TensorRT-LLM

October 2023

TensorRT-LLM Optimizing LLM Inference

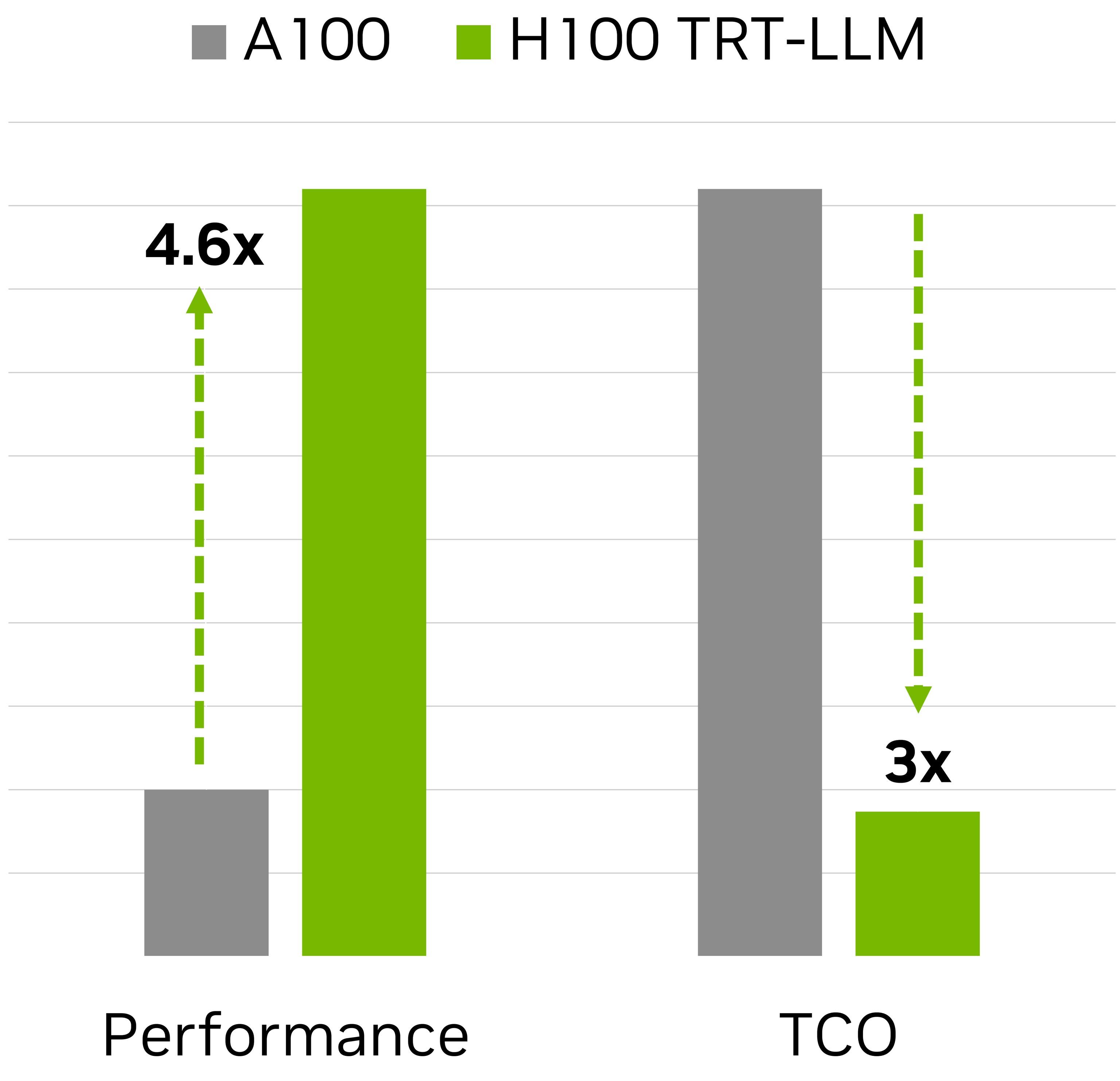
SoTA Performance for Large Language Models for Production Deployments

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 FasterTransformers, CUTLASS, OAI Triton, ++



