TensorRT-LLM Workshop

Dora Csillag, Sr. Solution Architect

Asma Farjallah, Sr. Solution Architect

Julien Demouth, Sr. Distinguished Eng. / TensorRT-LLM Eng. Lead

Production Language Apps

Deploying massive models for real-time applications

- Increasing need for deep learning in language applications
  - Chat, translation, summarization, search, generation, etc.
- High accuracy models are important for correct results
  - Model accuracy directly correlates to helpfulness for users
- “Online” deployment require end-user acceptable latencies
  - Ensure a great experience with applications
- Multi-functional, accurate models are *large* making them slow during inference & expensive to deploy

Making **cost effective** deployments **challenging**

Large Language Model Ecosystem

Rapid evolution makes optimization challenging

- Increasing rate of new foundational LLMs being released
  - Llama, Falcon, Starcoder, ChatGLM, MPT, & more
- New operators & customization techniques makes optimization a moving target
- Latest models continue to be very large for best accuracy
  - 70-200 Billion parameter or more

Need a **performant**, **robust**, & **extensible solution** for **cost-effective**, **real-time** LLM deployments

LLaMa | GPT | Falcon | Starcoder | ChatGLM | MPT | ...

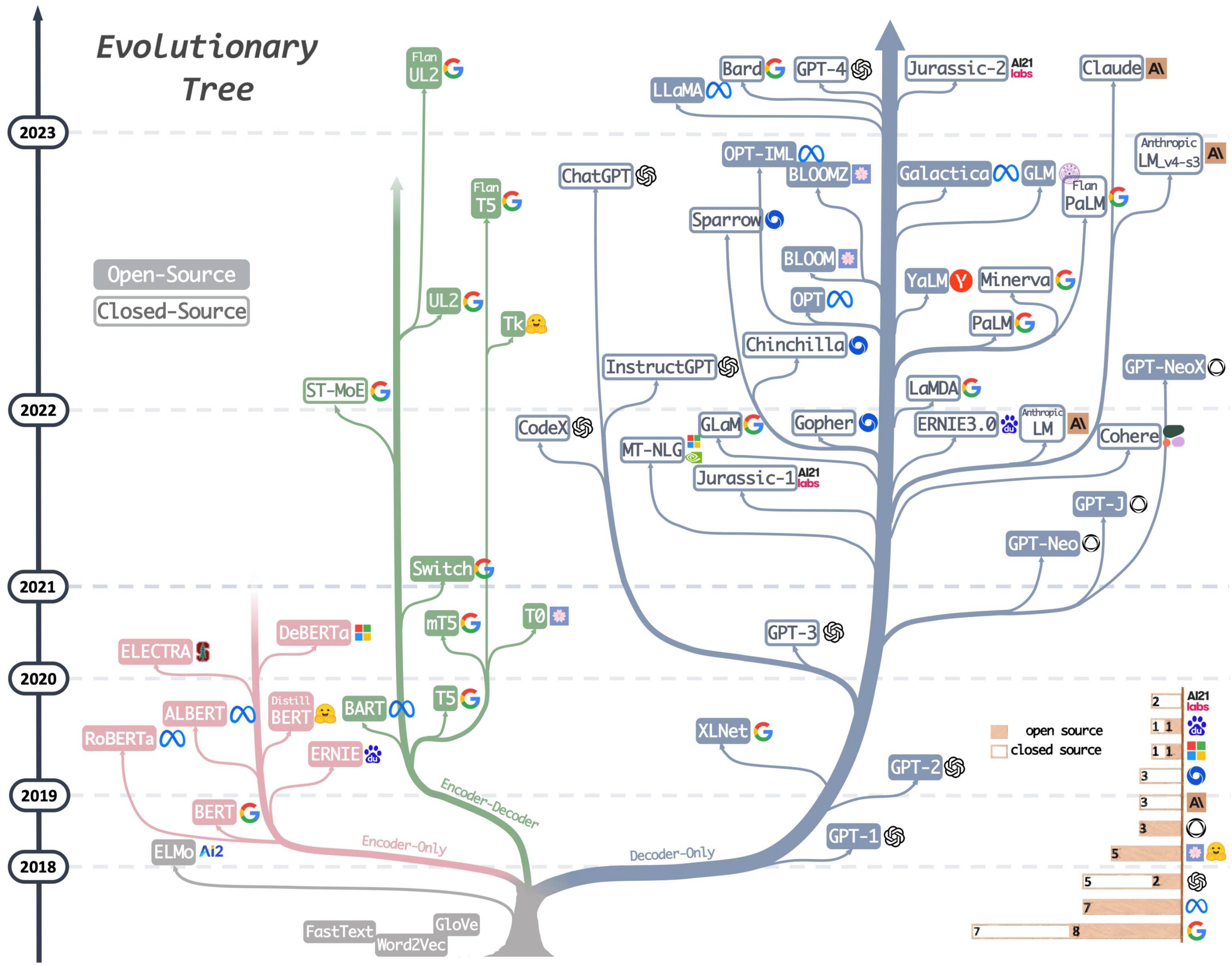


Image from Mooler0410/LLMsPracticalGuide

Yang, J., Jin, H., Tang, R., Han, X., Feng, Q., Jiang, H., ... Hu, X. (2023). Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond. arXiv [Cs.CL]. Retrieved from http://arxiv.org/abs/2304.13712

TensorRT-LLM Optimizing LLM Inference

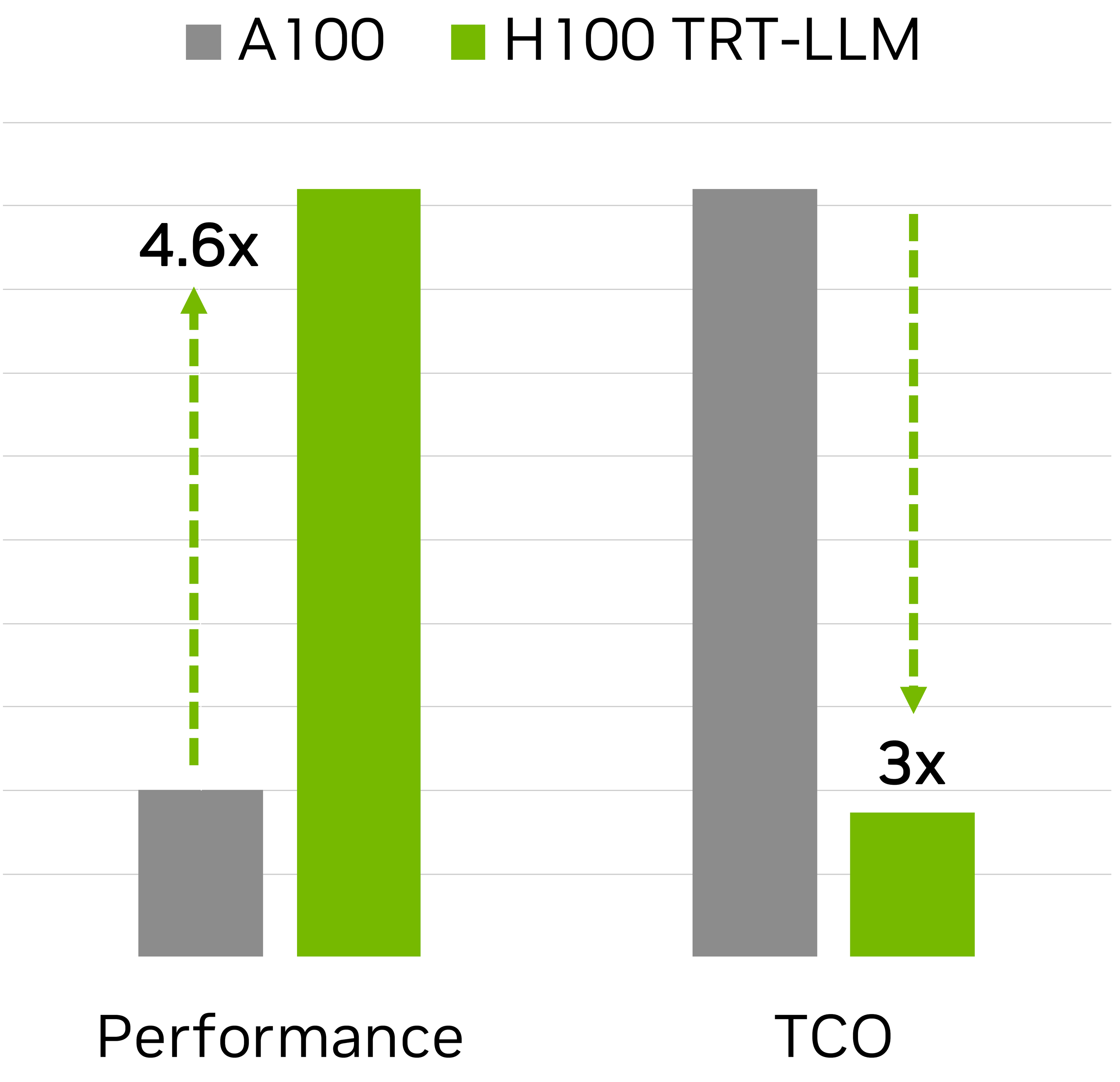
SoTA Performance for Large Language Models for Production Deployments

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 TensorRT with a simple Python API for defining, optimizing, & executing LLMs for inference in production.

