INFERENCE OF LARGE NEURAL NETWORKS

INFERENCE OF LARGE NEURAL NETWORKS

NVIDIA DEEP LEARNING INSTITUTE

PART 3

Inference Of Large Neural Networks

Lecture

- Overview of AI Inference Optimization techniques
- Distributed Inference
- TensorRT
- Faster Transformers
- Triton Inference Server
- Nemo Megatron

- Lab
  - Overview of the class environment
  - Hugging Face / Pytorch Inference for the GPT-J
  - Optimize GPT-J with Faster Transformers
  - Deploy GPT-J with Triton Inference Server

LARGE MODELS INFERENCE IS DIFFICULT

NEMO-MEGATRON WITH DGX SUPERPOD

Train what was once impossible

Algorithmic innovation
Train the world’s largest transformer-based language models using Megatron’s advanced optimizations and parallelization algorithms.

Direct access to world-class NLP experts
Access dedicated expertise from install to infrastructure management to scaling workloads to streamlined production AI.

Optimized Topology for Multi-Node Training
Train the largest models using model parallelism, with NVLINK and InfiniBand for fast cross-node communication.

Turnkey Experience for Rapid Deployment
A full-stack data center platform that includes industry-leading computing, storage, networking, software, and management tools.

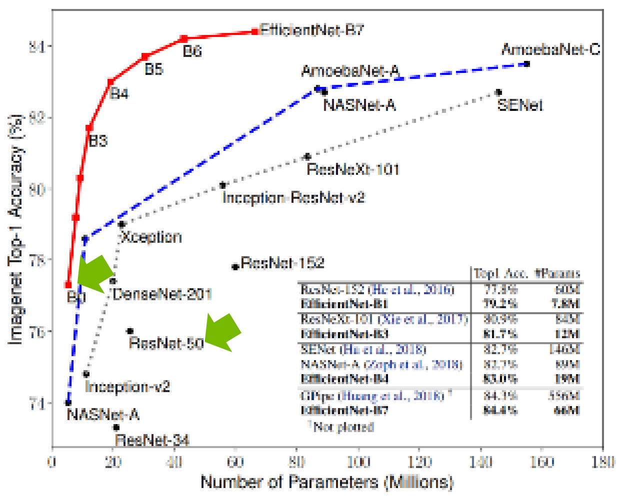
Efficiency at Extreme Scale
Training GPT-3 175B takes 355 years on a V100, 14.8 years on 1 DGX A100 and about 1 month on a 140-node DGX SuperPOD

LET’S DIVE INTO THE DETAILS

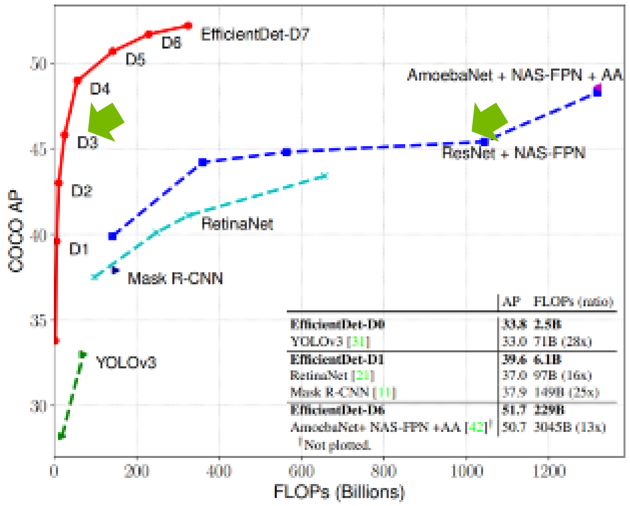
MODEL SELECTION

Not all models are created equally

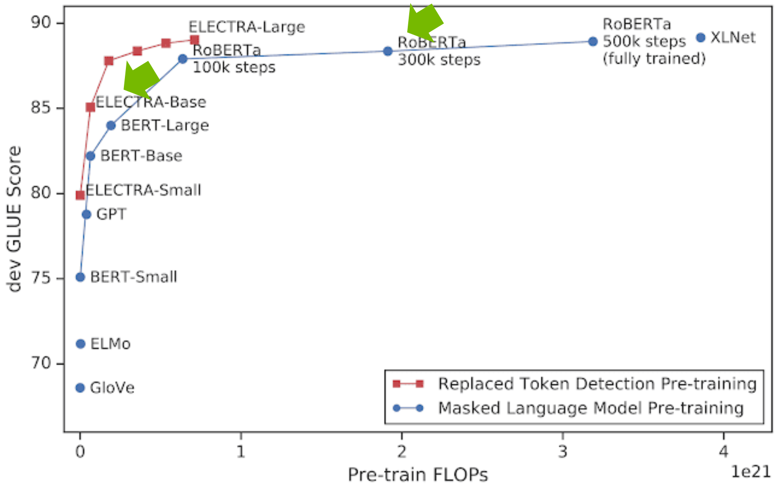
Image Classification



Object detection

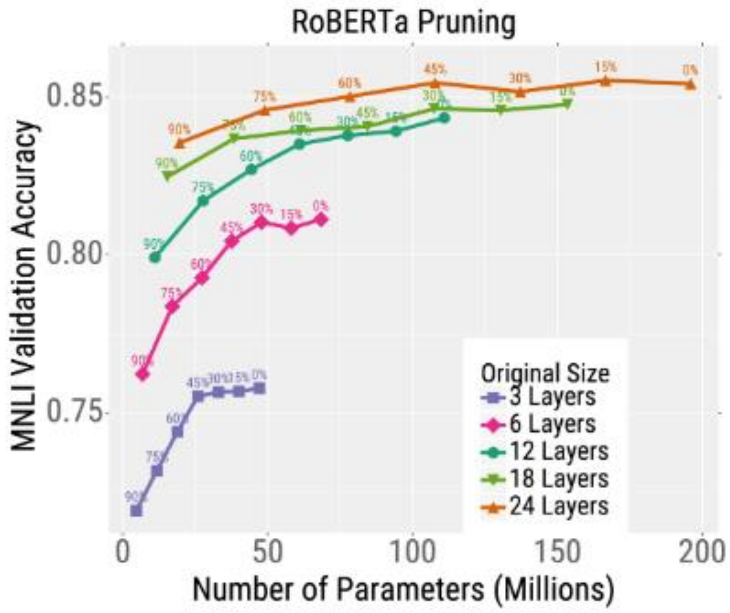


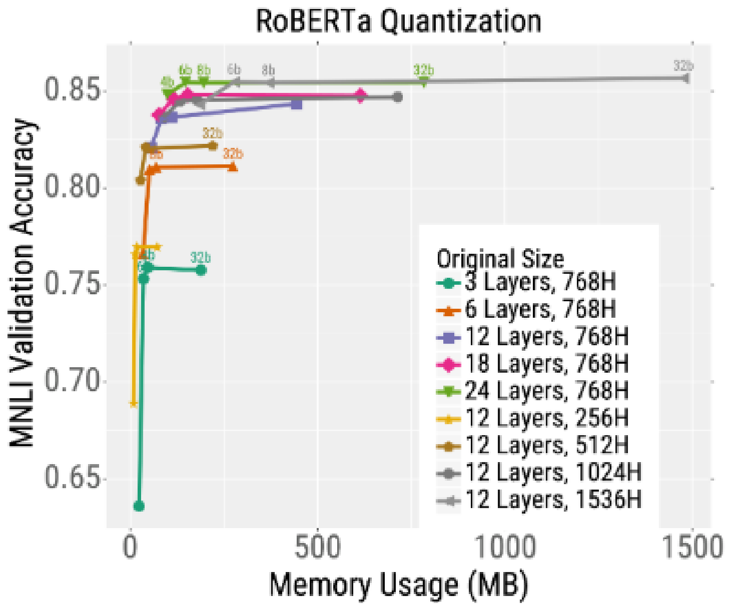
NLP



MODEL SELECTION

Not all models respond in the same way to knowledge distillation, pruning and quantization



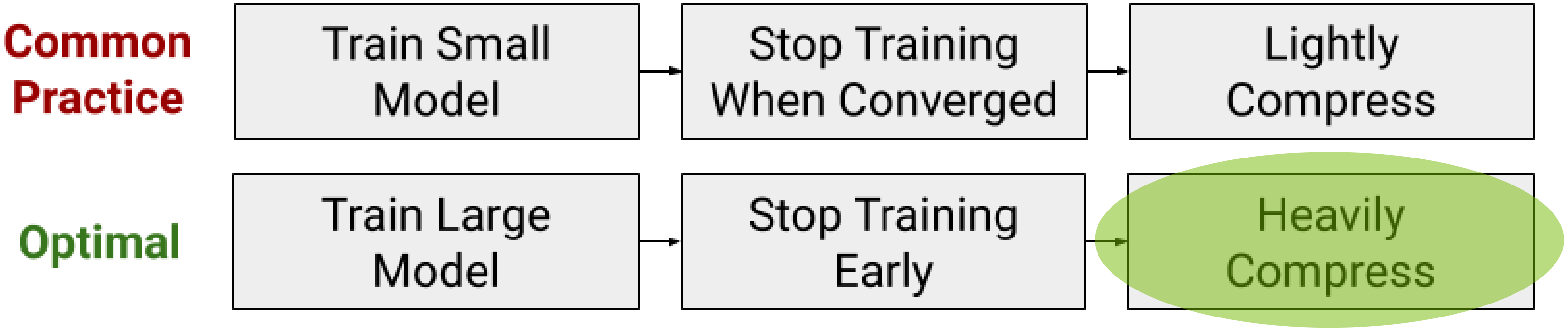


https://bair.berkeley.edu/blog/2020/03/05/compress/

Li, Z., Wallace, E., Shen, S., Lin, K., Keutzer, K., Klein, D., & Gonzalez, J. E. (2020). Train large, then compress: Rethinking model size for efficient training and inference of transformers. arXiv preprint arXiv:2002.11794.

INCREASING IMPORTANCE OF PRUNING AND QUANTIZATION

E.g. Train Large then compress



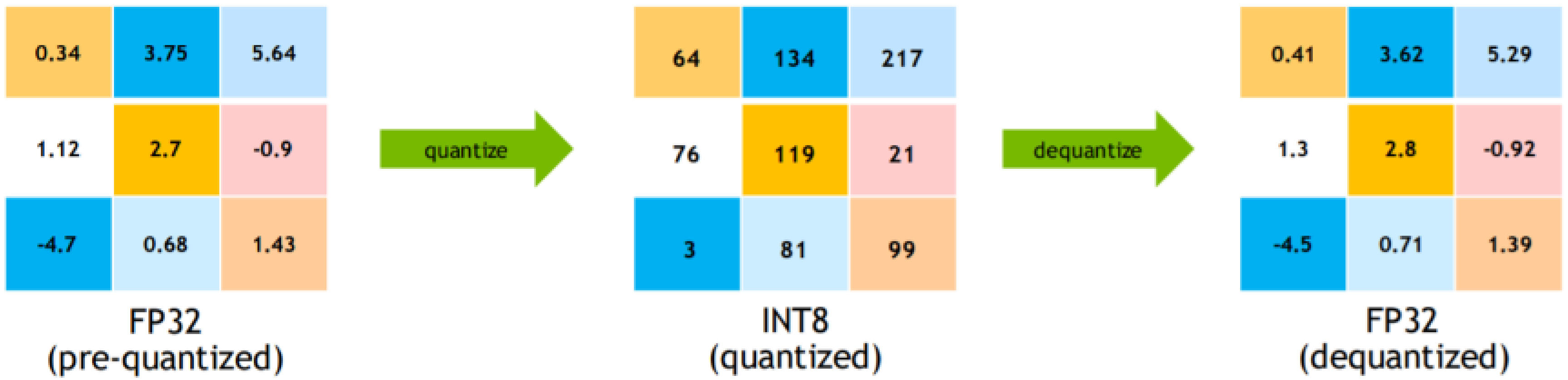
https://bair.berkeley.edu/blog/2020/03/05/compress/

Li, Z., Wallace, E., Shen, S., Lin, K., Keutzer, K., Klein, D., & Gonzalez, J. E. (2020). Train large, then compress: Rethinking model size for efficient training and inference of transformers. arXiv preprint arXiv:2002.11794.

