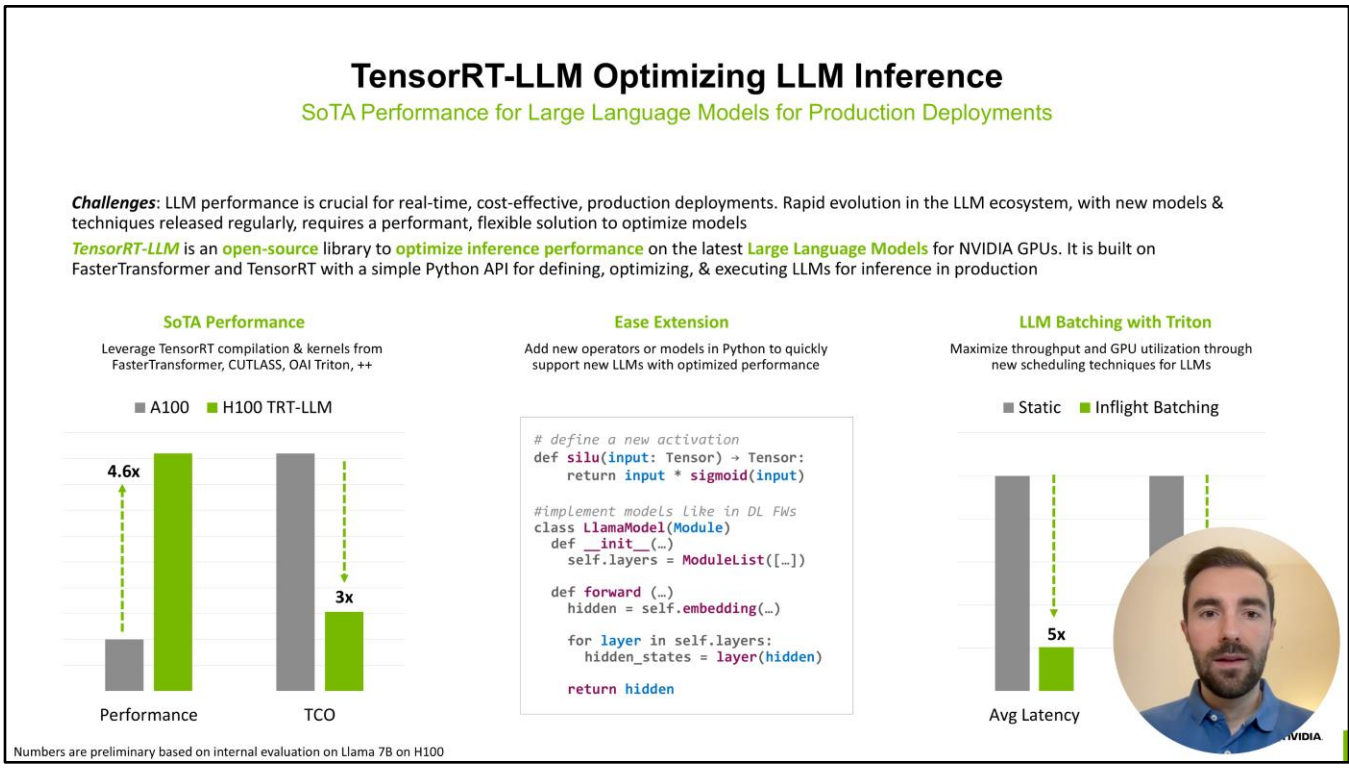
The second one is NIM, which stands for NVIDIA inference microservice. It's a great solution if you want to easily deploy LLMs as microservices in a few minutes.

Let's discuss about both approaches a bit deeper.