SoTA Performance

Leverage TensorRT compilation & hand-tuned kernels developed by GPU experts



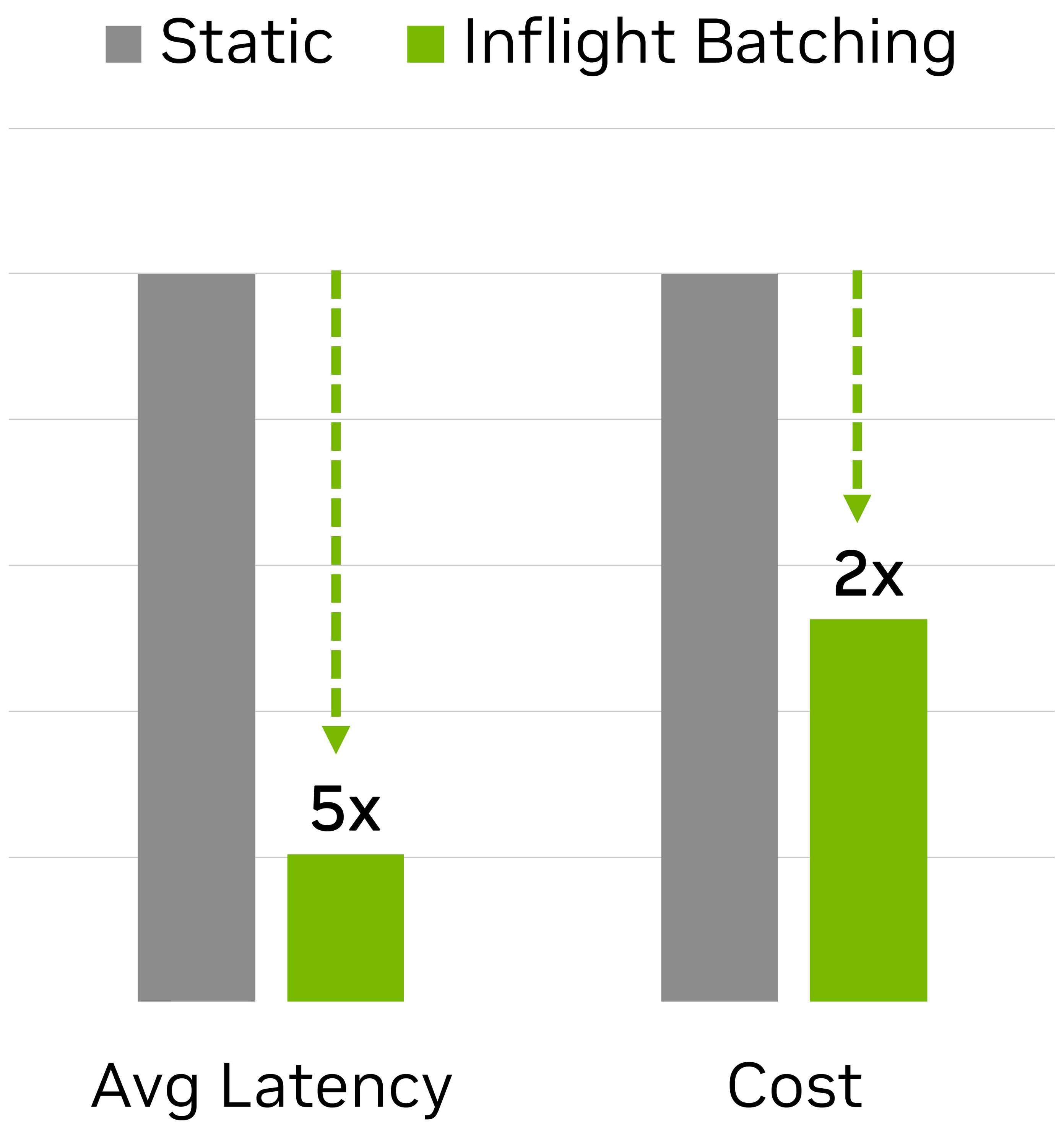
Ease Extension

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



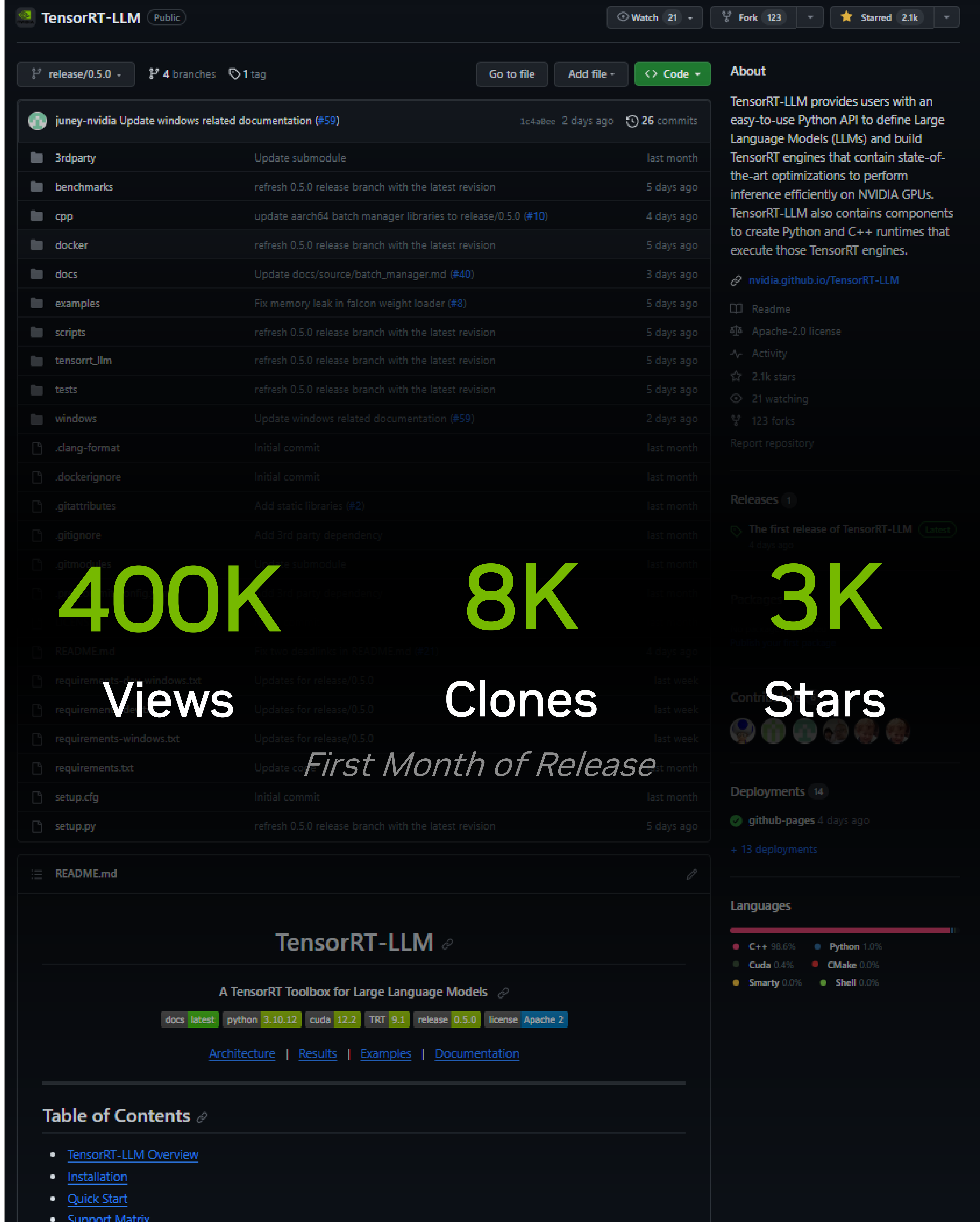
Numbers are preliminary based on internal evaluation on Llama 7B on H100

TensorRT-LLM Available Now!

Get it on Github, NGC, & coming soon to NeMo

TensorRT-LLM is live at NVIDIA/TensorRT-LLM

- All source provided under Apache 2.0
- Model examples, quantization toolkits & more
- In Triton 23.10+ NGC Container
- Coming soon to NeMo Framework Inference



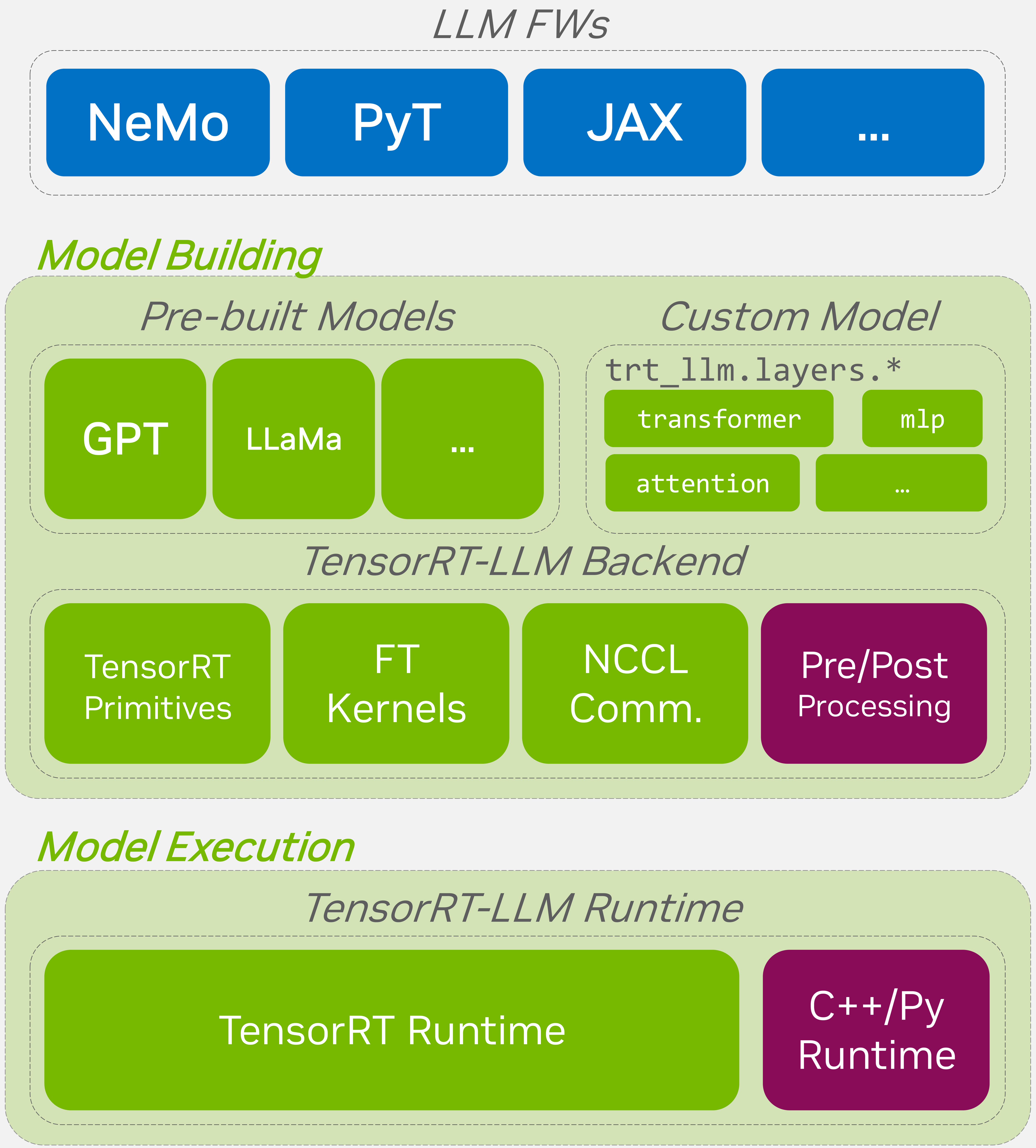
Workflows and Examples

TensorRT-LLM Usage

Use Pre-built models, or optimize new ones!