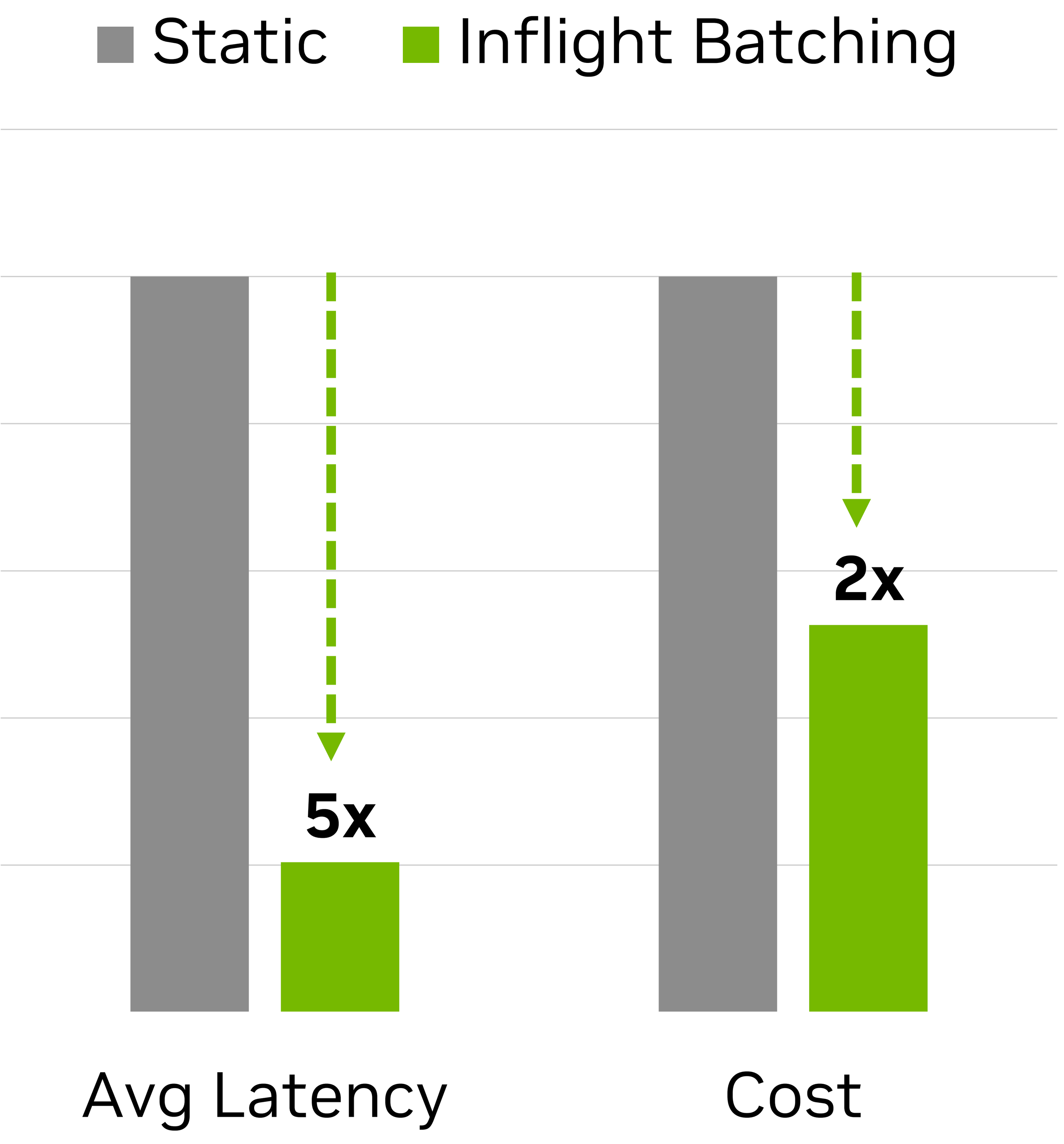
Ease Extension

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

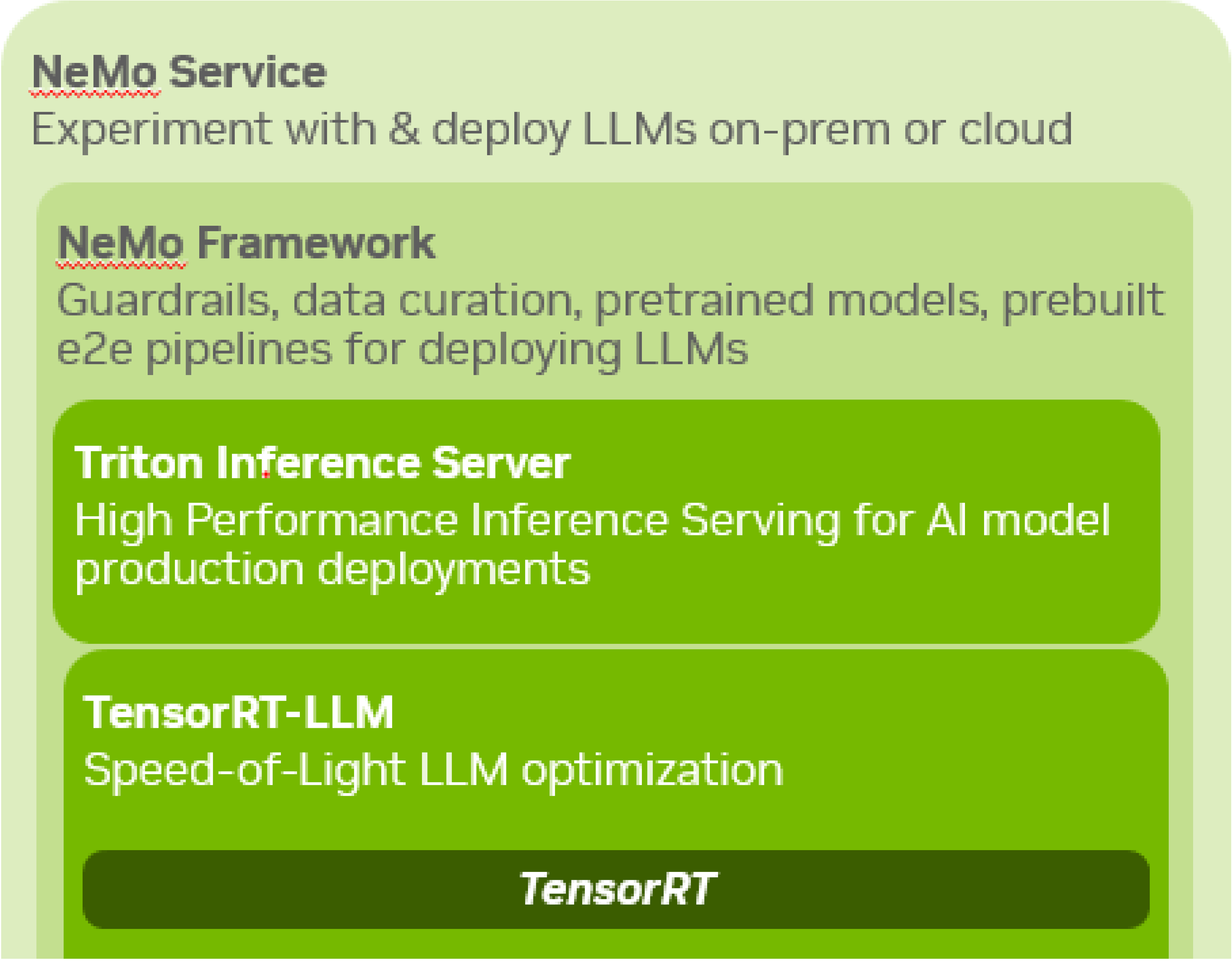
Maximize throughput and GPU utilization through new scheduling techniques for LLMs