QUANTIZATION

QUANTIZATION

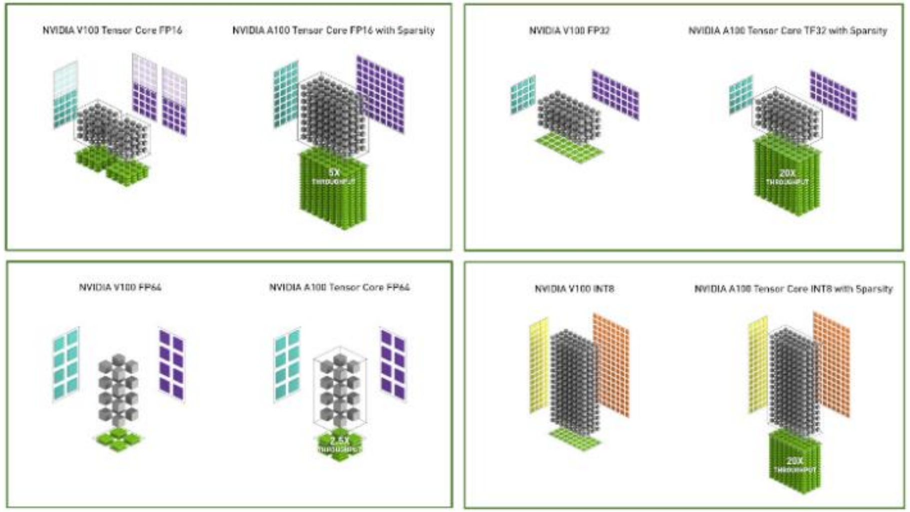
The idea



QUANTIZATION

The rationale

Input Datatype	Accumulation Datatype	Math Throughput	Bandwidth Reduction
FP32	FP32	1x	1x
FP16	FP16	8x	2x
INT8	INT32	16x	4x
INT4	INT32	32x	8x
INT1	INT32	128x	32x



QUANTIZATION

Approaches

Post-training quantization(PTQ)

Calibration data → Pre-trained model → Gather layer statistics → Compute *q-params* → Quantize model

Post-training quantization (PTQ)	Quantization-aware training (QAT)
Start with a pre-trained model and evaluate it on a calibration dataset.	Start with a pre-trained model and introduce quantization ops at various layers
Calibration data is used to calibrate the model. It can be a subset of training data.	Finetune it for a small number of epochs.
Calculate dynamic ranges of weights and activations in the network to compute quantization parameters (q-params).	Simulates the quantization process that occurs during inference.
Quantize the network using q-params and run inference	The goal is to learn the q-params which can help to reduce the accuracy drop between the quantized model and pre-trained model.

Quantization-aware training (QAT)

Pre-trained model → Add QAT ops → Finetune with QAT ops → Store *q-params* → Quantize model for inference

PRUNING

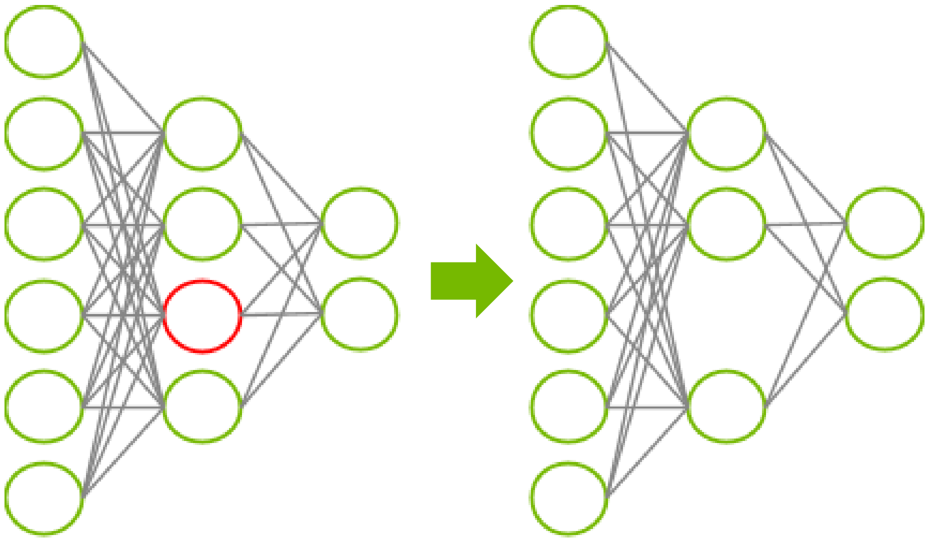
MODEL OPTIMIZATION

Pruning

Reduce the complexity of neural networks by removing unnecessary connections

- Reduce memory bandwidth
- Reduce memory footprint
- Accelerate the compute

Challenge: Maintain accuracy of the original unpruned network



6 inputs, 6 neurons (including 2 outputs), 32 connections

6 inputs, 5 neurons (including 2 outputs), 24 connections

SPARSITY SUPPORT

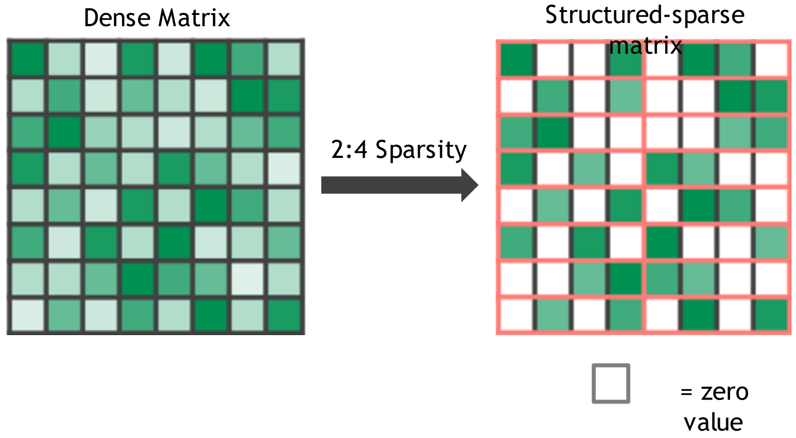
Accelerate Inference with Ampere Sparse Tensor Core

Maximize throughput at low latency with sparsity

New optimizations with 2:4 fine-grained structured sparsity result in greater performance reducing the weights in half

ASP (Automatic SParsity) provides easy-to-use workflow to induce the sparsity while maintaining accuracy of original dense network

TensorRT accelerates inference using sparse kernels




```
# Import ASP (Automatic SParsity)
from apex.contrib.sparsity import ASP

# In training phase, to augment the model and the optimizer #for
sparse training/inference
ASP.prune_trained_model(model, optimizer)
```