TensorRT-LLM Inference

- Framework/TE-like building blocks for transformers
  - Ex. fMHA, layerNorm, activations, etc.
  - Built on top of [TensorRT Python API]
- Build arbitrary LLM or deploy pre-built implementations
  - Ex. GPT, LLaMa, BERT, etc.
- MGMN inference
  - Leverages NCCL plugins for multi-device communication
  - Pre-segmented graphs in pre-built models
  - User can manually segment for custom models
  - *Future may allow automatic segmentation across gpus*
- Combines TensorRT layers, NCCL plugins, perf plugins, & pre/post processing ops into a single object
  - Include tokenization & Sampling (ex. Beam search)



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

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!

The **`examples/*`** are representative implementations of using TensorRT-LLM supported models.

- Guides for how to use TensorRT-LLM in an application

```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

Demo – Single GPU

TensorRT-LLM Usage

Model Building & Defining Operators

Easily modify models

- Modify the model similar to in a DL FW
- Add operators to forward call as desired
- Ops can be used with any model

Improved op coverage & definition

- Define ops in Python with TensorRT Python primitives
- *or* Map ops to arbitrary kernels with plugins
- Compose operators in Python

```python
# llama.py
class LLaMaModel(Module)
  def __init__(…)
    self.layers = ModuleList([…,trt_llm.linear()])

  def forward (…)
    hidden_states = self.embedding(…)

    for layer in self.layers:
      hidden_states = layer(hidden_states)

    # add a new layer to a model with modular building blocks.
    hidden_states = self.linear(hidden_states)
    return hidden_states
```

Modify models simply with modular Python-layers

```python
# functional.py
def sigmoid(input: Tensor) → Tensor:
  layer = default_trtnet().add_activation(input.trt_tensor,
                                          trt.ActivationType.SIGMOID)
  return _create_tensor(layer.get_output(0), layer)


def silu(input: Tensor) → Tensor:
    return input * sigmoid(input)


def swiglu(input: Tensor) -> Tensor:
    x, gate = chunk(input, 2, dim=-1)
    return silu(gate) * x
```

Creating new ops from TRT is only a few lines of Python

Implementing New Operators

RMSNorm via TensorRT Python APIs

Implement new operators quickly via TensorRT

- Quickly implement, entirely in python, unblock deployments
- Compiled entirely in TensorRT
- Fused into a single kernel
- Same or similar performance to custom CUDA kernels

TensorRT may not always fuse operators to a single kernel, impacting performance.

```
def rms_norm(input: Tensor,
             normalized_shape: Union[int, Tuple[int]],
             weight: Optional[Tensor] = None,
             eps: float = 1e-05) → Tensor:
  dim = tuple([-i - 1 for i in range(len(noramlized_shape))])

  with precision("float32")
    varx = pow(input, 2.0)
    varx = varx.mean(dim, keepdim=True)
    denom = varx + eps
    denom = denom.sqrt()
    y = input / denom