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

TensorRT-LLM in the LLM Ecosystem

Start with NeMo Framework for optimized inference, TensorRT-LLM for detailed control



TensorRT-LLM in the *DL Compiler* Ecosystem

TensorRT-LLM builds on TensorRT Compilation

- **TensorRT-LLM**
  - Built *on-top of* TensorRT
    - Leverages TensorRT for general graph optimizations & fast kernels
  - Adds LLM specific optimizations:
    - KV Caching & Custom MHA Kernels
    - Inflight batching, Paged KV Cache (Attention)
    - Multi-GPU, Multi-Node
    - *& more*
  - *ONLY for LLMs*

TensorRT

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

TensorRT-LLM Usage

Create, Build, Execute

- Instantiate model and load the weights
  - Load pre-built models or define via TensorRT-LLM Python APIs
- Build & serialize the engines
  - Compile to optimized implementations via TensorRT
  - Saved as a serialized engine
- Load the engines and run optimized inference!
  - Execute in Python, C++, or Triton

0. Trained Model in FW
NeMo, HuggingFace, or from DL Frameworks

1. Model Initialization
Load example model, or create one via python APIs

2. Engine Building
Optimized model via TensorRT and custom kernels

TensorRT-LLM Engine | **TRT Engine** | **Plugins**

3. Execution
Load & execute engines in Python, C++, or Triton

TensorRT-LLM Usage

Running Llama in TensorRT-LLM

[TensorRT-LLM Llama Getting Started Blog]
[TensorRT-LLM Examples]

- Install and Build TensorRT-LLM from Github
- Download the pre-trained model
- Compile the model in TensorRT-LLM
- Run Optimized Inference!

```
$ python3 examples/llama/build.py \
    --model_dir meta-llama/Llama-2-7b-chat-hf \
    --dtype float16 \
    --use_gpt_attention_plugin float16 \
    --use_gemm_plugin float16 \
    --output_dir examples/llama/out
```

Compile the Model

```
$ python3 examples/llama/run.py --engine_dir=examples/llama/L40 --
max_output_len 100 --tokenizer_dir meta-llama/Llama-2-7b-chat-hf --
input_text "How do I count to nine in French?"
```

Run the Model

TensorRT-LLM Usage

Technical details for Compilation steps

build.py

- Instantiates model & load weights from pretrained model
- Define builder configuration for optimization requirements
- Build and serialize the engines

run.py

- Load the prebuilt engines
- Initialize a runtime session
- Run optimized inference!

```python
# build.py
def build([...]):
  # define TRT builder config
  builder_config = builder.create_builder_config([...])

  # instantiate the TensorRT-LLM Llama model & load weights
  trtllm_llama = trtllm.models.Llama([...])
  load_from_hf_llama(tensorrt_llm_llama, hf_llama, [...])

  # build the TRT engines
  network = builder.create_network()
  network.set_named_parameters(trtllm_llama.named_parameters())
  engine = builder.build_engine(network, builder_config)

  # serialize engine
  serialize_engine(engine, path)
```

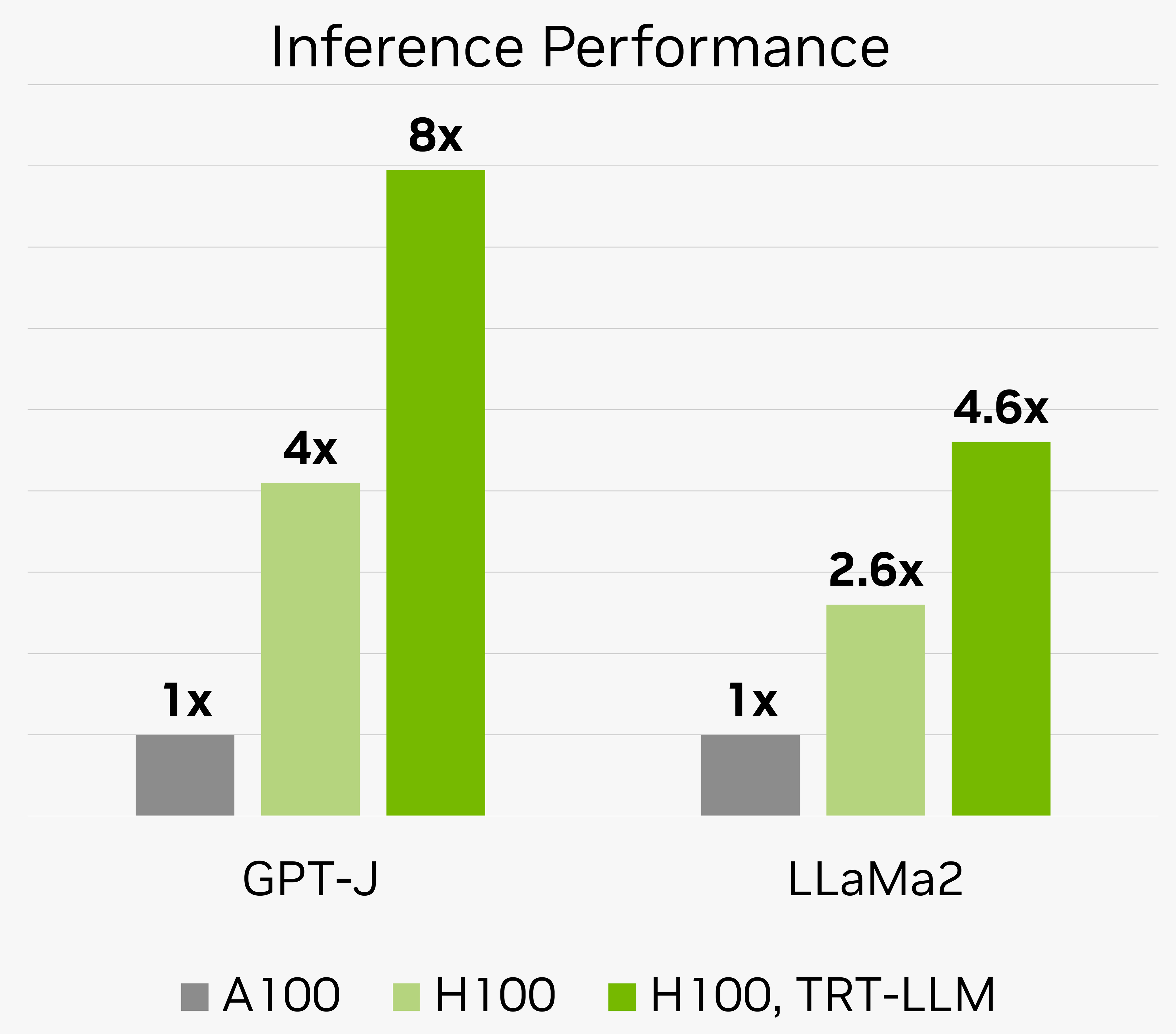
```python
# run.py
def run(path, [...]):
  # open the serialized model
  with open(path, 'rb') as f:
          engine_buffer = f.read()
  llama = trtllm.runtime.GenerationSession(engine_buffer, [...])
  # ...

  # run inference
  output = llama.decode(input, input_lengths, sampling_config)
```