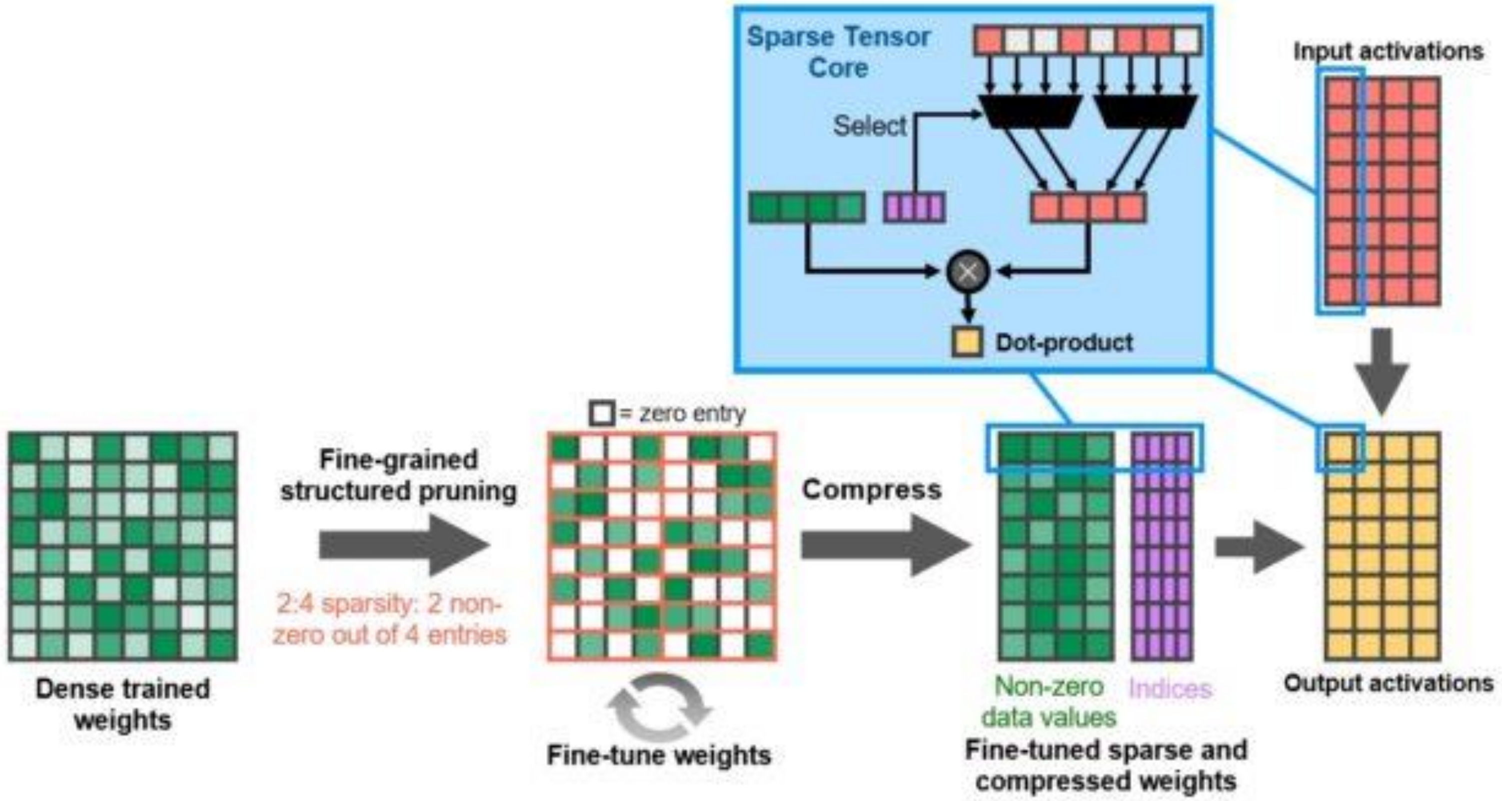
Training Phase

Enable **Sparsity** by setting the `kSPARSE_WEIGHTS` flag in `IBuilderConfig`

Inference Phase

developer.nvidia.com/tensorrt

SPARSITY IN A100 GPU

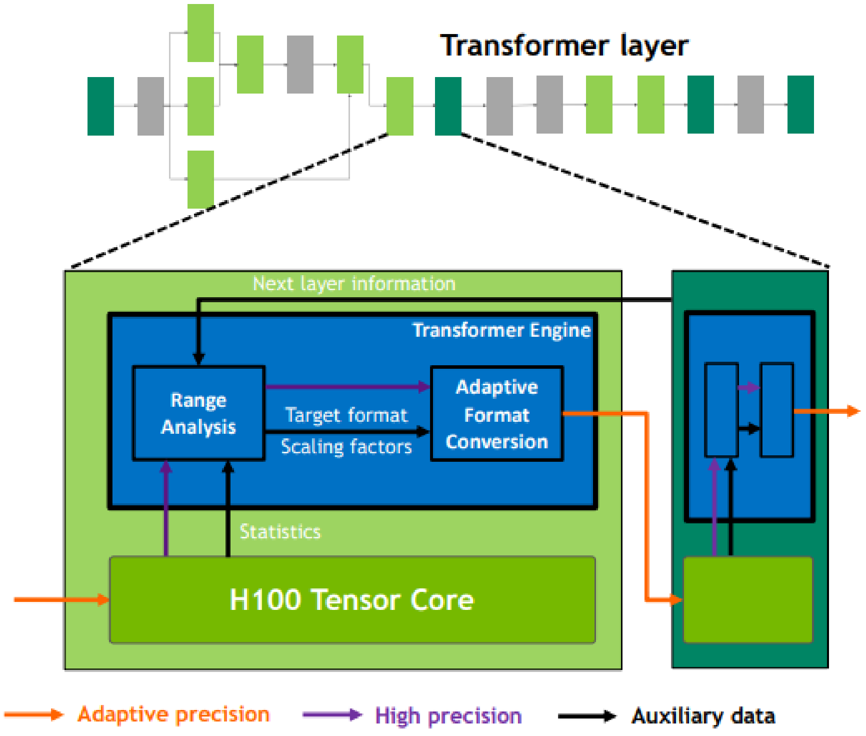


TRANSFORMER ENGINE

TRANSFORMER ENGINE

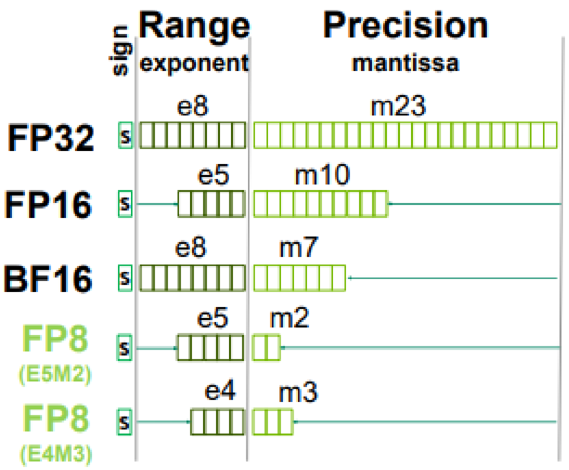
Optimal Transformer acceleration with Hopper Tensor Core

- Transparent to DL frameworks
- User can enable/disable
- Selectively applies new FP8 format for highest throughput
- Monitors tensor statistics and dynamically adjusts range to maintain accuracy

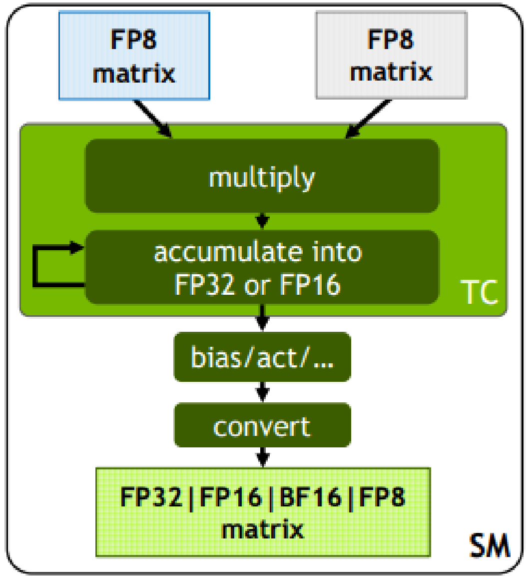


INSIDE THE NVIDIA HOPPER ARCHITECTURE

INSIDE 8-BIT FLOATING POINT (FP8)



Allocate 1 bit to either range or precision



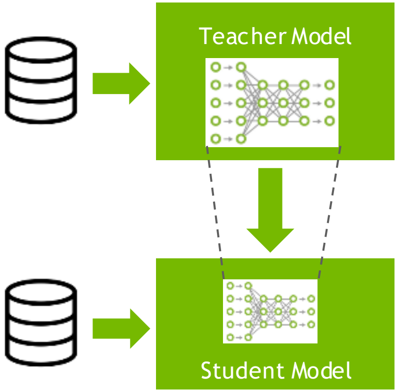
Support for multiple accumulator and output types

2x throughput & half footprint of FP16/BF16

DISTILLATION

KNOWLEDGE DISTILLATION

The idea



- Train a large model
- Use the trained model to train a smaller model

Distilling the Knowledge in a Neural Network

Geoffrey Hinton[*†]
Google Inc.
Mountain View
geoffhinton@google.com

Oriol Vinyals[†]
Google Inc.
Mountain View
vinyals@google.com

Jeff Dean
Google Inc.
Mountain View
jeff@google.com

Abstract

A very simple way to improve the performance of almost any machine learning algorithm is to train many different models on the same data and then to average their predictions [3]. Unfortunately, making predictions using a whole ensemble of models is cumbersome and may be too computationally expensive to allow deployment to a large number of users, especially if the individual models are large neural nets. Caruana and his collaborators [1] have shown that it is possible to compress the knowledge in an ensemble into a single model which is much easier to deploy and we develop this approach further using a different compression technique. We achieve some surprising results on MNIST and we show that we can significantly improve the acoustic model of a heavily used commercial system by distilling the knowledge in an ensemble of models into a single model. We also introduce a new type of ensemble composed of one or more full models and many specialist models which learn to distinguish fine-grained classes that the full models confuse. Unlike a mixture of experts, these specialist models can be trained rapidly and in parallel.

KNOWLEDGE DISTILLATION

DistillBERT

Table 1: **DistilBERT retains 97% of BERT performance.** Comparison on the dev sets of the GLUE benchmark. ELMo results as reported by the authors. BERT and DistilBERT results are the medians of 5 runs with different seeds.

Model	**Score**	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B	WNLI
ELMo	68.7	44.1	68.6	76.6	71.1	86.2	53.4	91.5	70.4	56.3
BERT-base	79.5	56.3	86.7	88.6	91.8	89.6	69.3	92.7	89.0	53.5
DistilBERT	77.0	51.3	82.2	87.5	89.2	88.5	59.9	91.3	86.9	56.3

Table 2: **DistilBERT yields to comparable performance on downstream tasks.** Comparison on downstream tasks: IMDb (test accuracy) and SQuAD 1.1 (EM/F1 on dev set). D: with a second step of distillation during fine-tuning.

Model	IMDb (acc.)	SQuAD (EM/F1)
BERT-base	93.46	81.2/88.5
DistilBERT	92.82	77.7/85.8
DistilBERT (D)	-	79.1/86.9

Table 3: **DistilBERT is significantly smaller while being constantly faster.** Inference time of a full pass of GLUE task STS-B (sentiment analysis) on CPU with a batch size of 1.

Model	# param. (Millions)	Inf. time (seconds)
ELMo	180	895
BERT-base	110	668
DistilBERT	66	410

DistillBERT retain 97% of BERT performance while is only 66M parameters

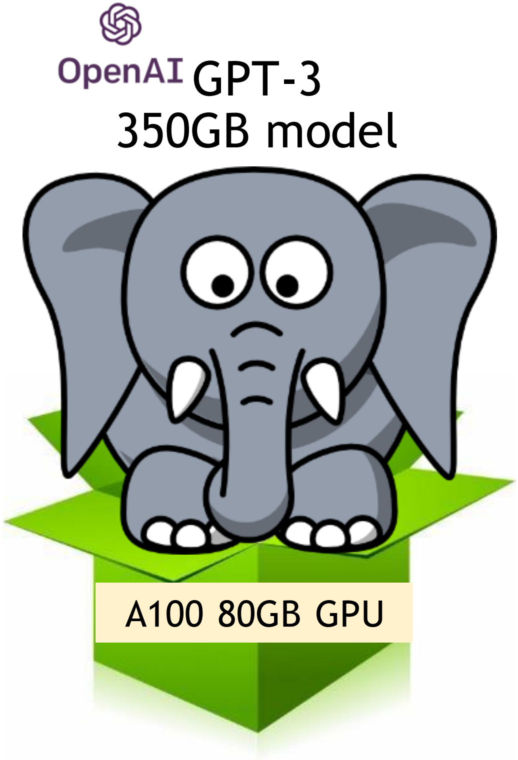
Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. arXiv preprint arXiv:1910.01108.

INFERENCE OF HUGE MODELS

INFERENCE OF HUGE MODELS

Goals and Challenges

- **Goal: To infer huge models in an efficient and convenient way, including**
    - Maximizing Utilization of GPUs
    - A unified and simple inference solution for many models in production
    - Easier deployments, scaling and support
    - Maximizing Throughput, Minimizing Latency
- **Challenges:**
    - Huge model requires more memory than available on 1 GPU
    - There are no tools to infer Huge Models, apart from Triton
    - Model needs to be optimized before the inference
    - Frameworks used for training Huge Models are quite complex and inadequate for inference



PRODUCTION DEPLOYMENT

Executive Math

175B parameters

2 bytes per weight
(FP16)

$$\frac{175 * 10^9 * 2}{1024 * 1024 * 1024} = 325.96GB$$

We want GPU memory
consumption in GB

PRODUCTION DEPLOYMENT

Pruning - 2:4 Structured Sparsity

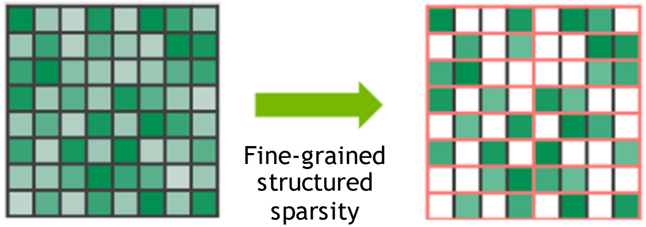
175B parameters

2 bytes per weight
(FP16)

$$\frac{175 * 10^9 * 2}{1024 * 1024 * 1024 * 2} = 162.98GB$$

We want GPU memory
consumption in GB

Remove half of
the weights



Dense Matrix

Sparse Matrix

https://developer.nvidia.com/blog/accelerating-inference-with-sparsity-using-ampere-and-tensorrt/

PRODUCTION DEPLOYMENT

Quantization

175B parameters

1 byte per weight
(INT8)

$$\frac{175 * 10^9 * 1}{1024 * 1024 * 1024 * 2} = 81.49GB$$

We want GPU memory
consumption in GB

Remove half of
the weights

0.34	3.75	5.64
1.12	2.7	-0.9
-4.7	0.68	1.43

FP16

Quantization

64	134	217
76	119	21
3	81	99

INT8

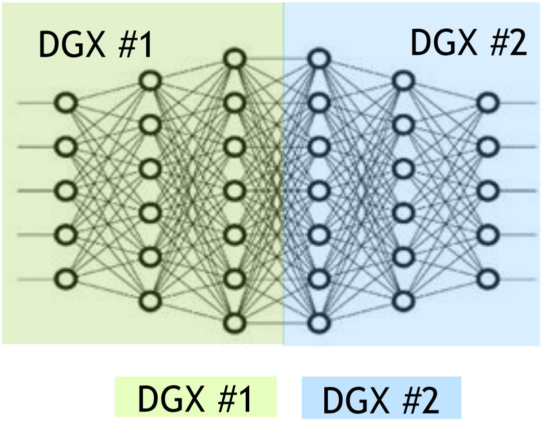
https://developer.nvidia.com/blog/achieving-fp32-accuracy-for-int8-inference-using-quantization-aware-training-with-tensorrt/