  if weight is not None:
    y = y * weight

  return y
```

Implementing RMSNorm in TensorRT-LLM

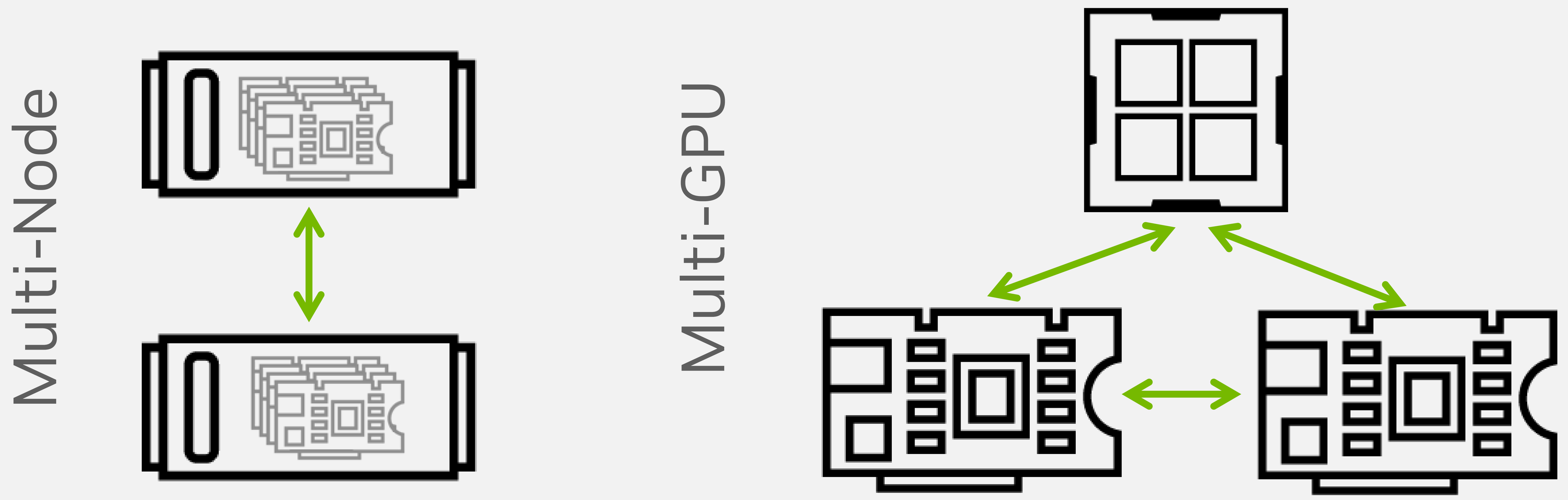
NVIDIA

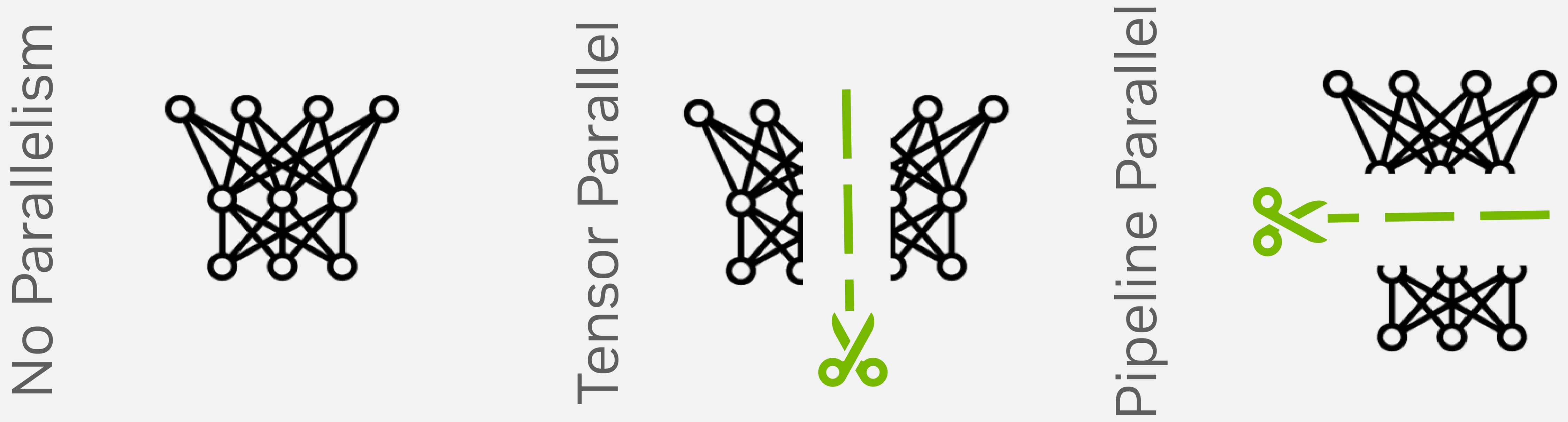
Performance

Multi-GPU Multi-Node

Sharding Models across GPUs

- Supports Tensor & Pipeline parallelism
- Allows for running very large models (tested up to 530B)
- Supports mutli-GPU (single node) & mutli-node
- TensorRT-LLM handles communication between GPUs
- Examples are parametrized for sharding across GPUs





Demo – Multi-GPU

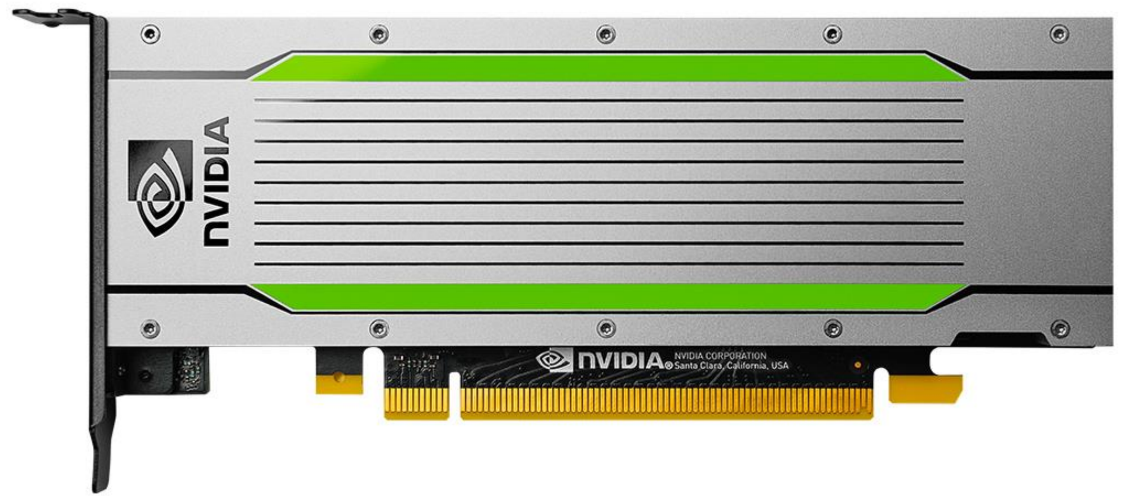
TensorRT-LLM GPU Support

Hopper to Volta

Hopper
H100

Ada
L4, L40, L40S

Ampere
A100, A30, A10



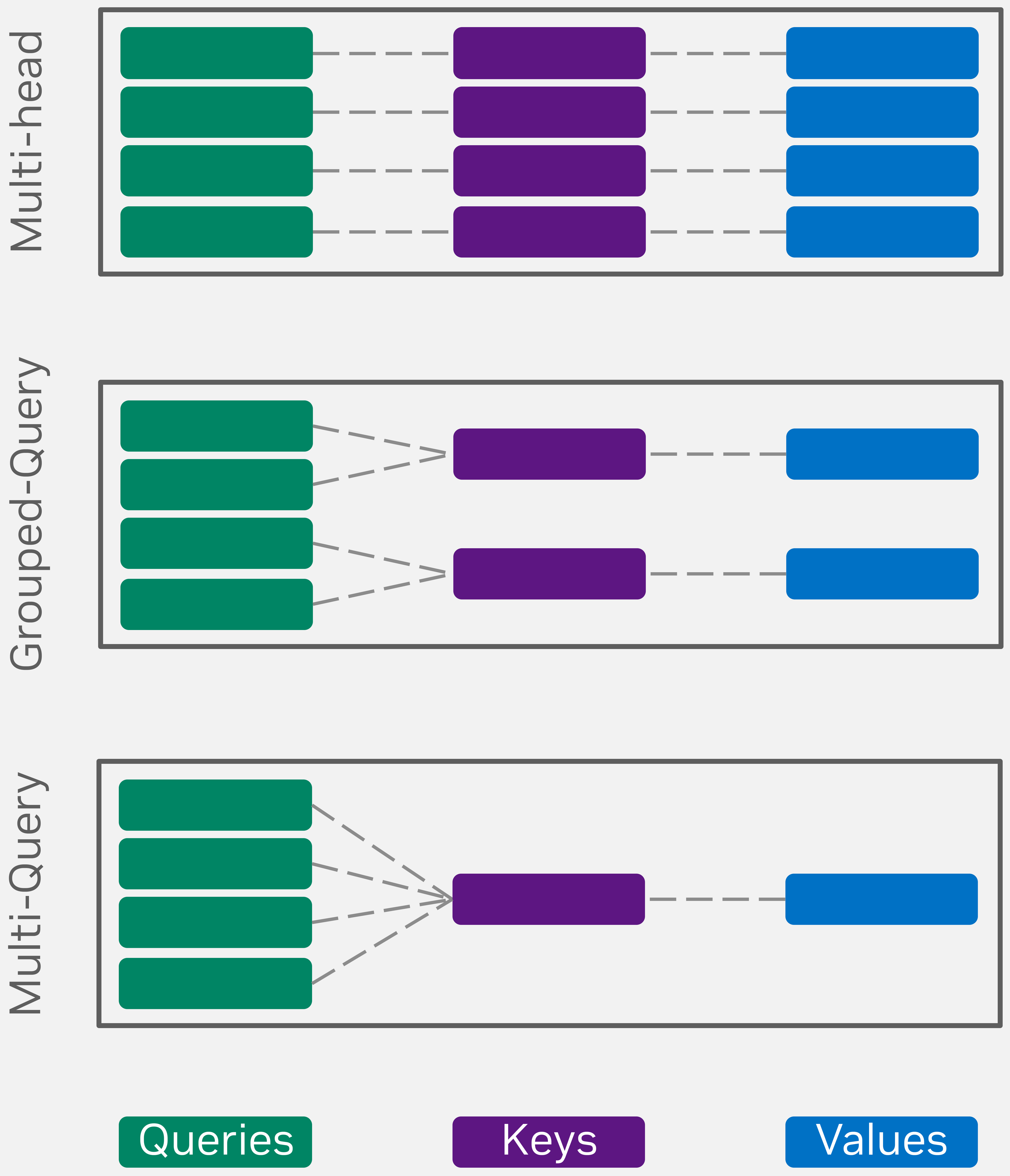
Turing
T4 (Experimental)

Volta
V100 (Experimental)

Optimized Attention

Custom Implementations for Attention

- Custom optimized CUDA kernels for Attention
  - Similar to FlashAttentionV2
- Optimized for A100 & H100
- Kernels for Encoder & Decoder, as well as context & prefill
- Supports MHA, MQA, GQA



KV Cache Optimizations