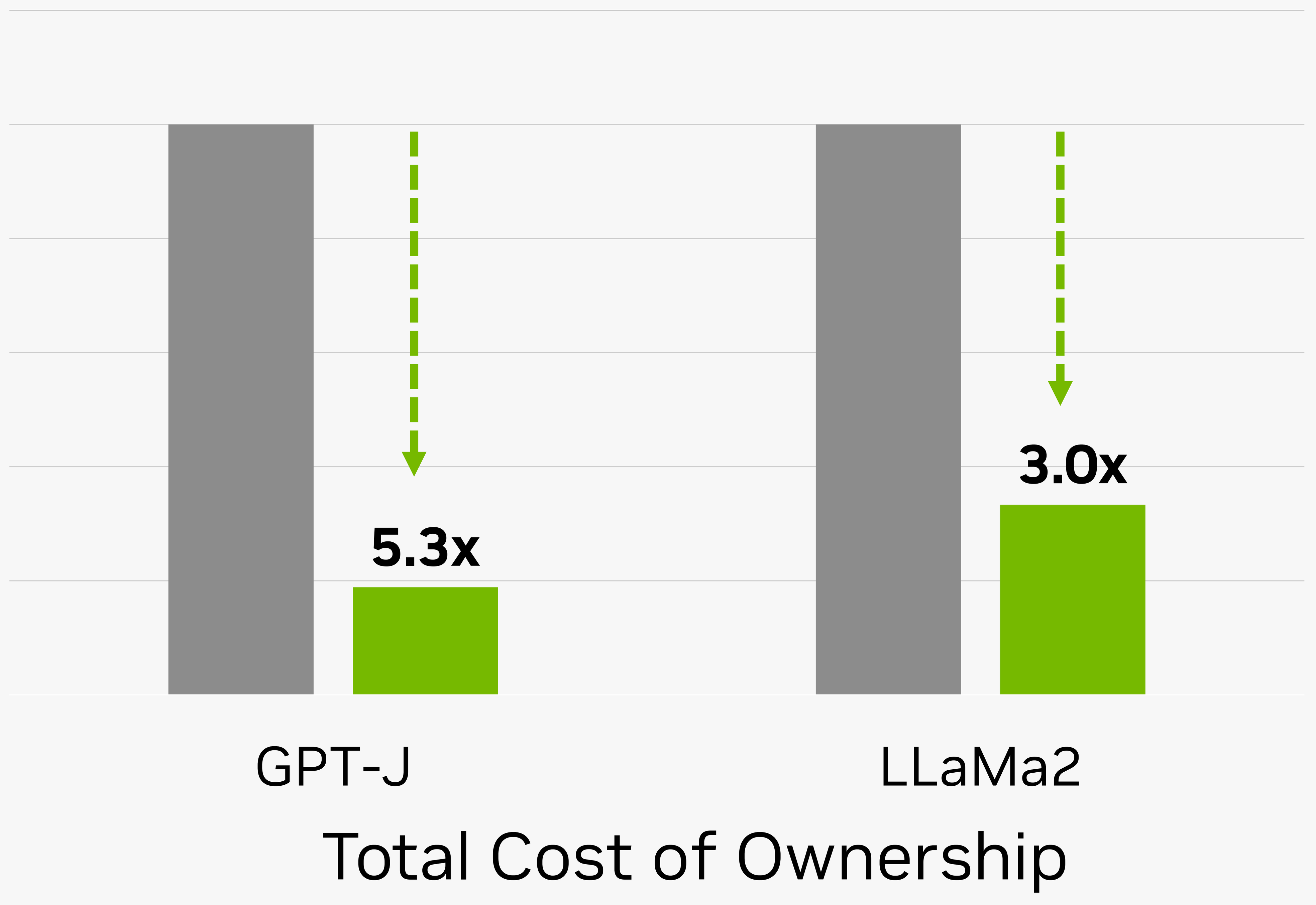
Instantiating, building, & executing inference in TensorRT-LLM