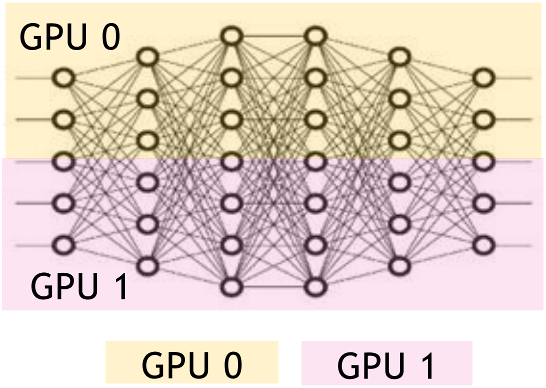
DISTRIBUTED INFERENCING

MODEL PARALLELISM

Complementary Types of Model Parallelism

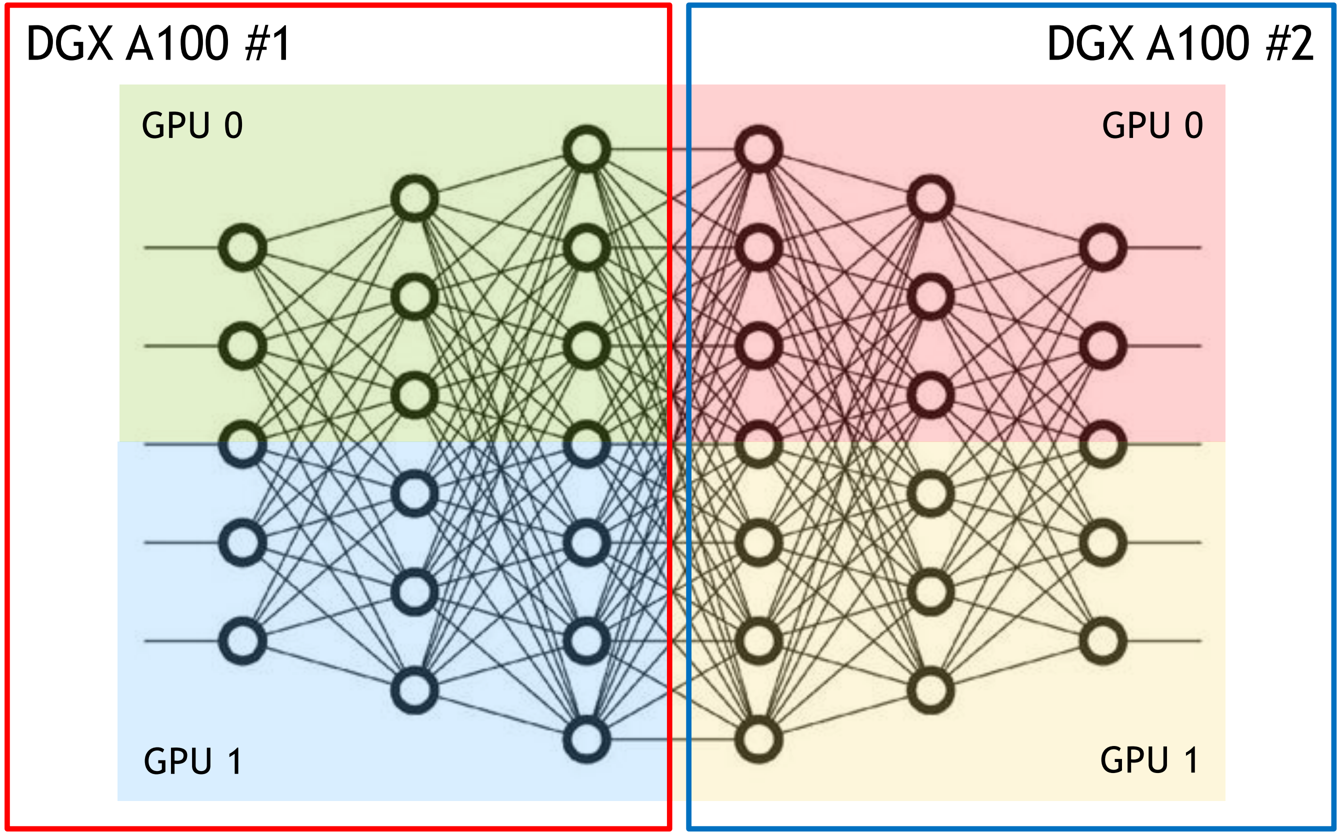
- Inter-Layer (Pipeline) Parallelism
  - Split sets of layers across multiple devices
  - ***Inference:***
    - ***Maximizes GPU utilization and Throughput***
    - ***Can be used easily with TRITON***
- Intra-Layer (Tensor) Parallelism
  - Split individual layers across multiple devices
  - ***Inference:***
    - ***Minimizes latency***





MODEL PARALLELISM

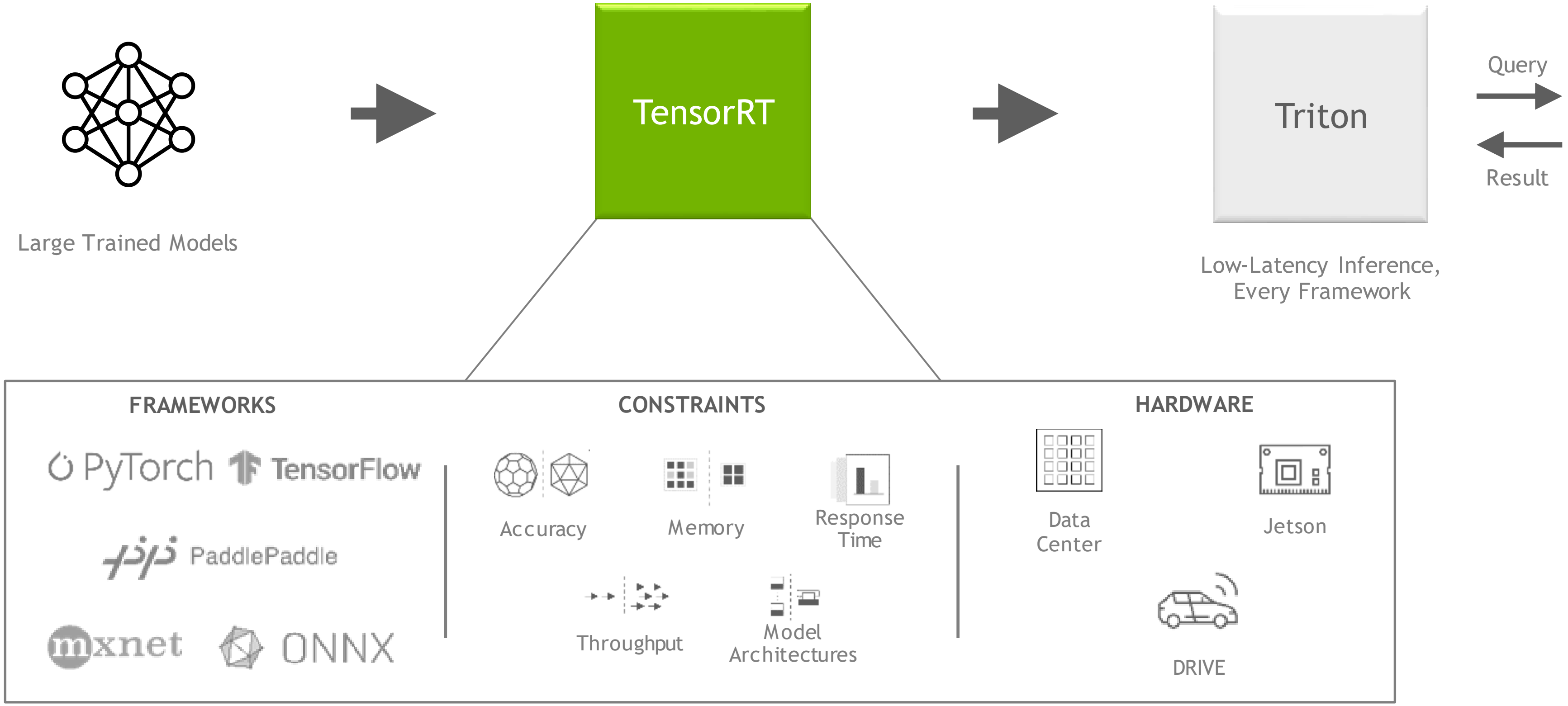
Combined Model Parallelism. Multiple GPUs in Multiple DGXs.



Inter + Intra Parallelism

INFERENCE IS COMPLEX

Real-Time | Competing Constraints | Rapid Updates



INFERENCE APPROACHES BY NVIDIA

LARGE SCALE NLP DEPLOYMENT

TensorRT vs FasterTransformer

TensorRT	FasterTransformer
No support for model parallelism Pipeline parallelism achieved using Triton Inference Server	Supports both tensor and pipeline parallelism
Support for a variety of models and types of layers Transformers: BERT, GPT, and T5	Limited support to BERT, GPT-2, Megatron GPT-3
Integration with Triton Inference Server	Integration with the Triton Inference Server
Additional steps are required to deploy large scale transformer Model TensorRT 8.2 supports GPT-2 up to 1.5B parameters and T5 up to 11B parameters	Supports large scale transformers
Fastest inference BERT like models	Fastest inference for GPT-3 like models

TENSORRT

NVIDIA TensorRT

SDK for High-Performance Deep Learning Inference

Optimize and deploy neural networks in production.

Maximize throughput for latency-critical apps with compiler and runtime.

Optimize every network, including CNNs, RNNs, and Transformers.

1. Reduced mixed precision: FP32, TF32, FP16, and INT8.

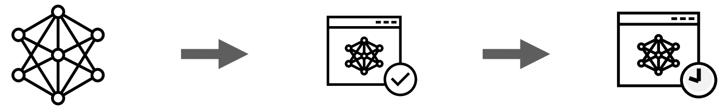
2. Layer and tensor fusion: Optimizes use of GPU memory bandwidth.

3. Kernel auto-tuning: Select best algorithm on target GPU.

4. Dynamic tensor memory: Deploy memory-efficient apps.

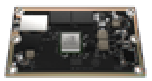
5. Multi-stream execution: Scalable design to process multiple streams.