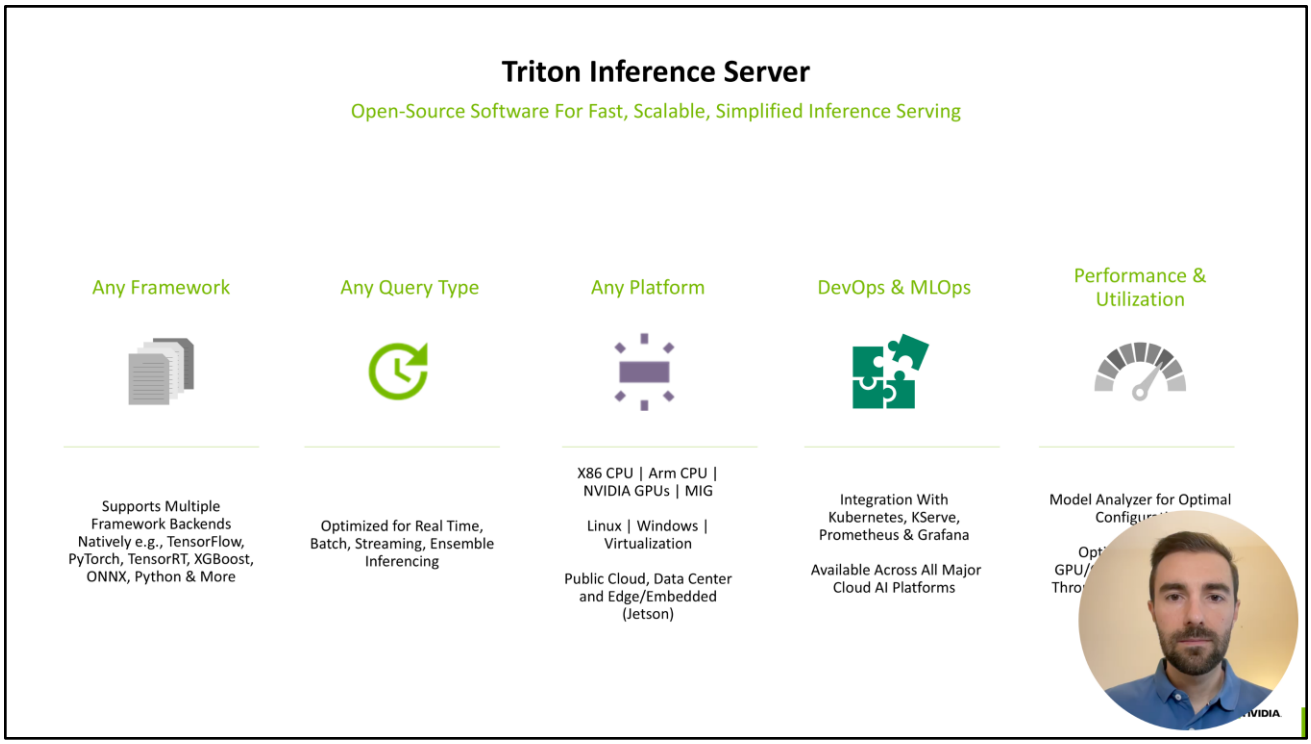


To understand TensorRT-LLM and Triton, let's discuss about the inference workflow. After training a model, it's time to optimize it for inference.
This is where tools like TensorRT-LLM play a big role. The model is optimized for a specific hardware and target constraints. For example, throughput, latency or memory constraints.

Once a model is optimized, it is stored in a model repository and then inference serving takes place.
Inference serving is the process where the model is loaded into the memory and then is ready for running. Triton is our recommended inference server.
The applications then query the inference server, which is in charge of batching all those queries and running the model. Finally, the inference server sends back the outputs to each of the clients.



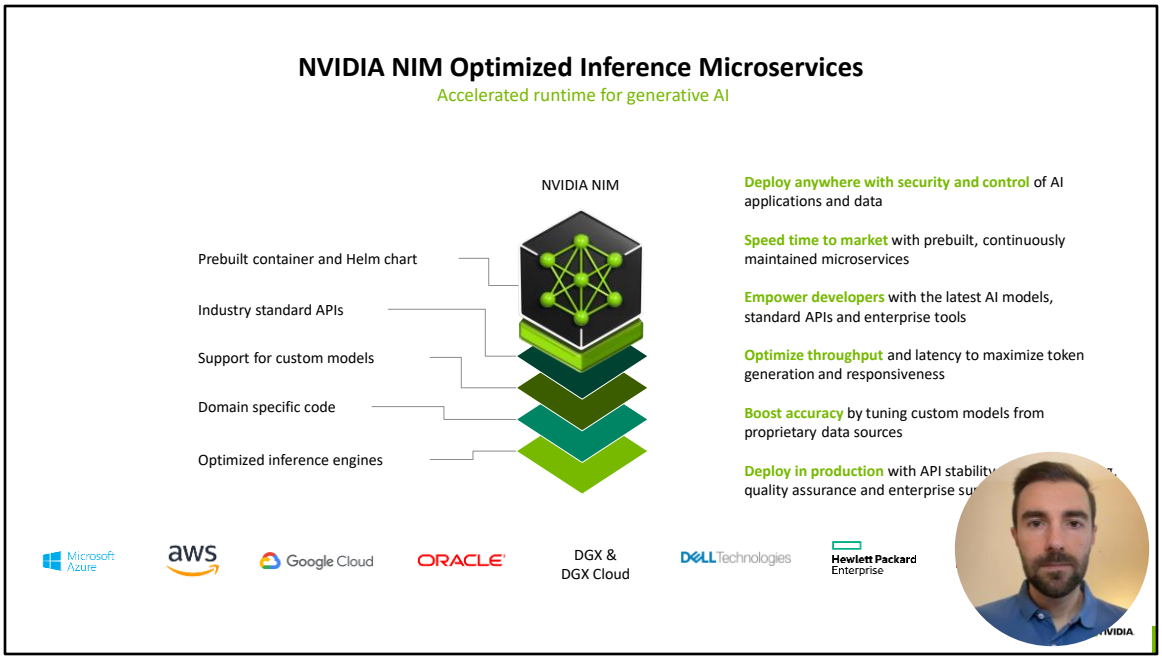
TensorRT-LLM is an open-source library to optimize LLMs and build inference engines. It offers SoTA performance for your particular available hardware. We already support most of the popular models, but you can also add new ones easily. Finally, TensorRT-LLMs offers great optimizations like inflight-batching or quantization.