Paged & Quantized KV Cache

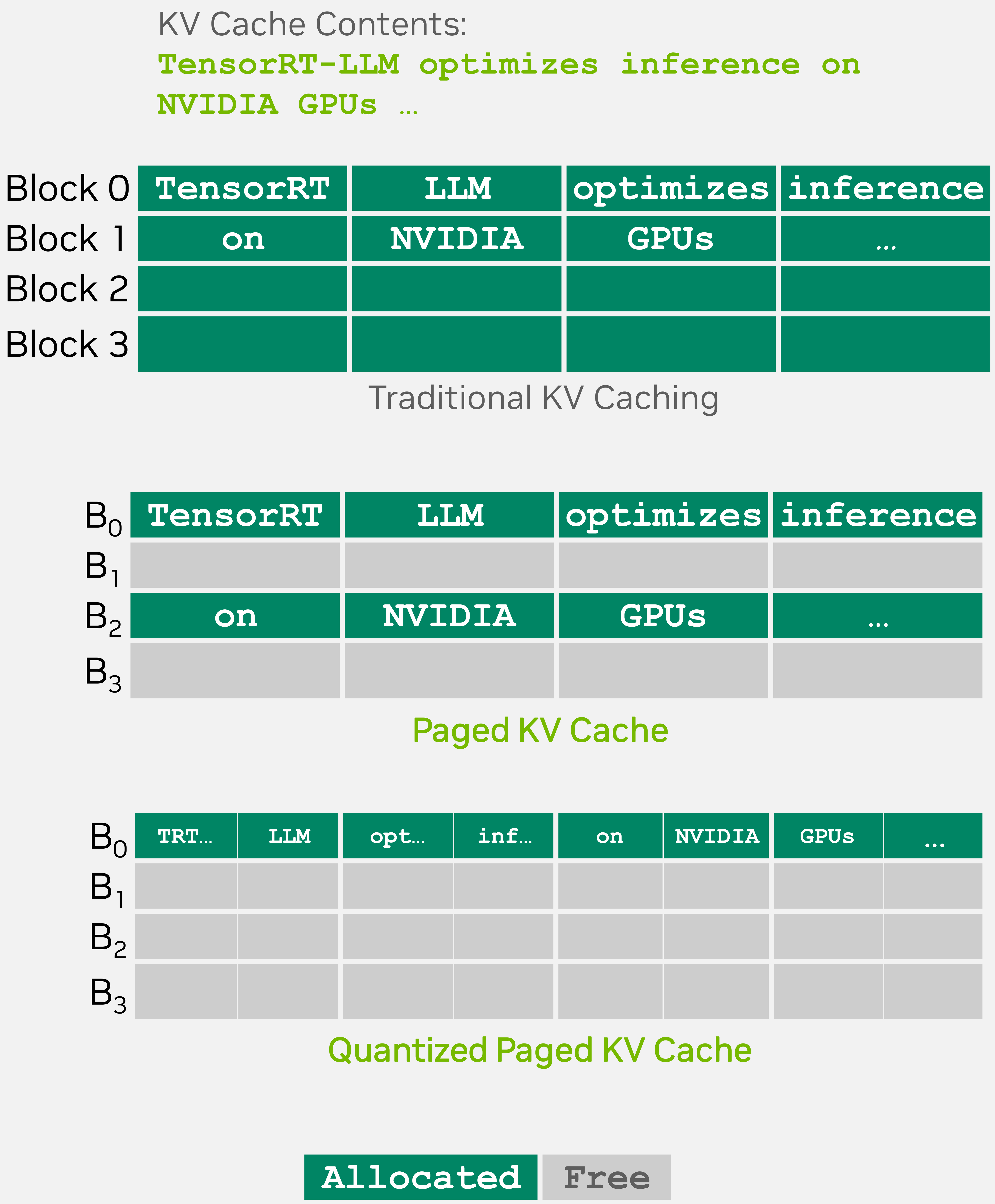
Paged KV Cache improves memory consumption & utilization

- Stores keys & values in non-contiguous memory space
- Allows for reduced memory consumption of KV cache
- Allocates memory on demand

Quantized KV Cache improves memory consumption & perf

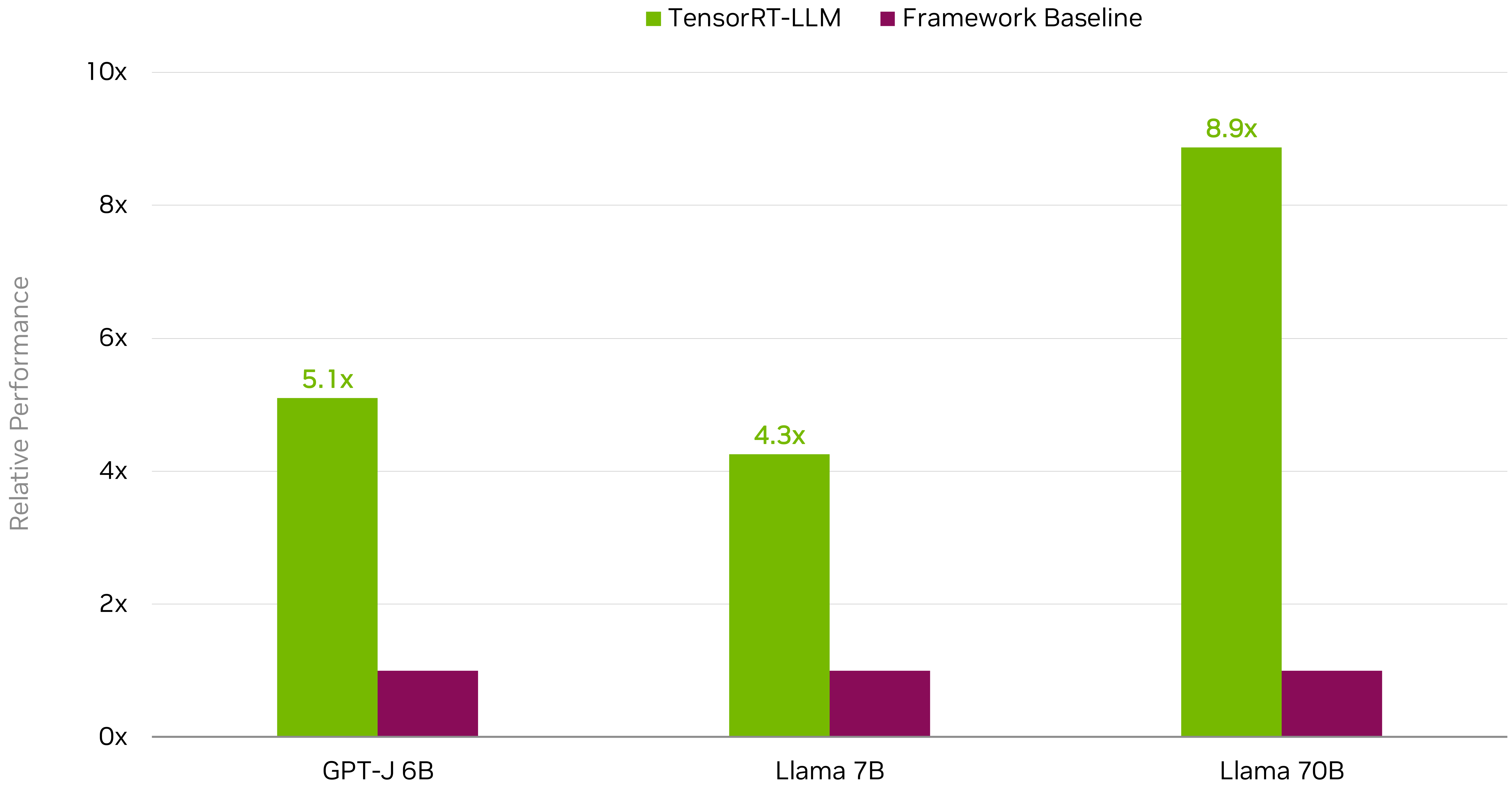
- Reduces KV Cache elements from 16b to 8b (or less!)
- Reduces memory transfer improving performance
- Supports INT8 / FP8 KV Caches

Both allow for increased peak performance



TensorRT-LLM Performance Improvement

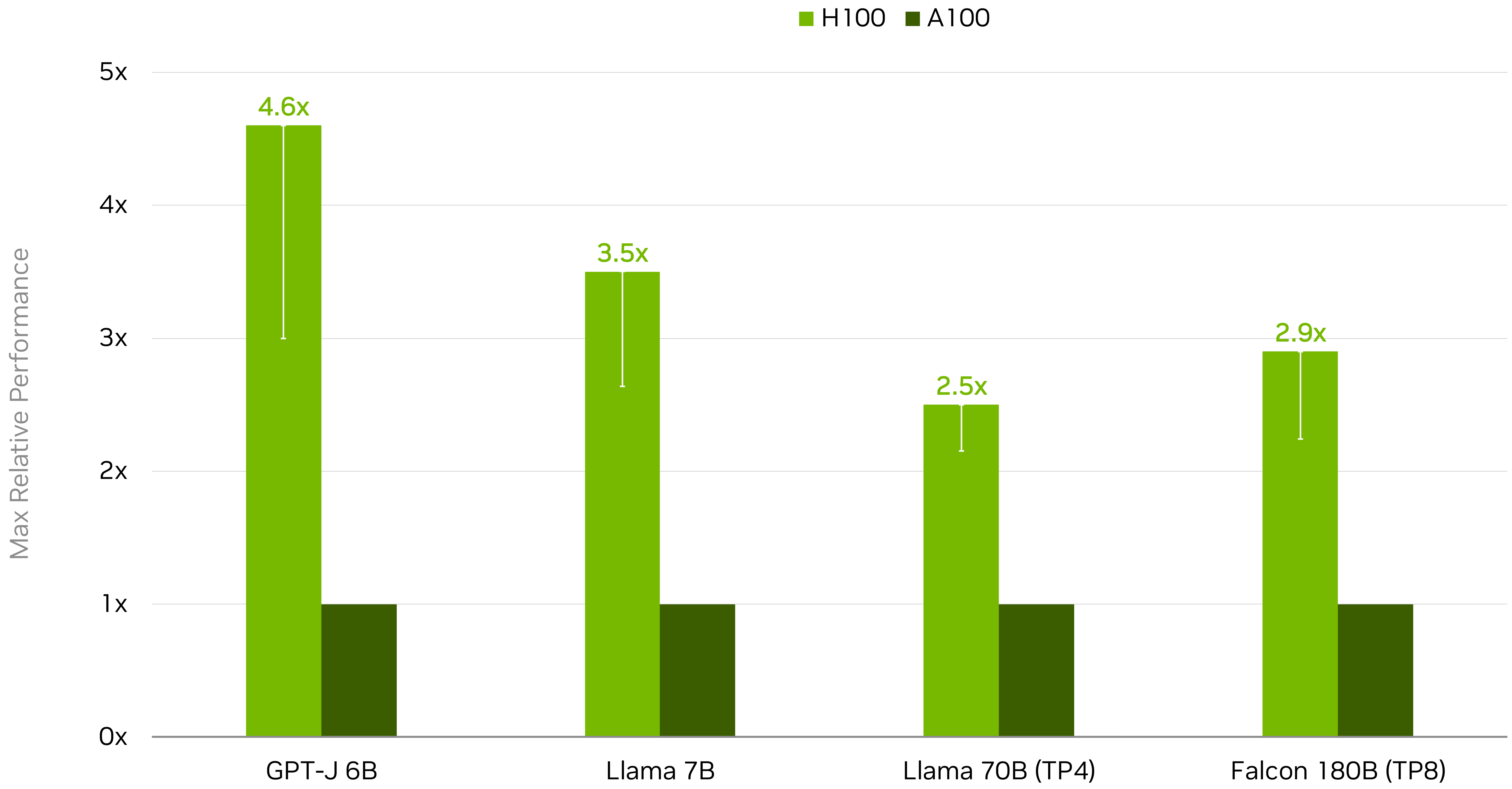
Up to 9x faster than baseline LLM implementations in DL frameworks



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

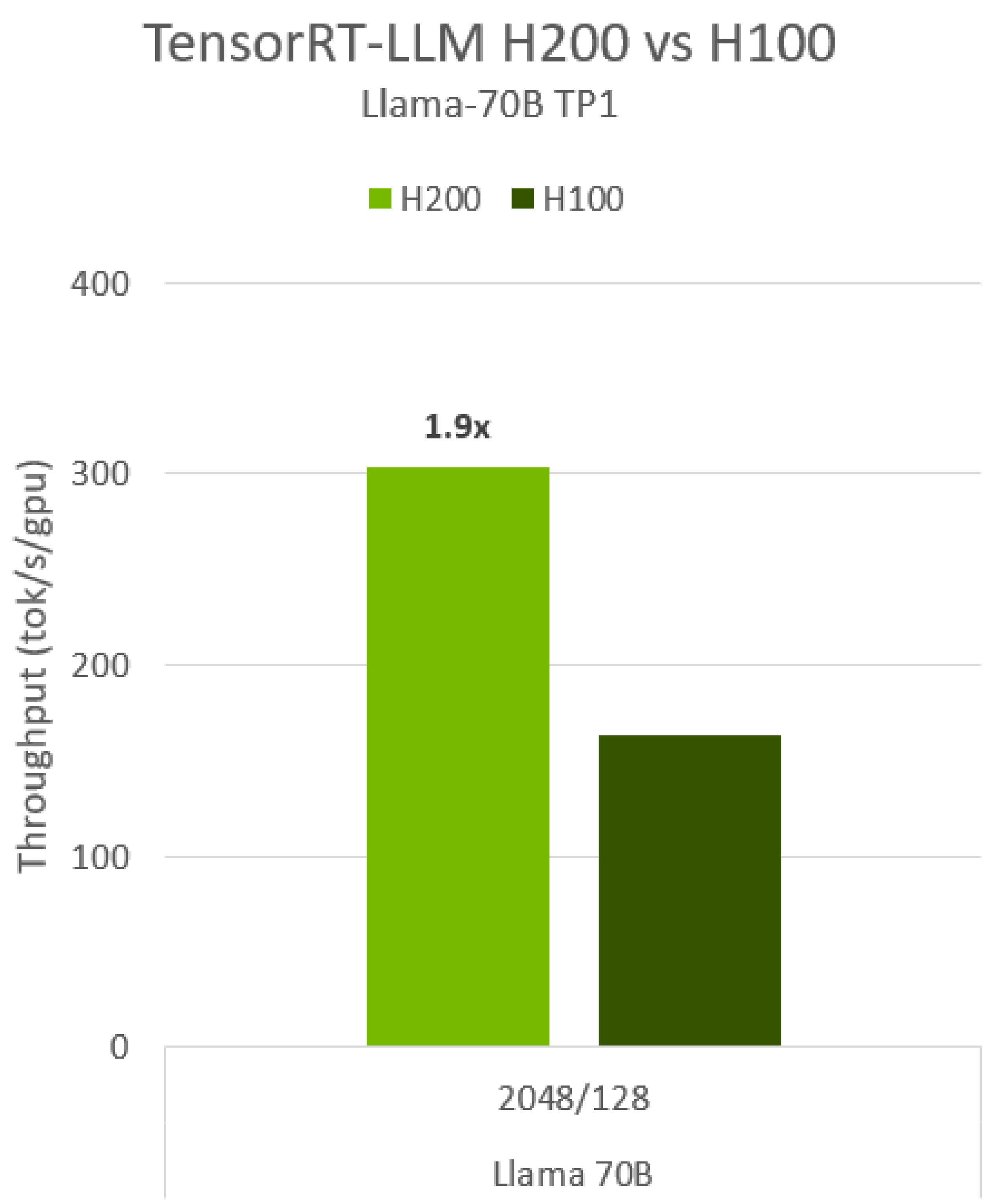


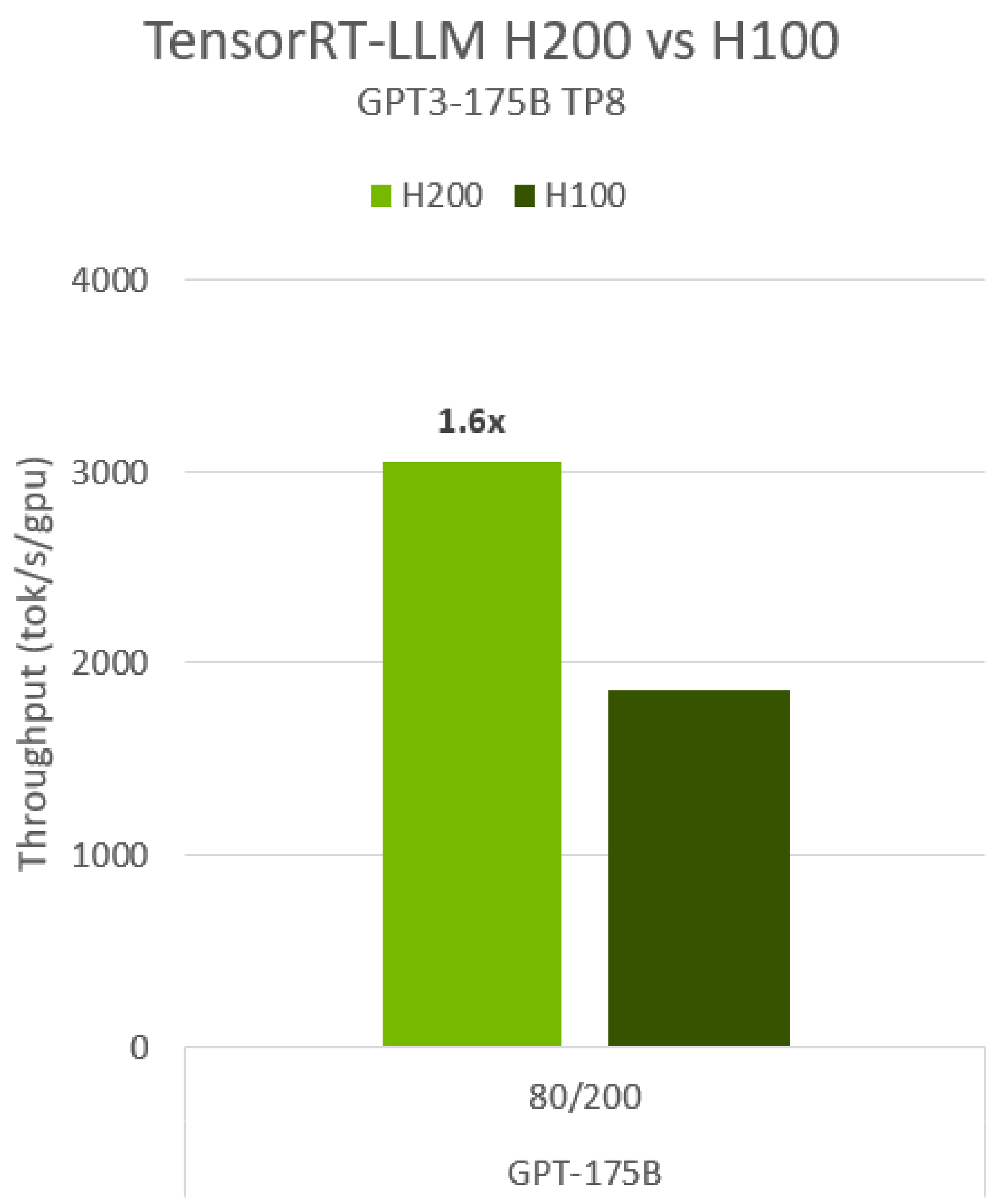
TensorRT-LLM v0.5.0 internal build. Tokens/s/gpu relative improvement
DGX H100 FP8 vs DGX A100 FP16.
Max batchsize up to 64. Input & output sequence lengths of {1,128, 2048, 4096}.
TPN = Tensor Parallel across N devices