Inference Performance

TensorRT-LLM has Speed-of-Light Performance

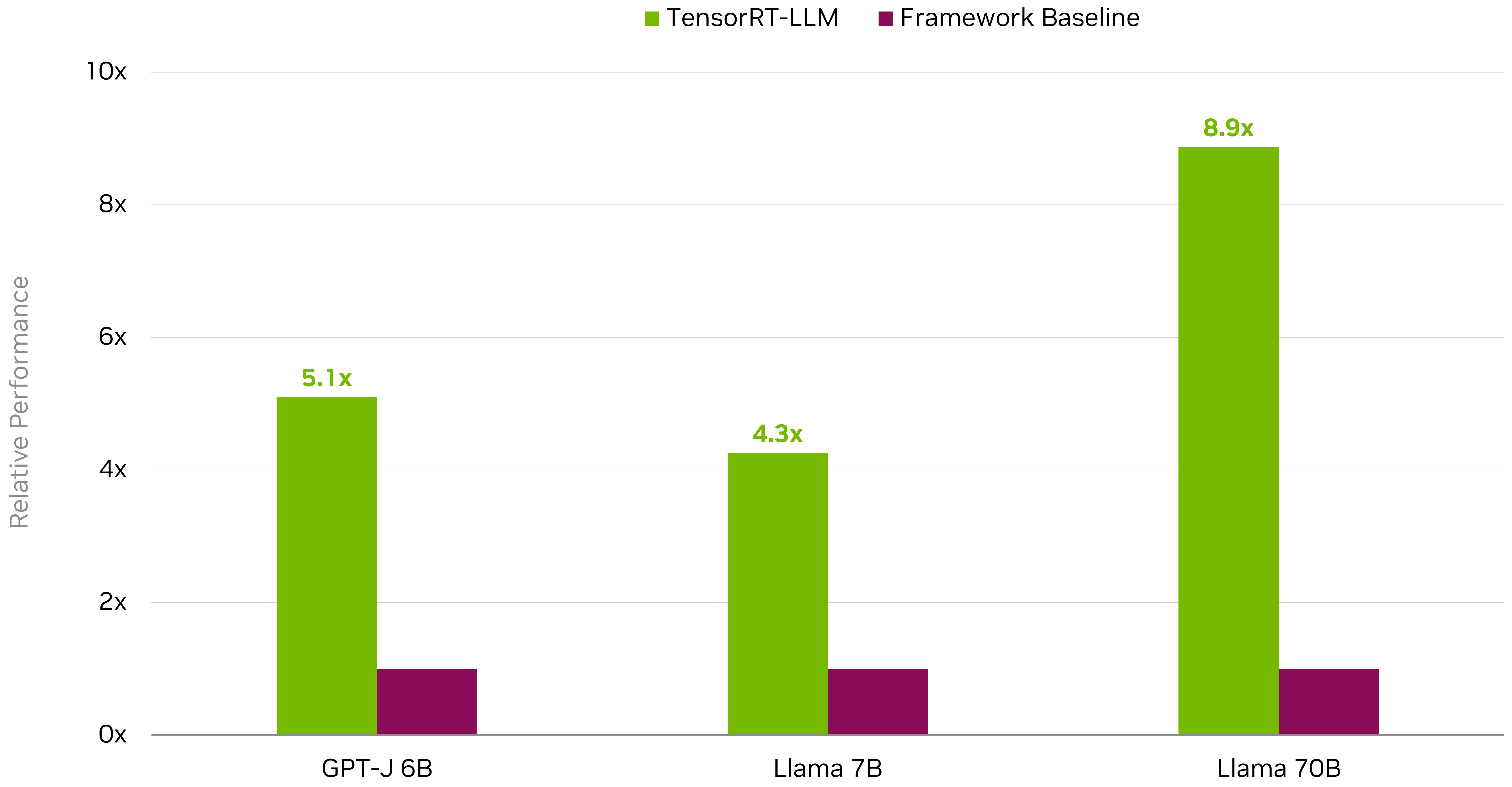
- **8x** faster Performance
  - Custom MHAs, Inflight-batching, paged & quantized KV cache, & more drive inference performance
- **5x** reduction in TCO
  - FP8 & Inflight-batching performance allows for H100 to improve TCO significantly
- **5x** reduction in Energy
  - Performance allows for reduce energy / inference allowing for more efficient use of datacenters