6. Time fusion: Optimizes RNN over time steps.



Trained DNN | TensorRT Optimizer | TensorRT Runtime

Embedded | Automotive | Data Center



Jetson

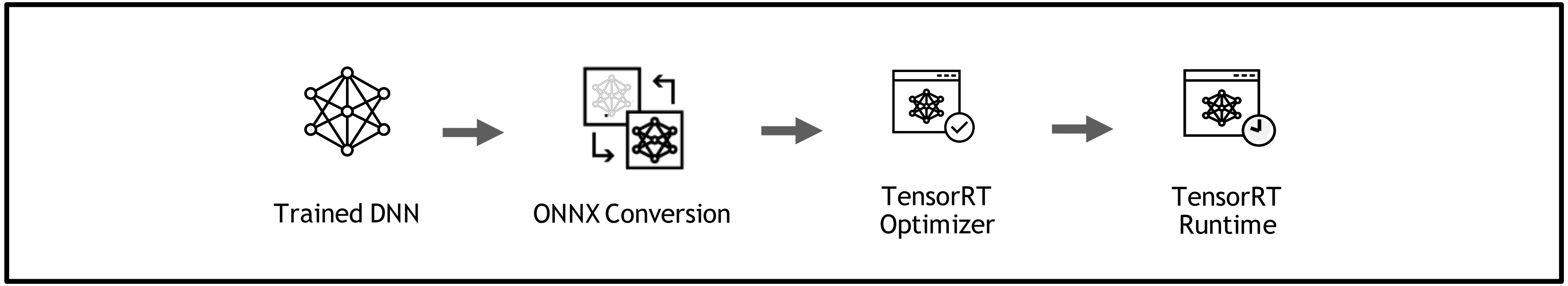
Drive

Data Center GPUs

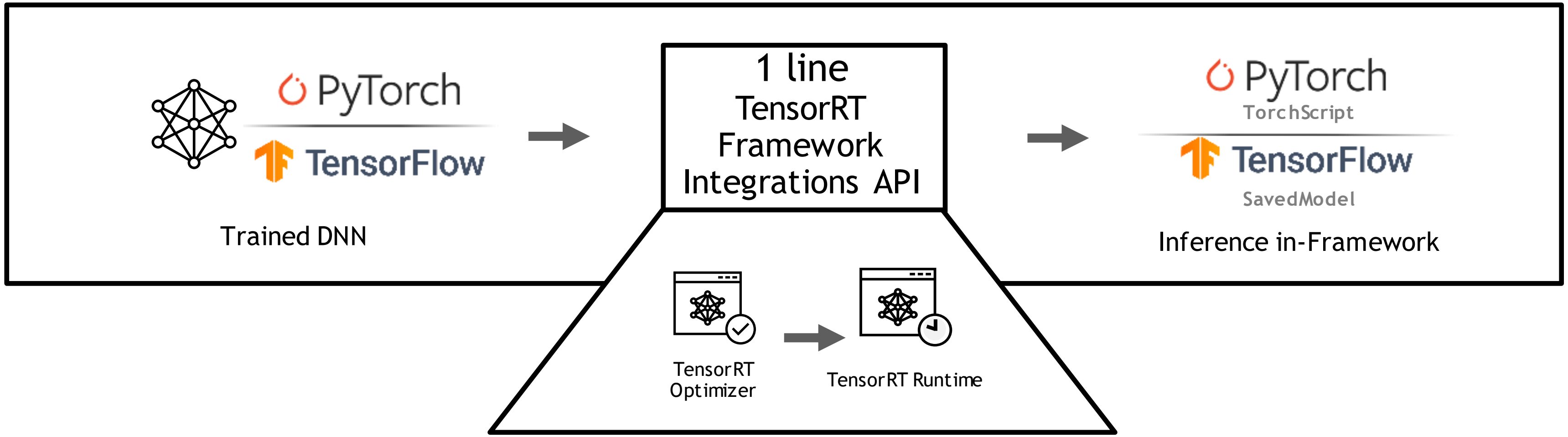
https://developer.nvidia.com/tensorrt

INFERENCE OPTIMIZATION WORKFLOW FOR TensorRT

TensorRT



TensorRT Framework Integrations



TensorRT INTEGRATED WITH PYTORCH AND TENSORFLOW

6x FASTER INFERENCE WITH 1 LINE OF CODE

Torch-TensorRT

PyTorch

```
import torch
import torch_tensorrt as torchtrt

# SET trained model to evaluation mode
model = model.eval()

# COMPILE TRT module using Torch-TensorRT
trt_module = torchtrt.compile(model,
inputs=[example_input],enabled_precisions={torch.half})

# RUN optimized inference with Torch-TensorRT
trt_module(x)
```

Available in PyTorch NGC Container

TensorFlow-TensorRT

TensorFlow

```
import tensorflow as tf
from tf.python.compiler.tensorrt import trt_convert as tftrt

# COMPILE TRT module using TensorFlow-TensorRT
trt_module =
tftrt.TrtGraphConverterV2(saved_model_pth).convert()

# RUN optimized inference with TensorFlow-TensorRT
trt_module(x)
```

Available in TensorFlow & NGC Container

WORLD LEADING INFERENCE PERFORMANCE

TensorRT Accelerates Every Workload

BEST IN CLASS RESPONSE TIME AND THROUGHPUT **vs CPUs**

36X Computer Vision < 7ms	**583X** Speech Recognition < 100ms	**21X** NLP < 50ms
10X Reinforcement Learning	**178X** Text-to-Speech < 100ms	**12X** Recommenders < 1 sec

TENOSRRT

TensorRT transformer optimization specifics

TensorRT optimizes the self-attention block by pointwise layer fusion:

- Reduction is fused with power ops (for LayerNorm and residual-add layer)
- Scale is fused with softmax
- GEMM is fused with ReLU/GELU activations

TensorRT also optimizes the network for inference:

- Eliminating transpose ops
- Fusing the three KQV projections into a single GEMM
- FP16 mode: Control the layer-wise precisions to preserve accuracy while running the most compute-intensive ops in FP16

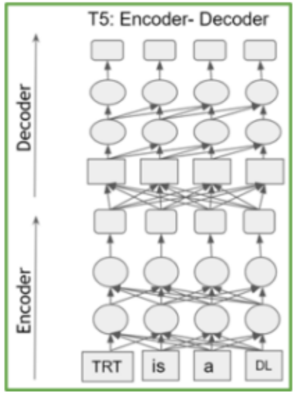


Figure 1a. T5 architecture

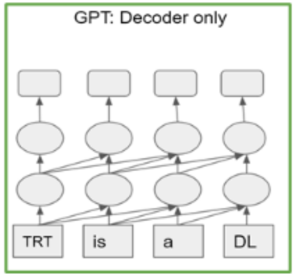
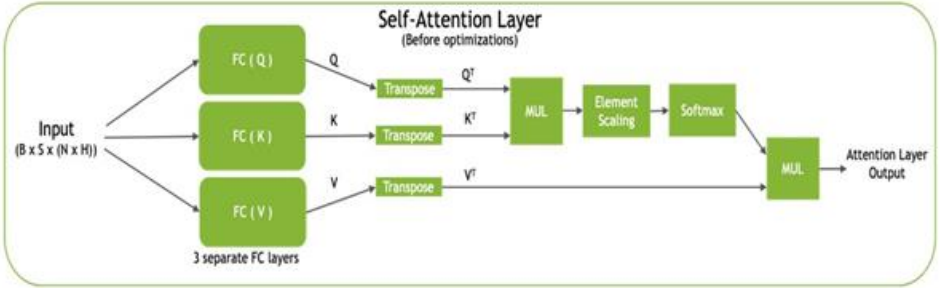


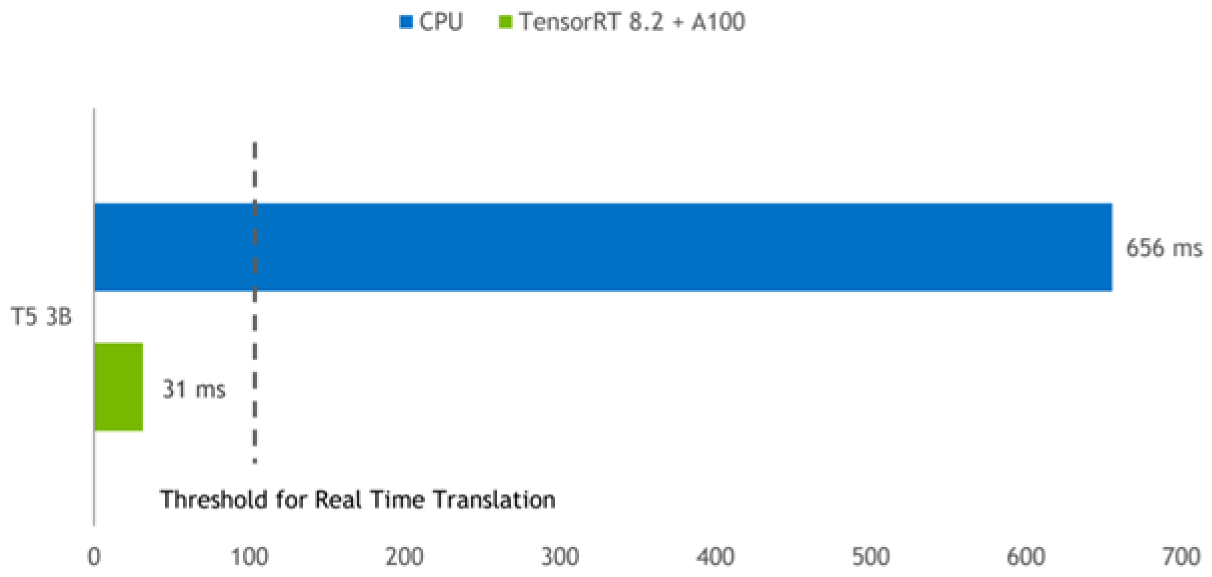
Figure 1b. GPT-2 architecture



https://developer.nvidia.com/blog/optimizing-t5-and-gpt-2-for-real-time-inference-with-tensorrt/

TENOSRRT

Inference libraries by NVIDIA



***T5-3B** model inference comparison. TensorRT on A100 GPU provides a 21x smaller latency compared to PyTorch CPU inference.*

CPU: Intel Platinum 8380, 2 sockets.
GPU: NVIDIA A100 PCI Express 80GB. Software: PyTorch 1.9, TensorRT 8.2.0 EA.
Task: "Translate English to German: that is good."

https://developer.nvidia.com/blog/optimizing-t5-and-gpt-2-for-real-time-inference-with-tensorrt/

NVIDIA FASTER TRANSFORMER

FASTERTRANSFORMER

Summary

FasterTransformer: Highly optimized transformer-based encoder and decoder component for inference

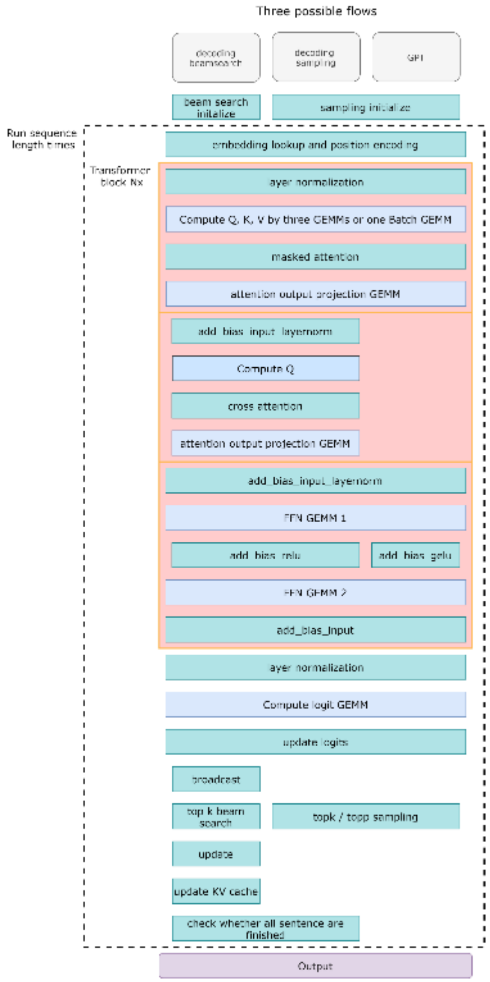
Based on CUDA and cuBLAS

- Encoder transformer: BERT
- Decoder transformer: GPT-2, Megatron-GPT-3 and OpenNMT-tf
- Decoding contains whole process of translation: OpenNMT-tf

Support FP32, FP16 and INT8

Provide C++ API and TensorFlow/PyTorch OP

FasterTransformer backend for Triton Inference Server (Alpha): multi-GPU, multi-node models (GPT and T5) with billions of parameters



Bo Yang Hsueh, NVIDIA, GTC 2020 : In-depth video about Faster Transformer
https://developer.nvidia.com/gtc/2020/video/s21417-vid