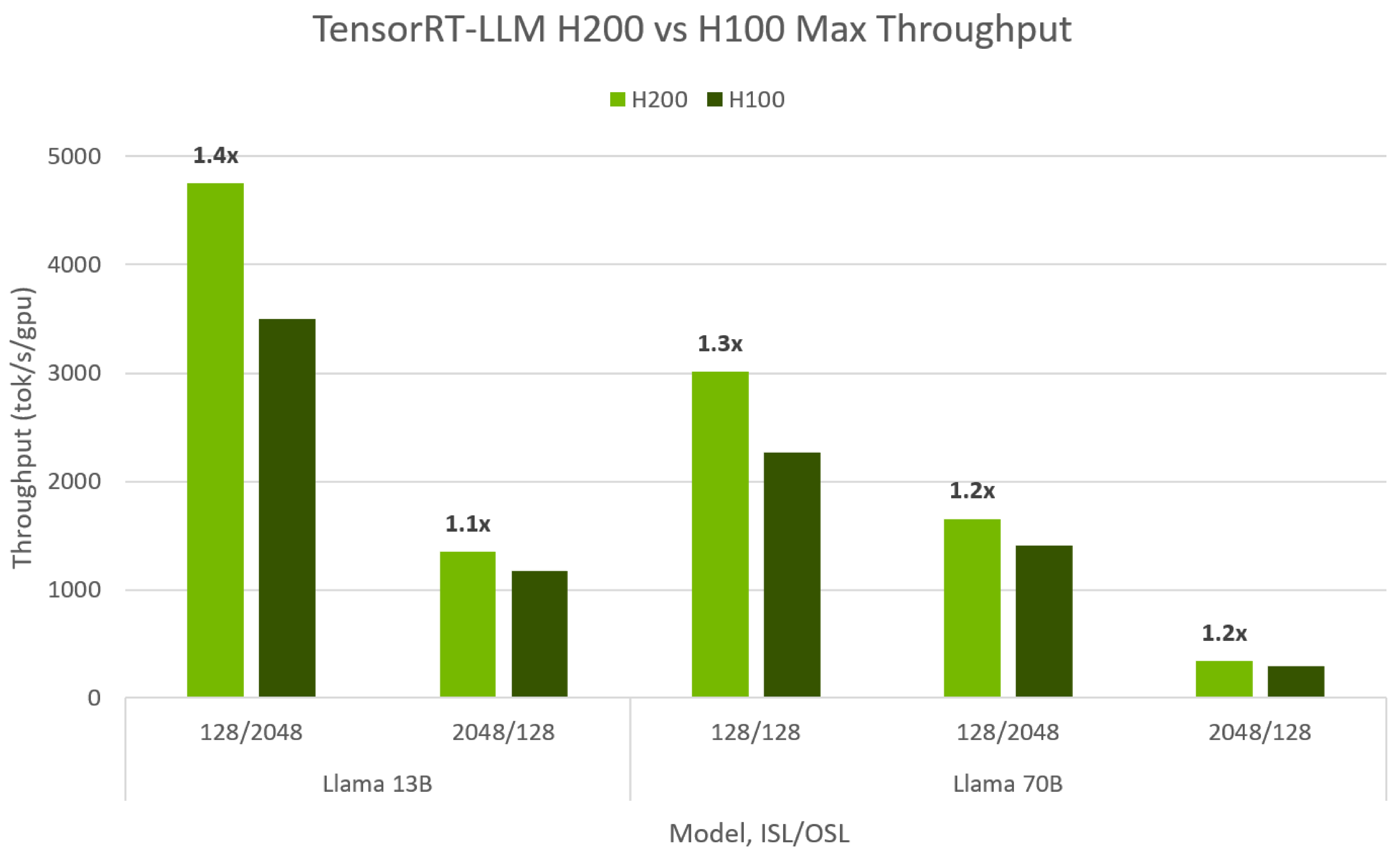
NVIDIA

TensorRT-LLM Performance Across Architectures

H200 already supported in TensorRT-LLM





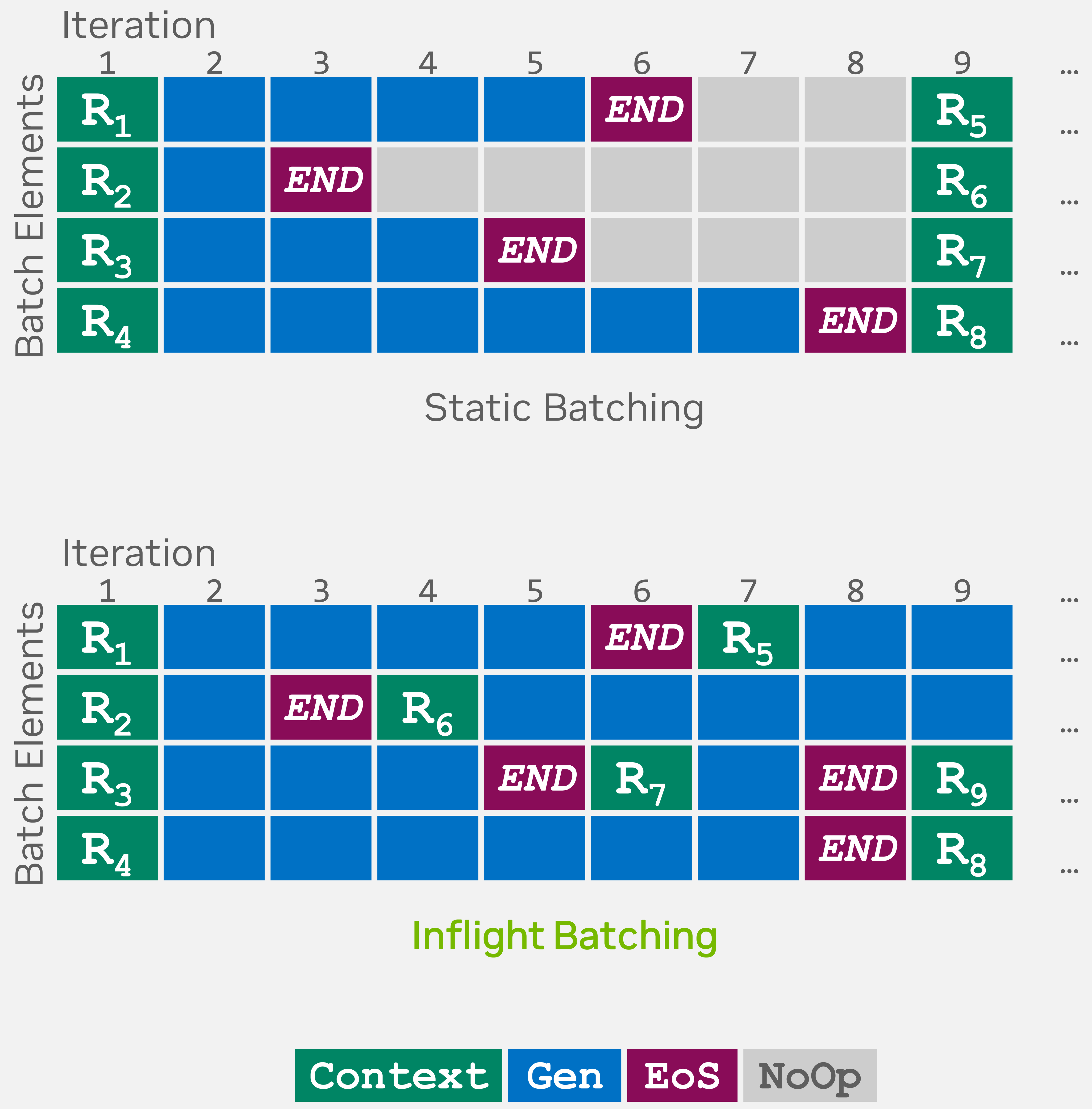


Inflight Batching

Maximing GPU Utilization during LLM Serving

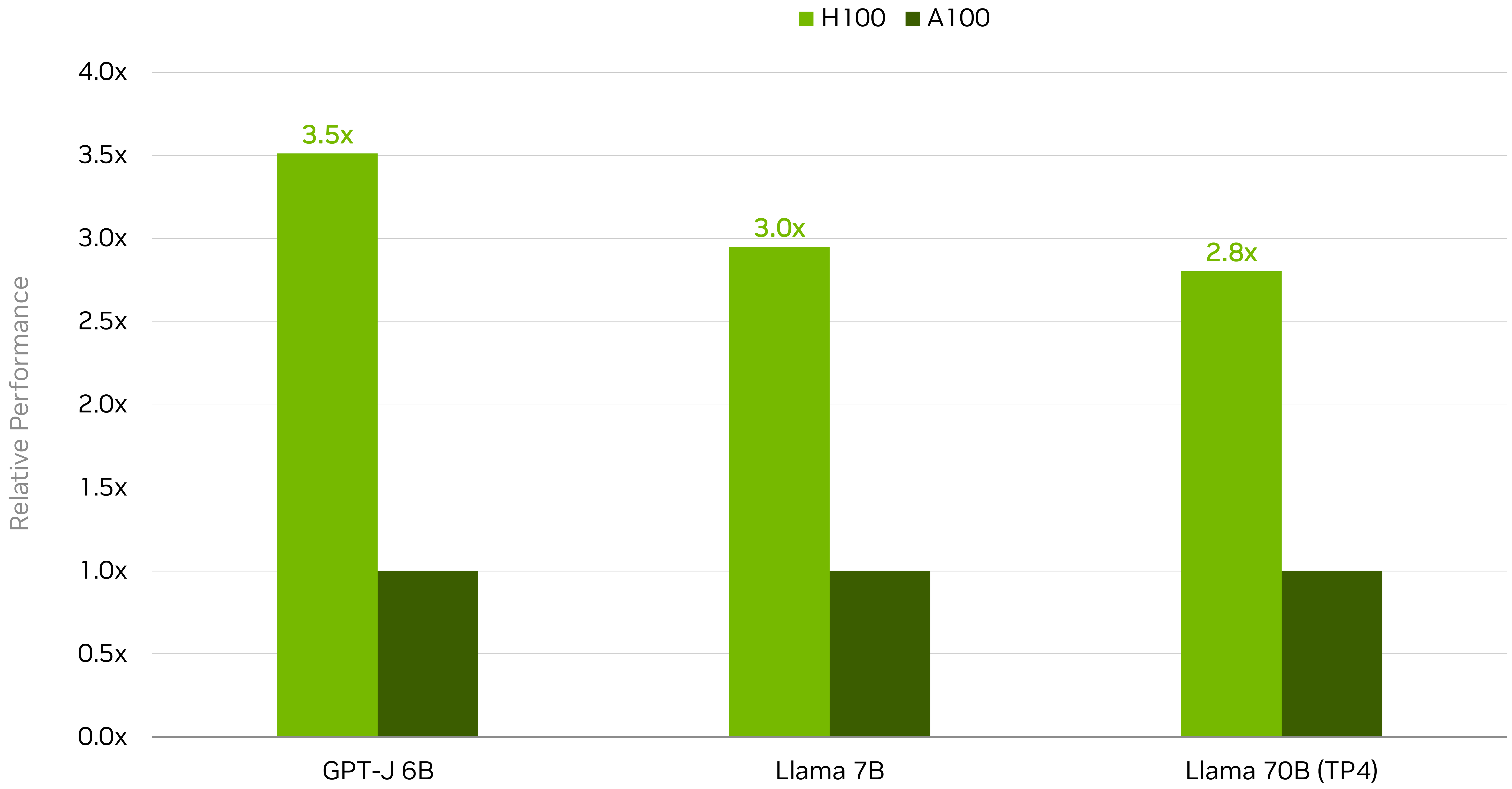
TensorRT-LLM provides custom Inflight Batching to optimize GPU utilization during LLM Serving

- Replaces completed requests in the batch
    - Evicts requests after EoS & inserts a new request
- Improves throughput, time to first token, & GPU utilizaiton
- Integrated directly into the TensorRT-LLM Triton backend
- Accessible though the TensorRT-LLM Batch Manager



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.

Demo – In-flight batching

Quantization

Supported Precisions & Models

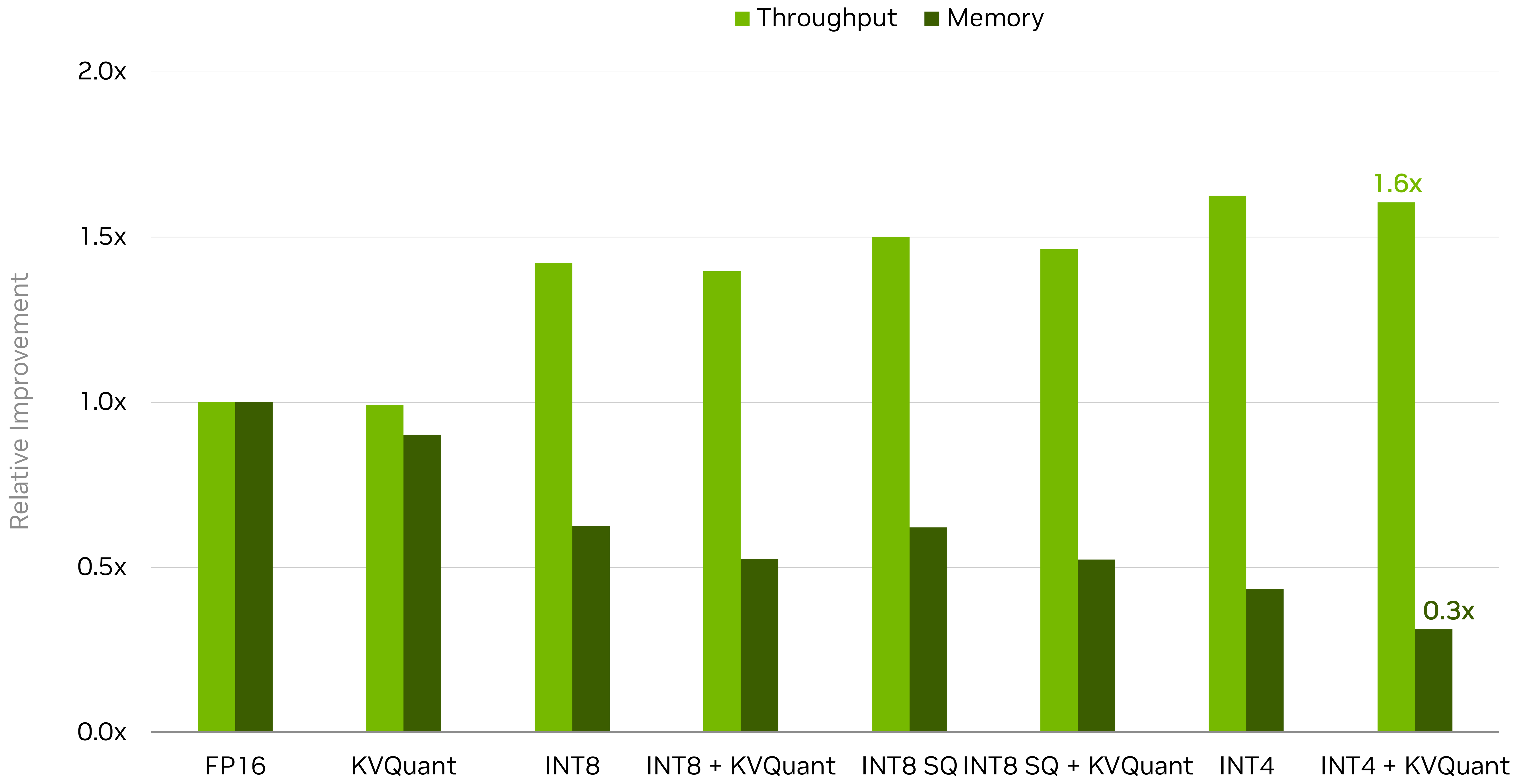
- Utilizes Hopper FP8 “Transfomer Engine”
- Support many 8bit & 4bit methods
  - FP8, INT8/INT4 Weight only, INT8 Smooth Quant, AWQ, GPTQ
  - Support varies by model
- Reduced model size, memory bandwidth, & compute