Triton Inference Server is an open source software for fast, scalable and simplified inference serving.

In this slide you can read about some of its advantages. It supports any framework, such as PyTorch, TensorFlow and many others. You can pass it any quary type, like the streaming or sequential models we discussed before. It runs inference across many platforms, not just GPUs. It has integration with DevOps and MLOps tools, with integration with Kubernetes and others.

Triton not only delivers all of those features out of the box, but it doesn't sacrifice the performance. It shows very high performance on both CPUs and GPUs, and through features like dynamic batching and current model execution. It also has a tool called Model Analyzer that looks for an optimal configuration for deployment.



NVIDIA NIM provides a simple way to deploy AI models. It offers many advantages, and in this slide we list some of them.

For instance, it offers compatibility with standard APIs. If you have built your application using a different Generative AI service, you can just swap the endpoint url to point to NIM.

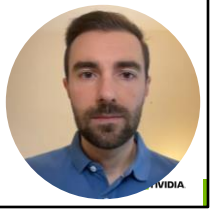
NIMs are also optimized to run efficiently for inference. In this DLI, you are going to learn to optimize inference workloads for your particular latency and throughput requirements. However, with NIMs we give you those optimizations out-of-the-box, so that you can quickly deploy your LLM.

We do have many NIMs available for models in language, vision or biology, so do check them out. With our developer program, you can access them for free. You can also leverage NIMs for production via NVIDIA's AI Enterprise license.

In this course, you will have the chance to use NIMs too to measure their performance.

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

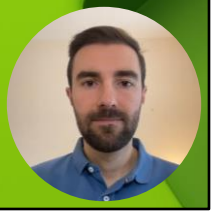


We also offer tools to measure the latency and throughput of your inference workloads.

Our recommended tool is genai-perf, which is developed by our Triton team. You will have the chance of using it during this course, and it's great to measure latency and throughput of NIMs and any other inference endpoint.

We also have other tools in TensorRT-LLM and Triton that you can try, so do check out the links provided in this slide.

Example of benchmarking



Let me give you an appetizer of the types of benchmarking plots you'll produce during this course. Dmitry will dive much deeper about them later in the course, but it's good that you start familiarizing yourself with them.

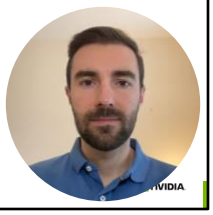
Example with Llama 3 8B

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



The plot that you see here is a classical latency versus throughput plot for inference. The plot corresponds to small model Llama 3 8 B, with 2000 input and 200 output length, on a DGX H100. We have also imposed a maximum time to first token of 0.5s.

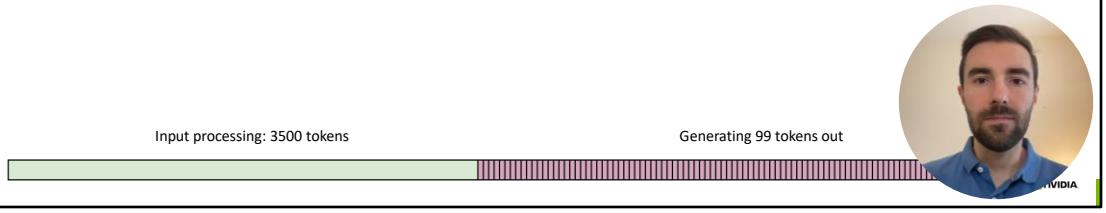
First thing to remark in the plot is that there is a tradeoff between latency and throughput. Each color denotes a different configuration, and each dot in the lines a different concurrency level. For the time being, just consider that you have chosen a particular dot in this plot. I've marked it with a downwards triangle. Dmitry later will explain how to make a decision about the dot to choose.

For that dot, you get that the throughput is 70.7 prompts per second. From this number, you can compute that we can serve 679k daily active users, under some assumptions described in the slide.

This number of daily active users is served with only one DGX H100. If you want to serve more users, If we need to add more DGXs working in parallel.

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



Before concluding the presentation, let me give you some rules of thumb for sizing. First, make use of NVIDIA software stack: NIM, TensorRT-LLM or Triton are great tools to run inference workloads.

Second, for big models, make sure you use nv link enabled system.

Third, in the streaming mode the time to first token is determined by the input length.

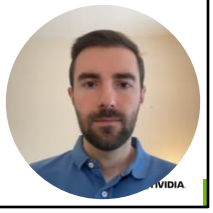
Next, remember that cost and latency are dominated by the number of output tokens. Input tokens are usually much cheaper than output tokens. You can check an example about this in the slide.

For the next rule, remember the trade off between throughput and latency. If you limit first-token latency, you will decrease throughput.

Finally, larger models require more memory and have higher latency, so bear that in mind when choosing your 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/



Let me conclude this presentation by giving you some references of various resources to check. Our NIM for LLM Benchmarking Guide is a great document to understand inference benchmarking. Other resources to check are our NIM docs, the session that Dmitry and I gave at GTC, and our series of blogpost about inference optimization.

Now it's your chance to begin working on the notebooks. Click on "start" to launch the jupyter lab environment.

See you there!


NVIDIA.