FASTERTRANSFORMER

Summary

- **Checkpoint converter**
- Huggingface
    - Megatron
    - Nemo Megatron
    - TensorFlow
- **Data type**
    - FP32
    - FP16
    - INT8 weight only PTQ for bs 1 and 2

- **Feature**
    - Multi-GPU multi-node inference
    - Dynamic random seed
    - Stop tokens
    - Beam search and sampling are both supported
    - FP32, FP16 and INT8 inference
- **Frameworks**
    - TensorFlow
    - PyTorch
    - C++
    - Triton backend

Bo Yang Hsueh, NVIDIA, GTC 2020 : In-depth video about Faster Transformer
https://developer.nvidia.com/gtc/2020/video/s21417-vid

OPTIMIZATIONS

1.Layer Fusion

2. Inference optimization for autoregressive models

3. Memory optimization

4. Usage of MPI and **NCCL** to enable inter/intra node communication and support Model parallelism

5. MatMul kernel autotuning (GEMM Autotuning)

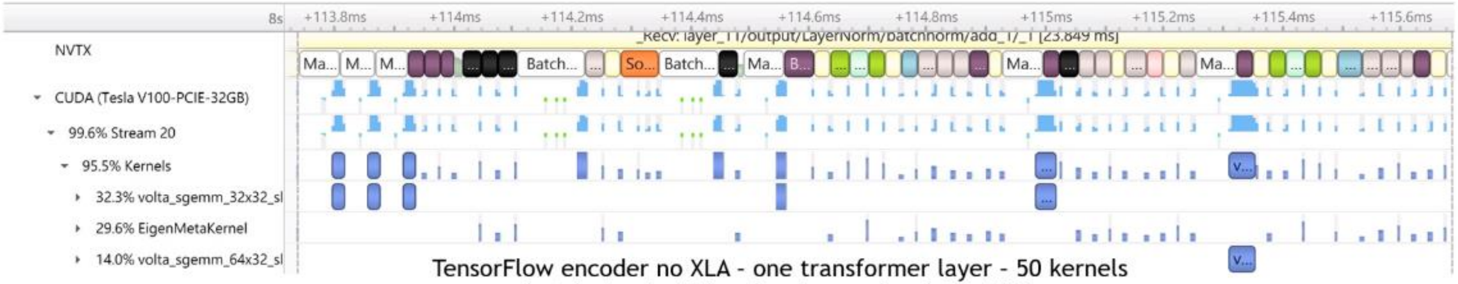
6. Inference with lower precisions and quantization

7. Others:

    1. Rapidly fast C++ BeamSearch implementation
    2. Optimized all-reduce for the TensorParallelism 8 mode. When eights of the model are split between 8 GPUs

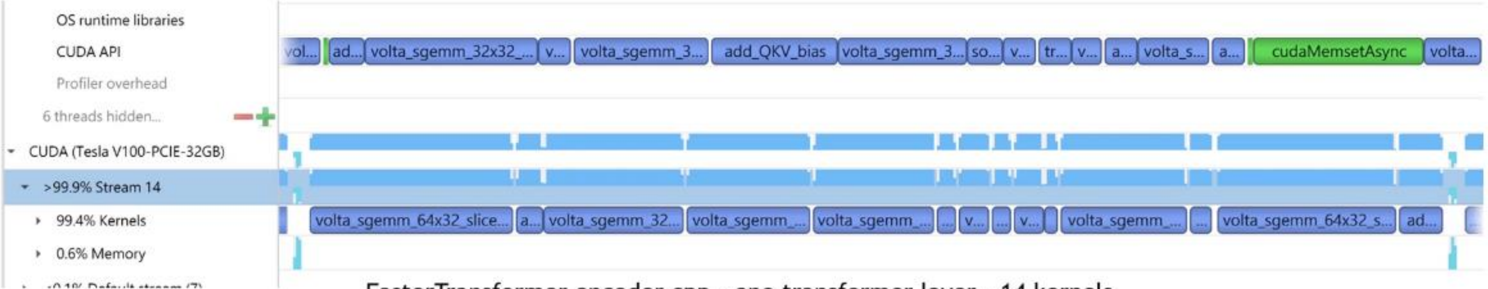
OPTIMIZATIONS LEAD TO THIS

Encoder Inference in the Framework



TensorFlow encoder no XLA - one transformer layer - 50 kernels
batch size 1, 12 heads, size per head 64, FP32

Encoder Inference in the FasterTransformer



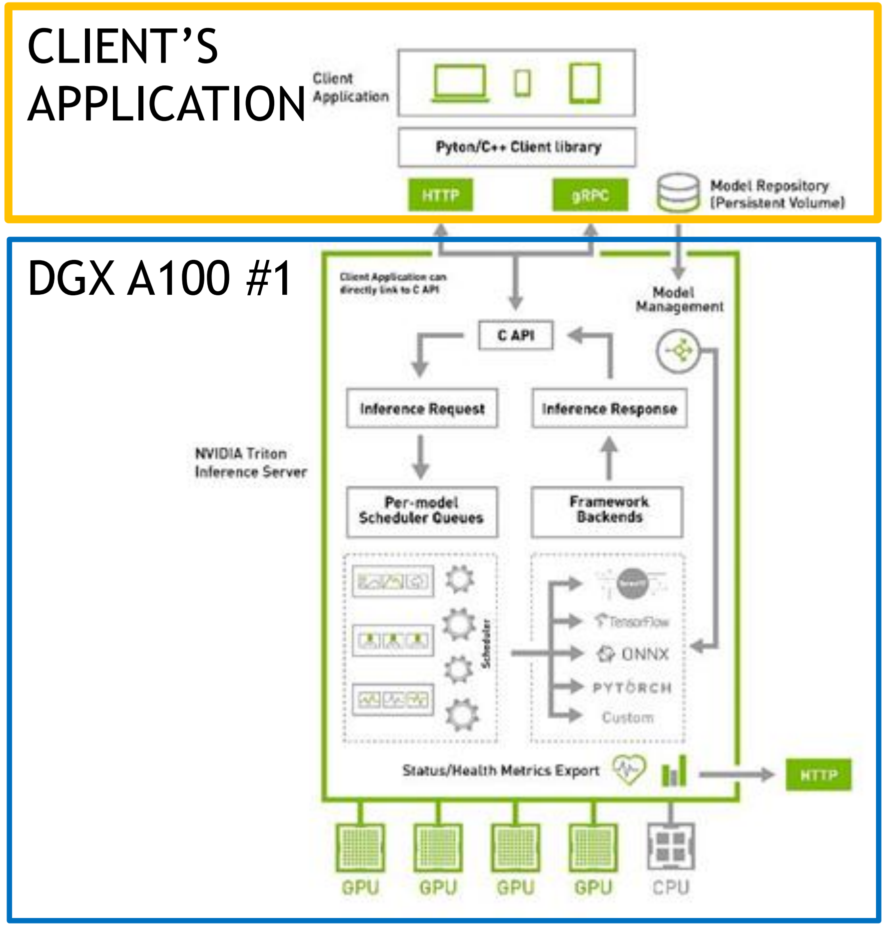
FasterTransformer encoder cpp - one transformer layer - 14 kernels
batch size 1, 12 heads, size per head 64, FP32

Bo Yang Hsueh, NVIDIA, GTC 2020 : In-depth video about Faster Transformer
https://developer.nvidia.com/gtc/2020/video/s21417-vid

TRITON INFERENCE SERVER

TRITON: INFERENCE SERVER ARCHITECTURE

Easy to Use



Pretrained Neural Network is placed on DGX and ready for inference with TRITON

DEVELOPERS CAN FOCUS ON MODELS AND APPLICATIONS

Triton Takes Care of Plumbing To Deploy Models for Inference

Multiple Frameworks

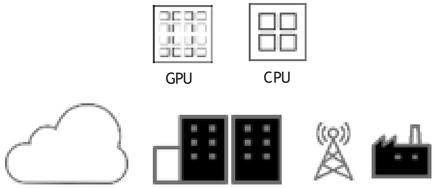
Custom

All Major Framework Backends For Flexibility & Consistency

Concurrent Model Execution For High Throughput & Utilization, lower TCO

Standard HTTP/gRPC Communication

Inferencing on GPU and CPU

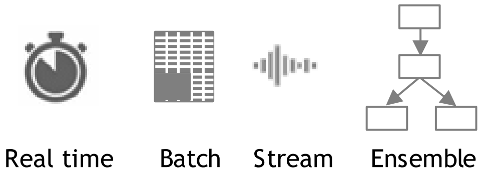


Inference Serving on GPU & CPU Across

Cloud | Data Center | Edge

Bare metal | Virtualization

Different Types of Queries



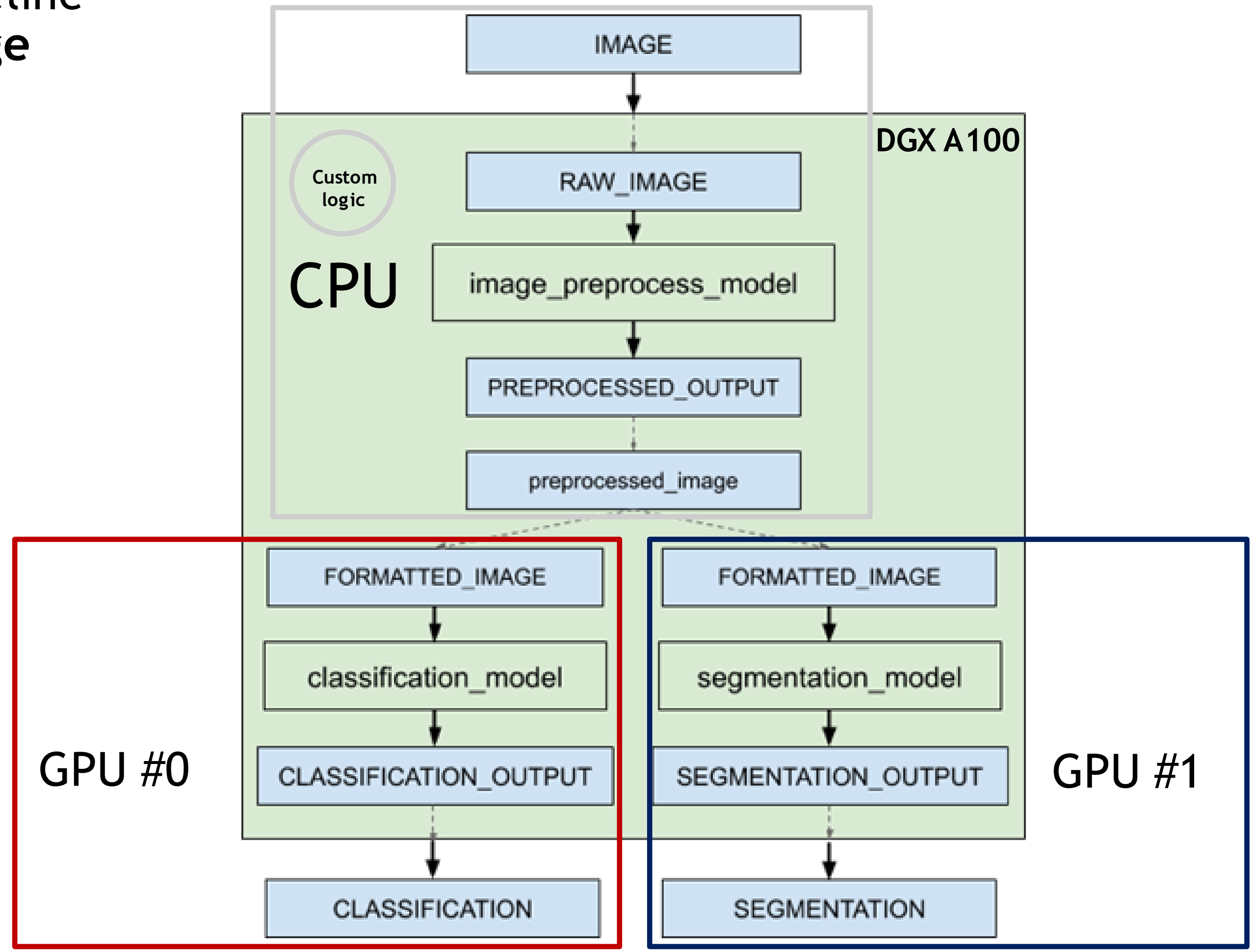
Support for Different Types of Inference Queries Used in Different Use Cases