  - Improves performance & allows for larger models per GPU
- Model optimization toolkit to quantize pre-trained models

	FP32	FP16	BF16	FP8	INT8	INT4
Volta (SM70)	Y	Y	N	N	Y	Y
Turing (SM75)	Y	Y	N	N	Y	Y
Ampere (SM80, SM86)	Y	Y	Y	N	Y	Y
Ada-Lovelace (SM89)	Y	Y	Y	Y	Y	Y
Hopper (SM90)	Y	Y	Y	Y	Y	Y

Model	FP32	FP16	BF16	FP8	W8A8 SQ	W8A16	W4A16	W4A16 AWQ	W4A16 GPTQ
Baichuan	Y	Y	Y	.	.	Y	Y	.	.
BERT	Y	Y	Y	.	.	.	.	.	.
BLOOM	Y	Y	Y	.	Y	Y	Y	.	.
ChatGLM	Y	Y	Y	.	.	.	.	.	.
ChatGLM-v2	Y	Y	Y	.	.	.	.	.	.
Falcon	Y	Y	Y	.	.	.	.	.	.
GPT	Y	Y	Y	Y	Y	Y	Y	.	.
GPT-J	Y	Y	Y	Y	Y	Y	Y	Y	.
GPT-NeMo	Y	Y	Y	.	.	.	.	.	.
GPT-NeoX	Y	Y	Y	.	.	.	.	.	Y
LLaMA	Y	Y	Y	.	Y	Y	Y	Y	Y
LLaMA-v2	Y	Y	Y	Y	Y	Y	Y	Y	Y
OPT	Y	Y	Y	.	.	.	.	.	.
SantaCoder	Y	Y	Y	.	.	.	.	.	.
StarCoder	Y	Y	Y	.	.	.	.	.	.

TensorRT-LLM Performance

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

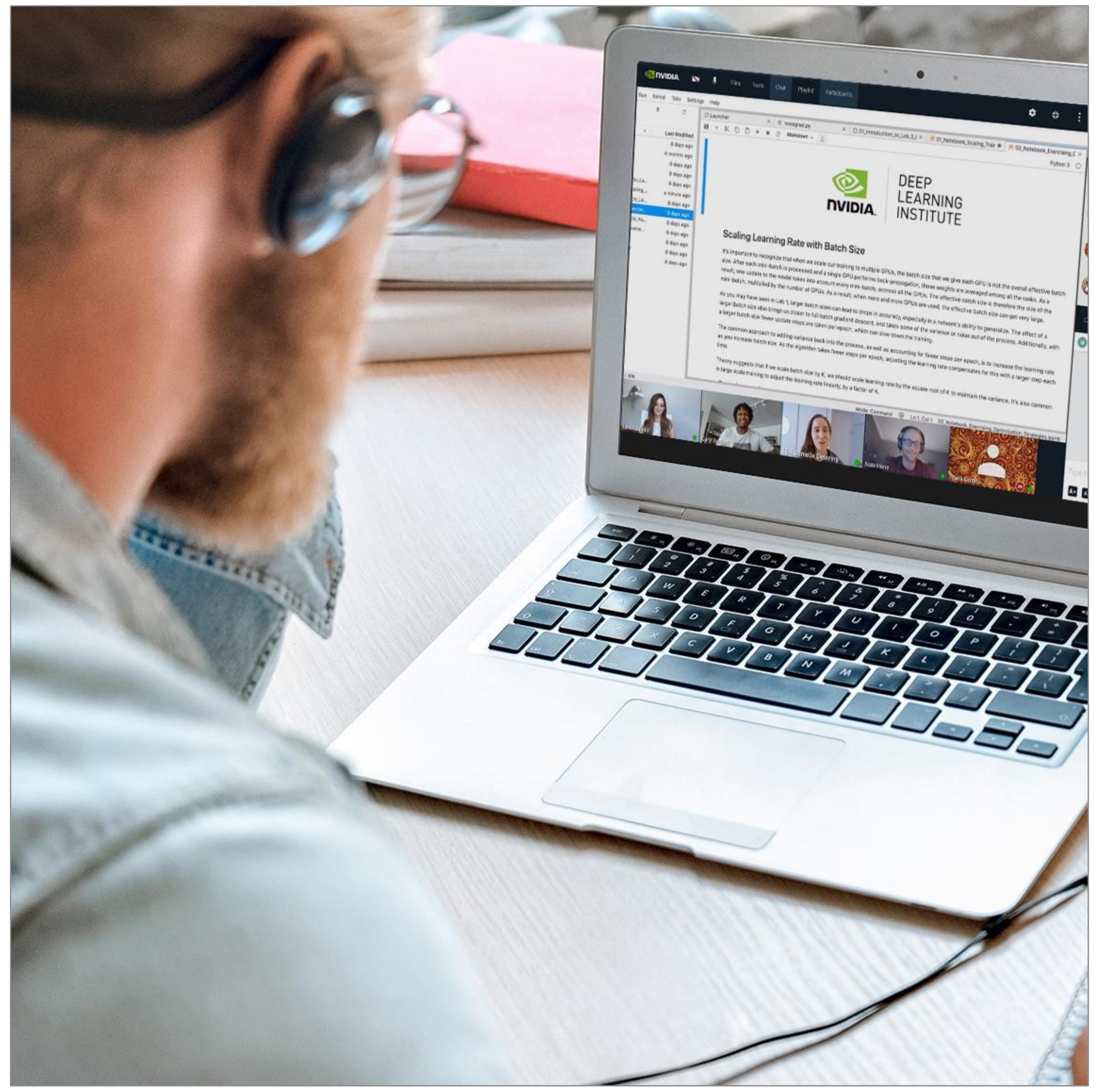


TensorRT-LLM v0.2.0 internal build. MPT-7B
1xA100-40GB. Averaged across BS [1, 512], seqlen [1, 512]

Demo – Quantization

NVIDIA Developer Ecosystem

Driving adoption with developers