TensorRT-LLM Performance Improvement

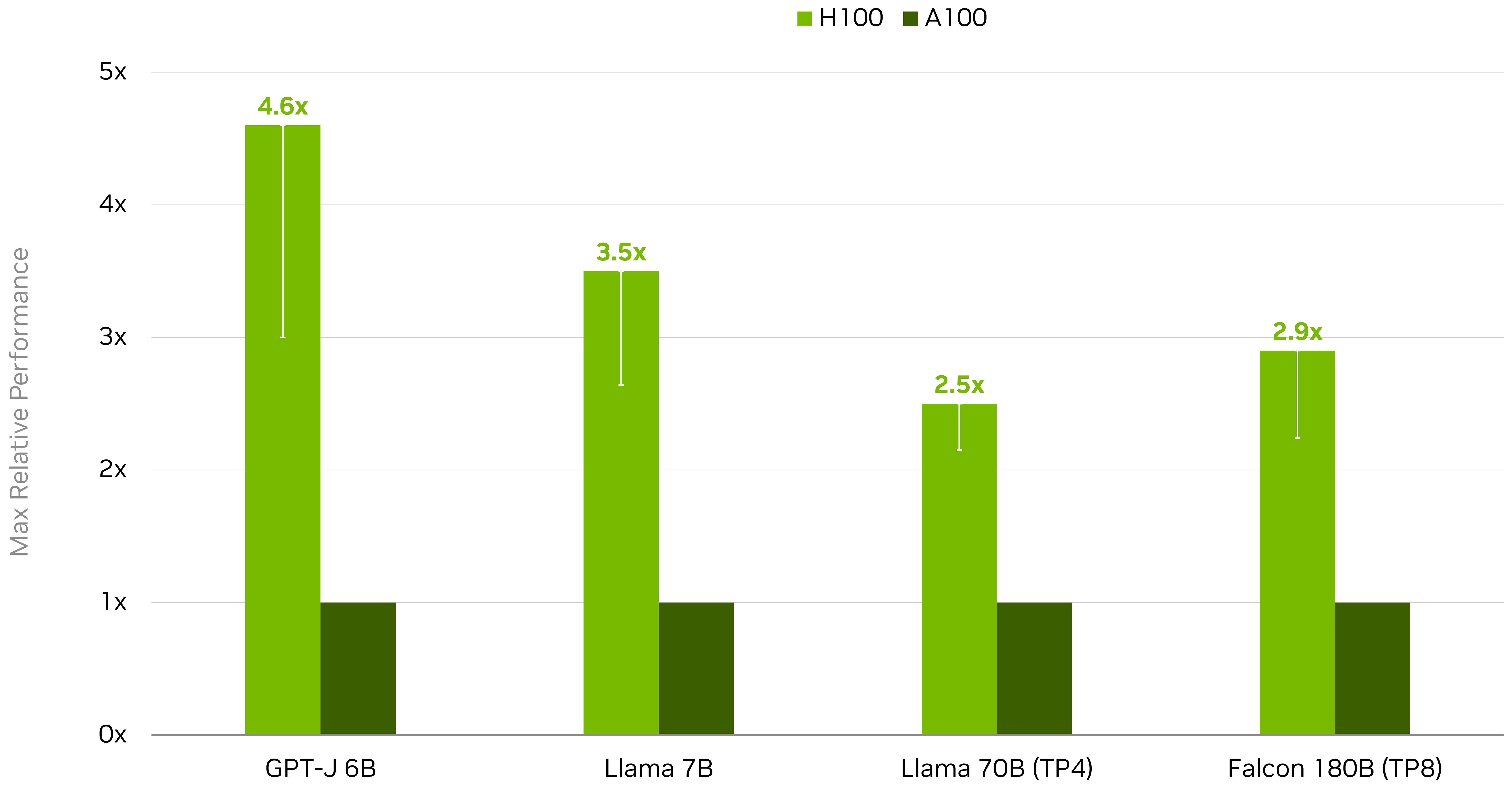
Up to 9x faster than baseline LLM implementations in DL frameworks



TensorRT-LLM v0.5.0 internal build. HF Accelerate. Tokens/s/GPU relative improvement
DGX H100. TensorRT-LLM FP8, HF Accelerate FP16.
Max batchsize up to 64. Input & output sequence length 128:128
TensorRT-LLM Llama 70B TP2, HF Accelerate PP4