Dynamic Batching

Dynamic Batching Maximizes Throughput Under Latency Constraint

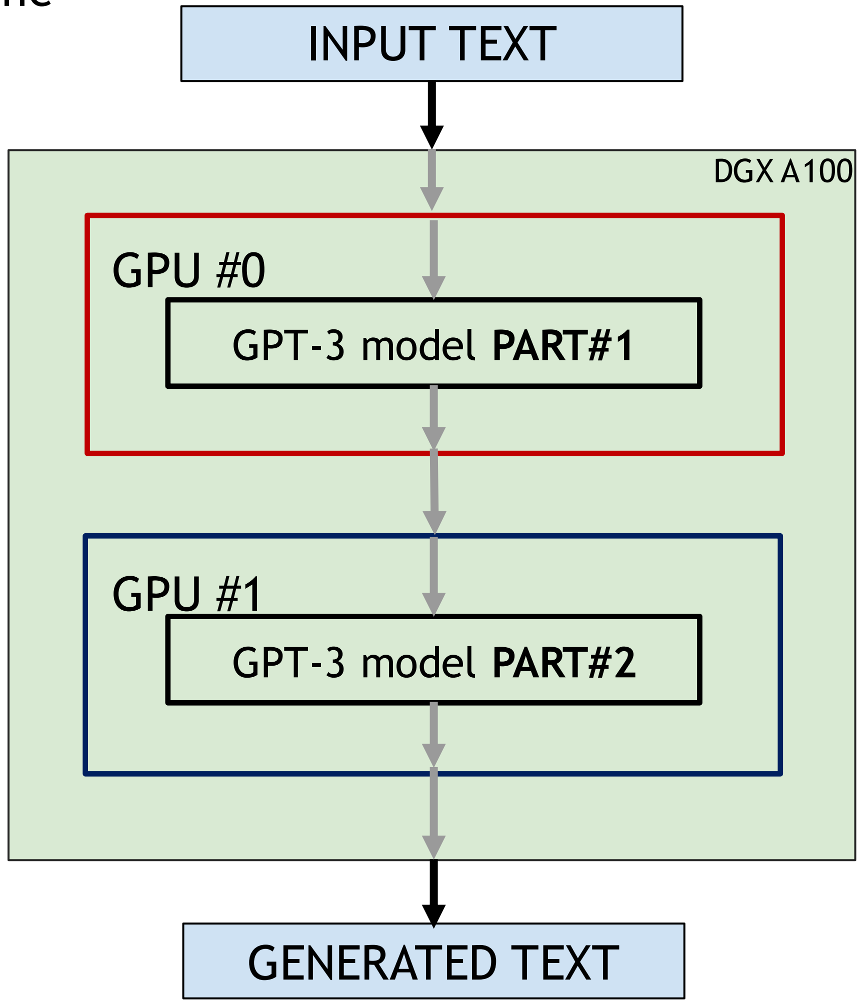
MODEL ENSEMBLING

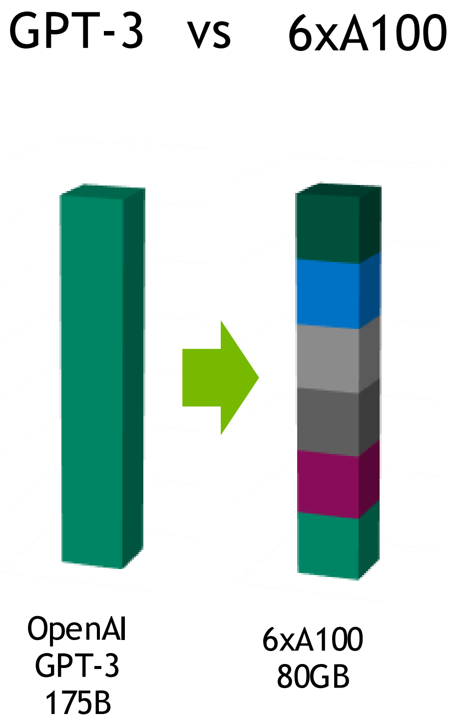
It’s easy to create pipeline for parts of **one huge model in TRITON**



MODEL ENSEMBLING

It's easy to create pipeline for parts of **one huge model in TRITON**





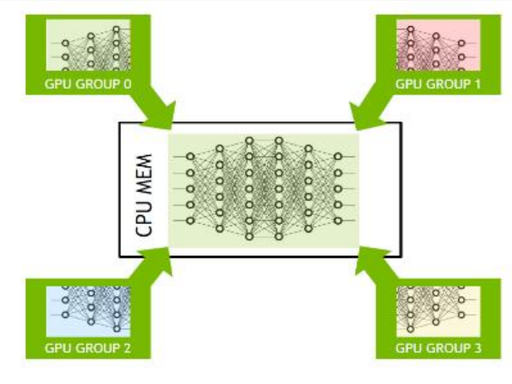
HOW TO DEPLOY LARGE MODELS?

TENSORRT+ TRITON INFERENCE SERVER

LARGE SCALE NLP DEPLOYMENT

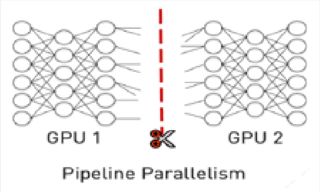
TensorRT and Triton Inference Server

1. Merge Checkpoints



2. Pipeline Split

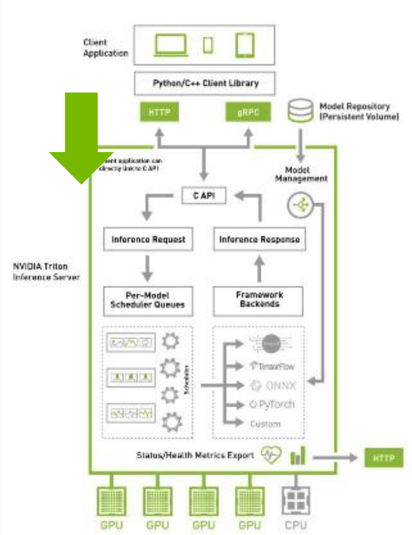
ONNX Graph Surgeon



3. TensorRT Optimization

4. Triton Inference server

TensorRT Backend + Ensemble



Learn more: Denis Timonin, GTC 2021, Megatron GPT-3 Large Model Inference with Triton and ONNX Runtime

LARGE SCALE NLP DEPLOYMENT

TensorRT and Triton Inference Server

```
name: "megatron_gpt3_18b_onnx_ensemble"
platform: "ensemble"
input [
{
    name: "input_.1"
    data_type: TYPE_INT64
    dims: [4, 1024]
},
{
    name: "input.1"
    data_type: TYPE_INT64
    dims: [4, 1024]
},
{
    name: "2"
    data_type: TYPE_BOOL
    dims: [4, 1,1024,1024]
}
]
output [
{
    name: "9761"
    data_type: TYPE_FP16
    dims: [4, 1024, 50304]
}
]
```

```
ensemble_scheduling {
  step [
    {
      model_name: "megatron_gpt3_18b_onnx_part1"
      model_version: -1
      input_map {
        key: "input_.1"
        value: "input_.1"
      }
      input_map {
        key: "input.1"
        value: "input.1"
      }
      input_map {
        key: "2"
        value: "2"
      }
      output_map {
        key: "4888"
        value: "middle_tensor"
      }
    },
    {
      model_name: "megatron_gpt3_18b_onnx_part2"
      model_version: -1
      input_map {
        key: "4888"
        value: "middle_tensor"
      }
      input_map {
        key: "2"
        value: "2"
      }
      output_map {
        key: "9761"
        value: "9761"
      }
    }
  ]
}
```

Learn more: Denis Timonin, GTC 2021, Megatron GPT-3 Large Model Inference with Triton and ONNX Runtime

LARGE SCALE NLP DEPLOYMENT

TensorRT and Triton Inference Server

```
name: "megatron_gpt3_18b_onnx_part1"
platform: "onnxruntime_onnx"
input [
{
    name: "input_.1"
    data_type: TYPE_INT64
    dims: [4, 1024]
},
{
    name: "input.1"
    data_type: TYPE_INT64
    dims: [4, 1024]
},
{
    name: "2"
    data_type: TYPE_BOOL
    dims: [4, 1,1024,1024]
}
]
output [
{
    name: "4088"
    data_type: TYPE_FP16
    dims: [1024, 4, 4096]
}
]
instance_group[{
    count: 1
    kind: KIND_GPU
    gpus: [0]
}]
```

```
name: "megatron_gpt3_18b_onnx_ensemble"
platform: "ensemble"
input [
{
    name: "input_.1"
    data_type: TYPE_INT64
    dims: [4, 1024]
},
{
    name: "input.1"
    data_type: TYPE_INT64
    dims: [4, 1024]
},
{
    name: "2"
    data_type: TYPE_BOOL
    dims: [4, 1,1024,1024]
}
]
output [
{
    name: "9761"
    data_type: TYPE_FP16
    dims: [4, 1024, 50304]
}
]
```

```
ensemble_scheduling {
  step [
    {
      model_name: "megatron_gpt3_18b_onnx_part1"
      model_version: -1
      input_map {
        key: "input_.1"
        value: "input_.1"
      }
      input_map {
        key: "input.1"
        value: "input.1"
      }
      input_map {
        key: "2"
        value: "2"
      }
      output_map {
        key: "4088"
        value: "middle_tensor"
      }
    },
    {
      model_name: "megatron_gpt3_18b_onnx_part2"
      model_version: -1
      input_map {
        key: "4088"
        value: "middle_tensor"
      }
      input_map {
        key: "2"
        value: "2"
      }
      output_map {
        key: "9761"
        value: "9761"
      }
    }
  ]
}
```

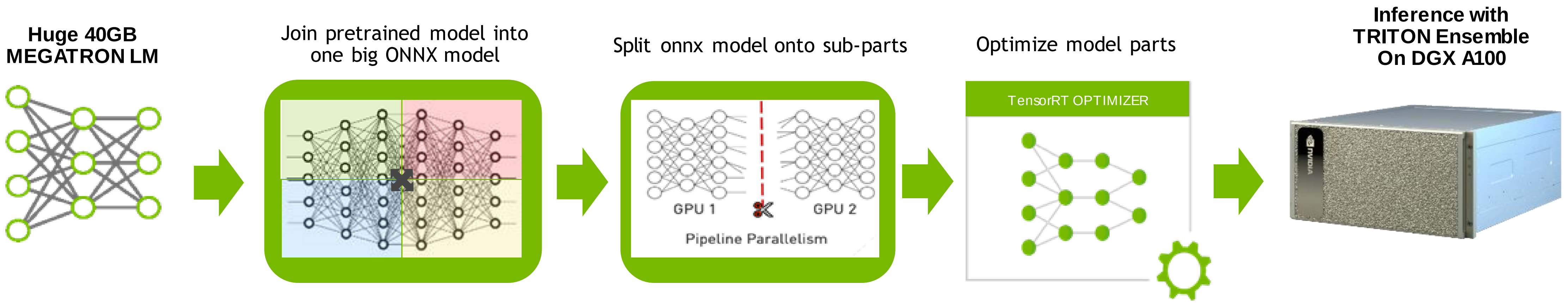
```
name: "megatron_gpt3_18b_onnx_part2"
platform: "onnxruntime_onnx"
input [
{
    name: "4088"
    data_type: TYPE_FP16
    dims: [1024, 4, 4096]
},
{
    name: "2"
    data_type: TYPE_BOOL
    dims: [4, 1,1024,1024]
}
]
output [
{
    name: "9761"
    data_type: TYPE_FP16
    dims: [4, 1024, 50304]
}
]
instance_group[{
    count: 1
    kind: KIND_GPU
    gpus: [1]
}]
```