NVIDIA Developer Program
4.5 Million Developers

- SDKs & Frameworks
- Early access programs
- NVIDIA On-Demand videos
- DevZone / Forums
- Technical blogs
- AMAs

NVIDIA Inception
15,000 Startups

- Acceleration program
- Cloud credits
- Go-to-market support

Deep Learning Institute (DLI)
450,000 Devs Trained

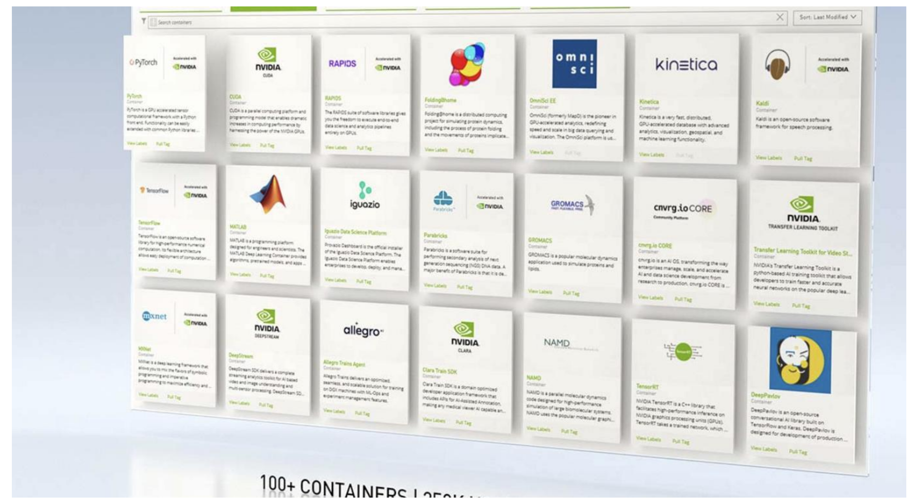
- Hands-on, self paced courses
- Live, instructor-led workshops
- Educator programs
- University Teaching Kits

Higher Education & Research Programs
600 Projects / Year

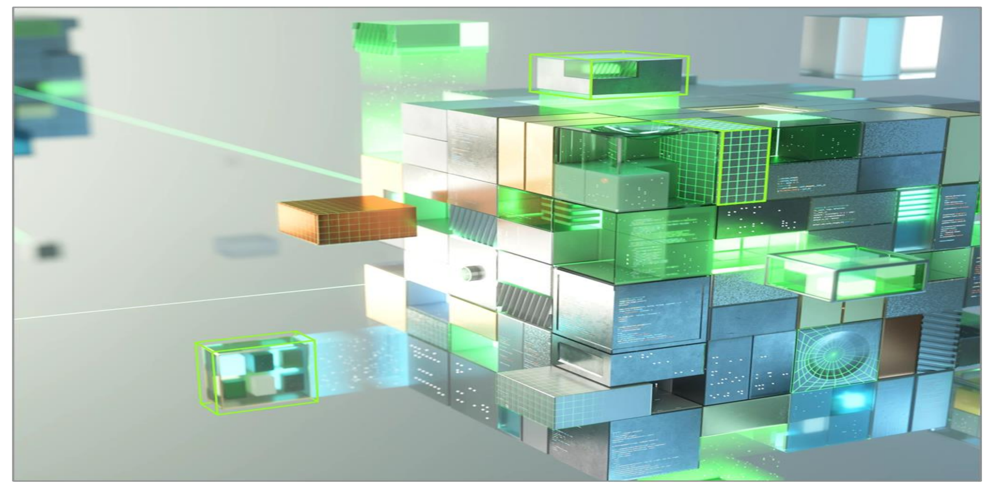
- NVIDIA Student Network
- Hackathons & Bootcamps
- Jetson Specialist Certification

NVIDIA Developer Program

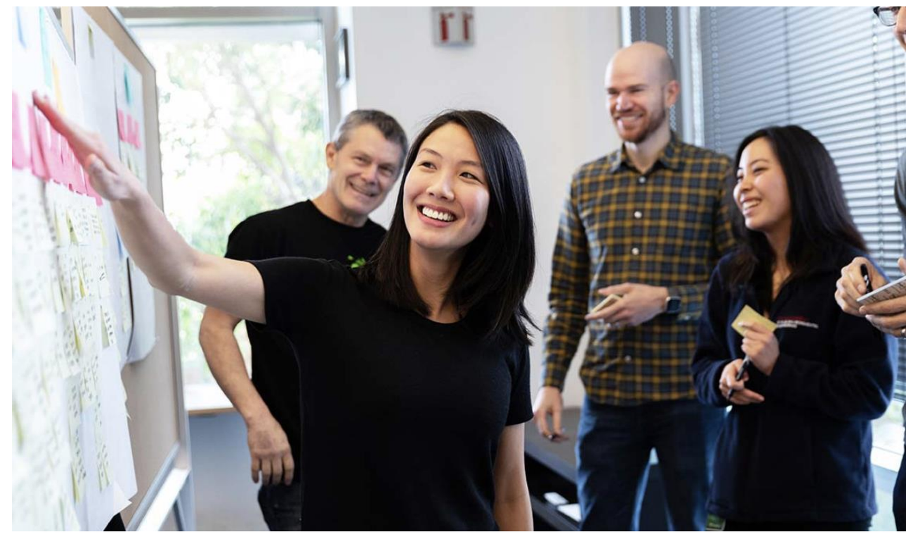
Benefits and resources



Access to Developer Tools



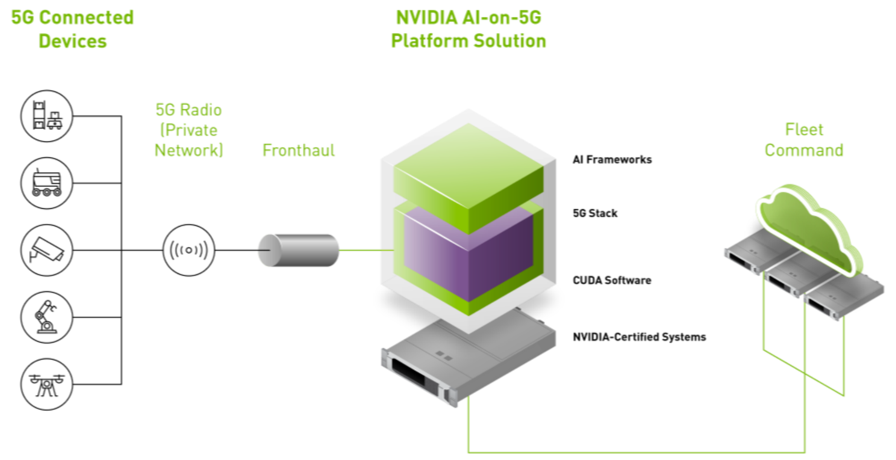
Early Access Programs



Webinars

Developer Newsletter

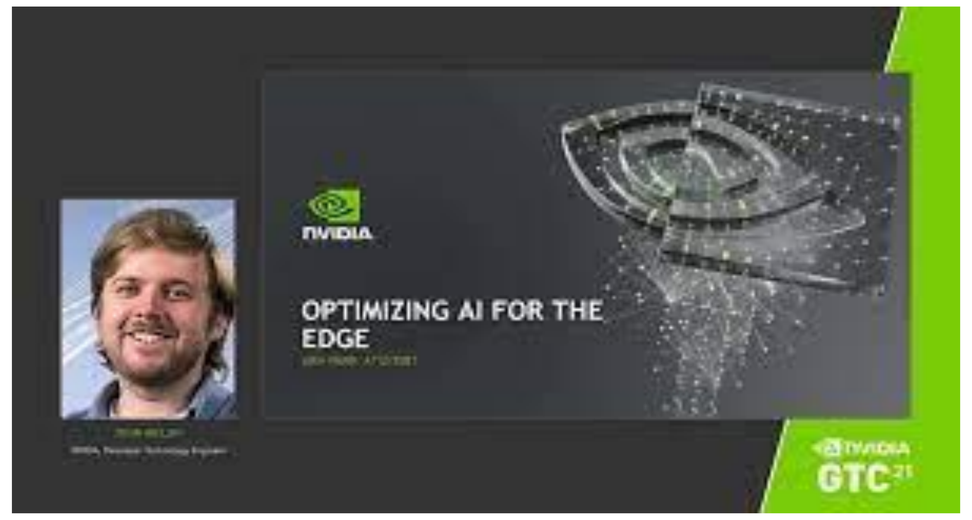
Developer Forums



Technical Blogs

Exclusive Invite-Only Events

Hands-on Training



NVIDIA On-Demand's Full Catalog