TensorRT-LLM Performance Across Architectures

H100 up to 4.6x faster than A100 on TensorRT-LLM

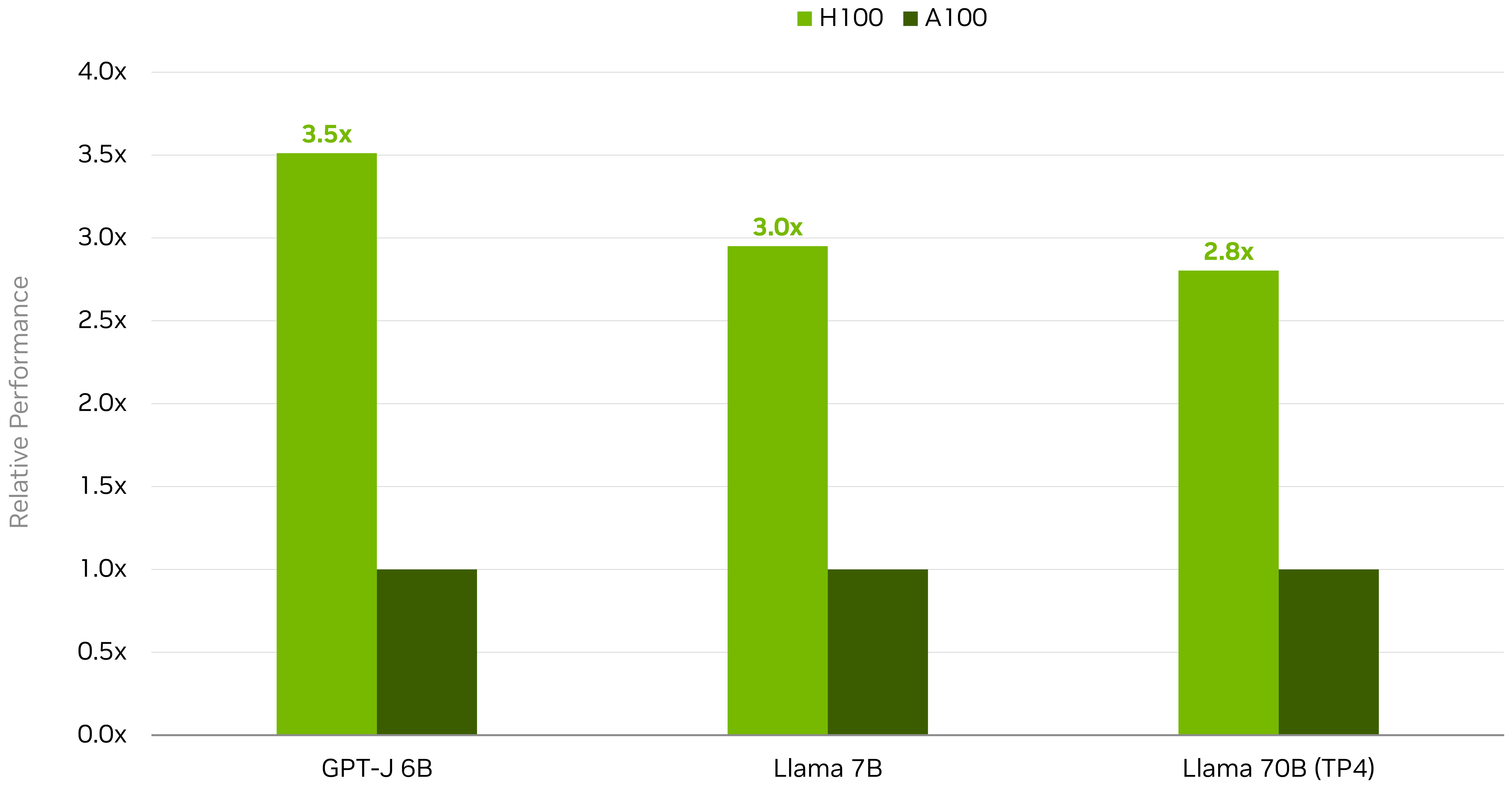


TensorRT-LLM v0.5.0 internal build. Tokens/s/gpu relative improvement
DGX H100 FP8 vs DGX A100 FP16.
Max batchsize up to 64. Input & output sequence lengths of {1,128, 2048, 4096}.
TPN = Tensor Parallel across N devices

NVIDIA

TensorRT-LLM Performance

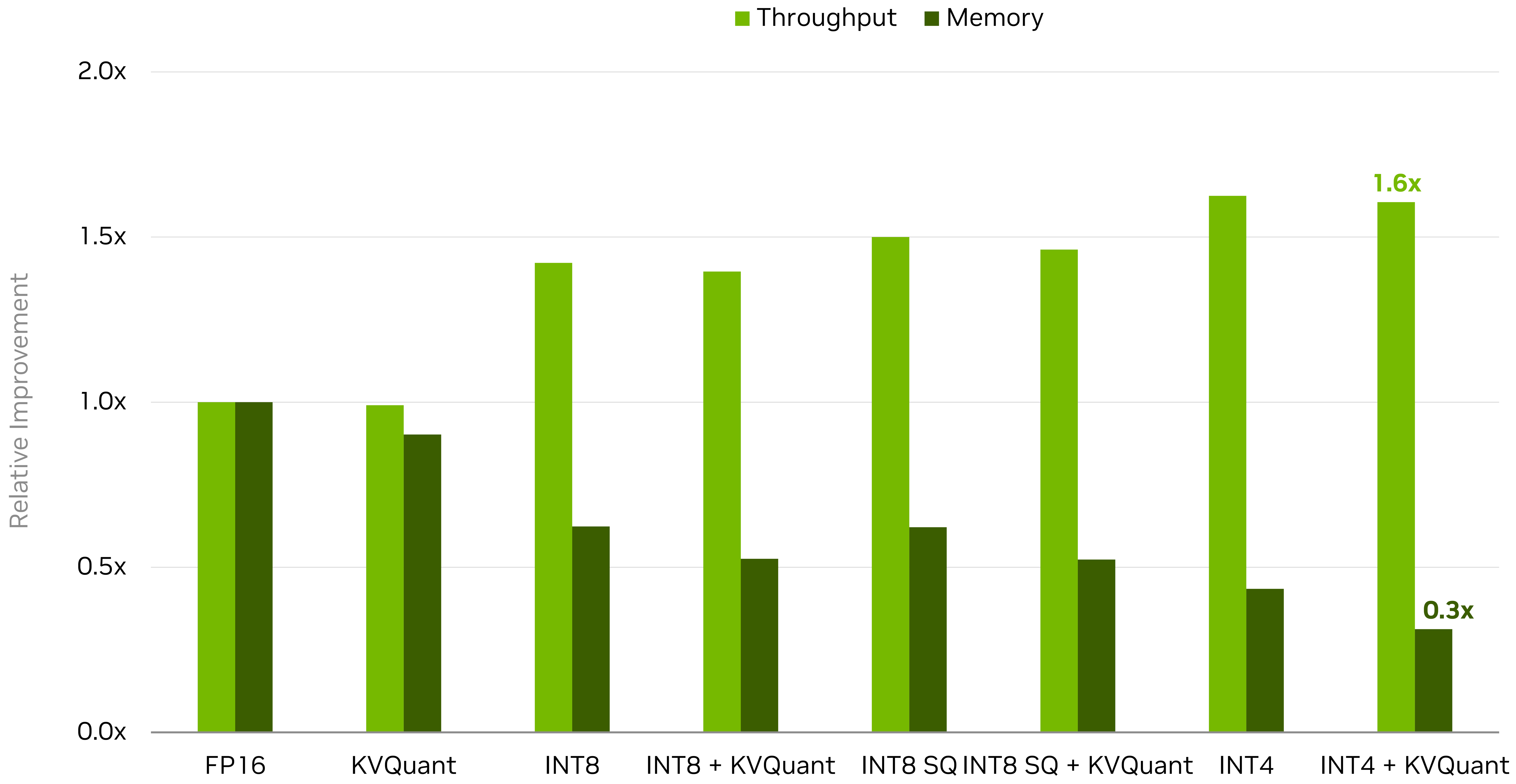
End-to-End Performance Using Inflight Batching & Triton



TensorRT-LLM v0.5.0 internal build. Triton with TensorRT-LLM inflight batching backend
DGX H100 FP8 & DGX A100 FP16
CNN Daily Mail dataset. Varying max concurrency. SOL serving scenario
TPN = Tensor Parallel across N devices.

TensorRT-LLM Performance

Advance Techniques can further improve TensorRT-LLM performance & memory consumption



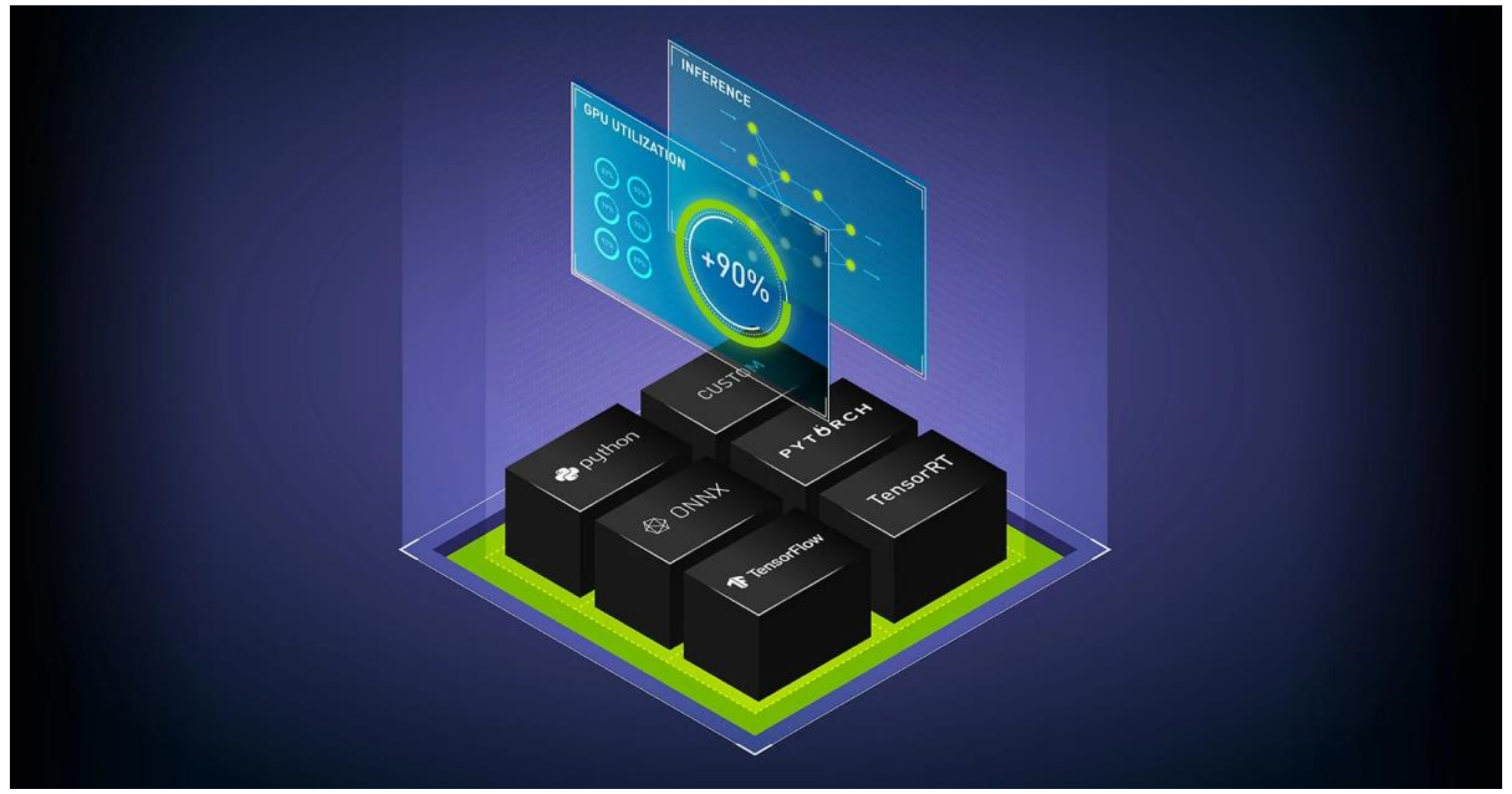
TensorRT-LLM v0.2.0 internal build. MPT-7B
1xA100-40GB. Averaged across BS [1, 512], seqlen [1, 512]

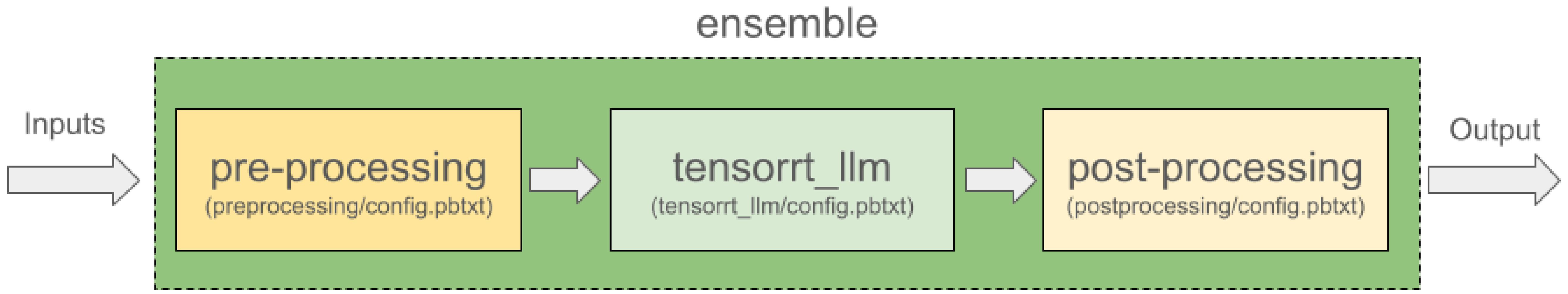
NVIDIA

TensorRT-LLM Backend

Deployment using Triton Server

- **TensorRT-LLM Backend**
  - TensorRT-LLM backend uses the Triton Inference server to deploy and server the models.
- **Feature Rich**
  - Inflight-batching allows for increased performance and throughput in Model inference.
- **Multi-client support**
  - HTTP, gRPC and Python/C++ Client Library can be used to fetch results from the inference server.





NVIDIA®