Learn more: Denis Timonin, GTC 2021, [Megatron GPT-3 Large Model Inference with Triton and ONNX Runtime]

GPT-3 MEGATRON-LM EXAMPLE

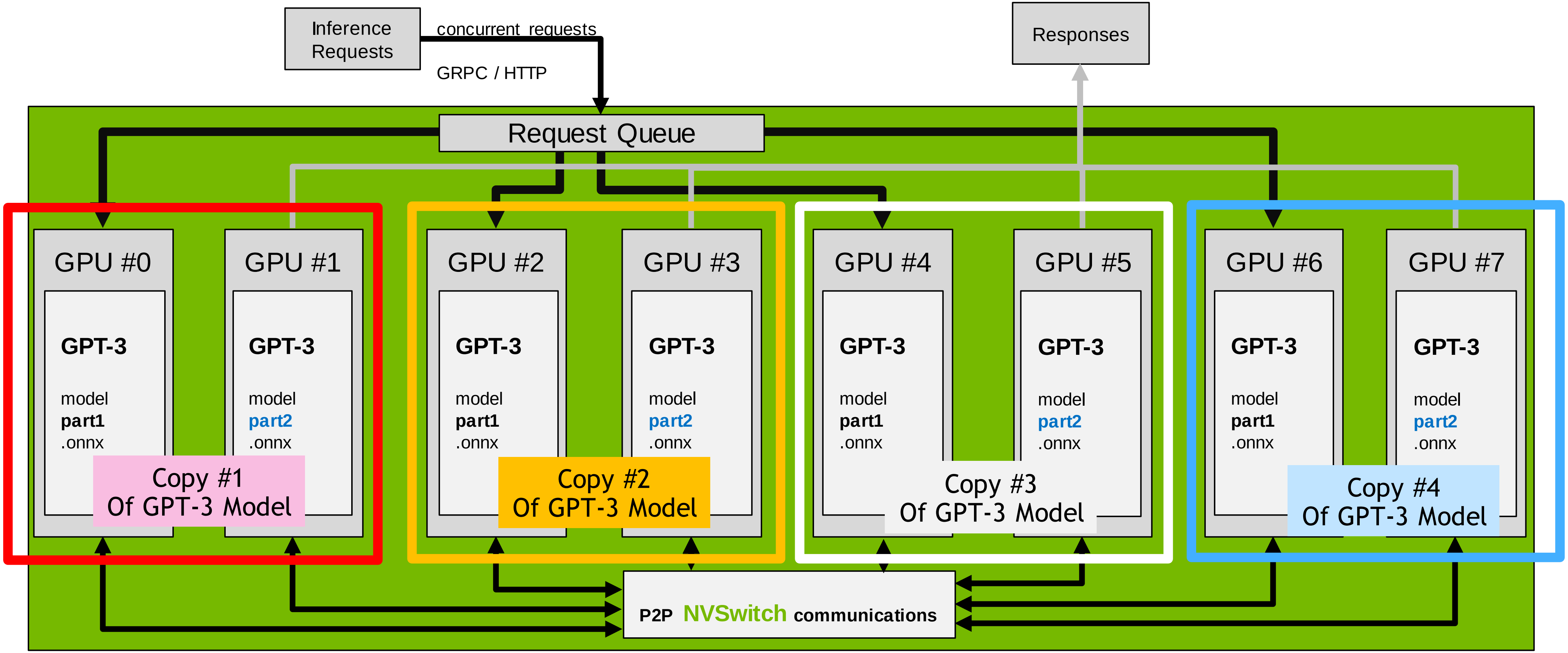
MEGATRON-LM GPT-3

Pipeline-Parallelism Inference steps



SCALING BY ADDING ONE SIMPLE LINE OF CODE

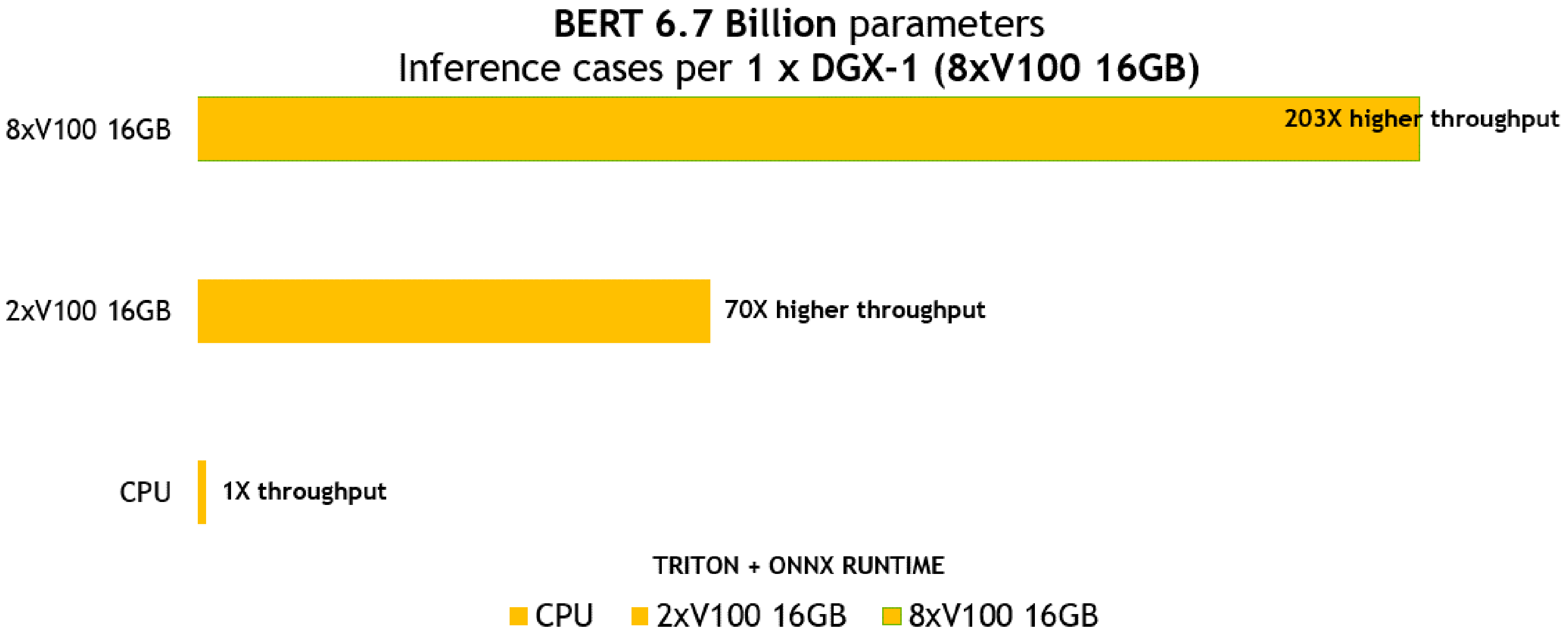
Running 4 Different Inference Jobs on one DGX A100



Inference on DGX A100 with 4 x 2 x GPUs used for our model

INFERENCE RESULTS: MEGATRON-LM ON BERT

203X Higher Throughput on 8x V100 16GB (DGX-1) than CPU

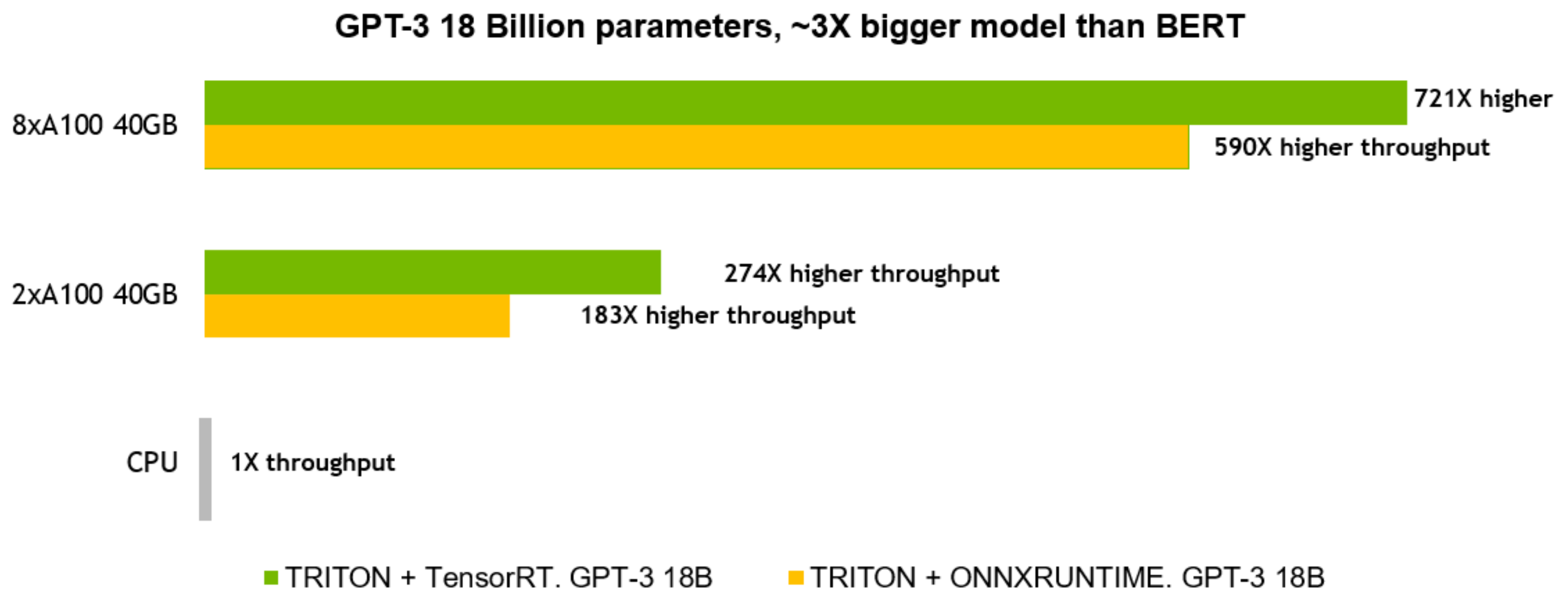


Inference throughput comparisons (requests / per second)
BERT MEGATRON-LM 6.7B parameters. Seq_length=1024.

**CPU case: Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz, FP32. Container: nvcr.io/nvidia/pytorch:21.03-py3. OnnxRuntime-CPU version.*
Code was not properly optimized for this processor, so with better optimization, the difference in results between GPU and CPU may differ multiple times.
GPU: 8xV100 for 4x models in parallel. FP16 Container: TRITON + ONNXRUNTIME nvidia/tritonserver:21.03-py3

INFERENCE RESULTS: MEGATRON-LM ON GPT-3

590X – 720X Higher Throughput on DGX A100 320GB vs. CPU

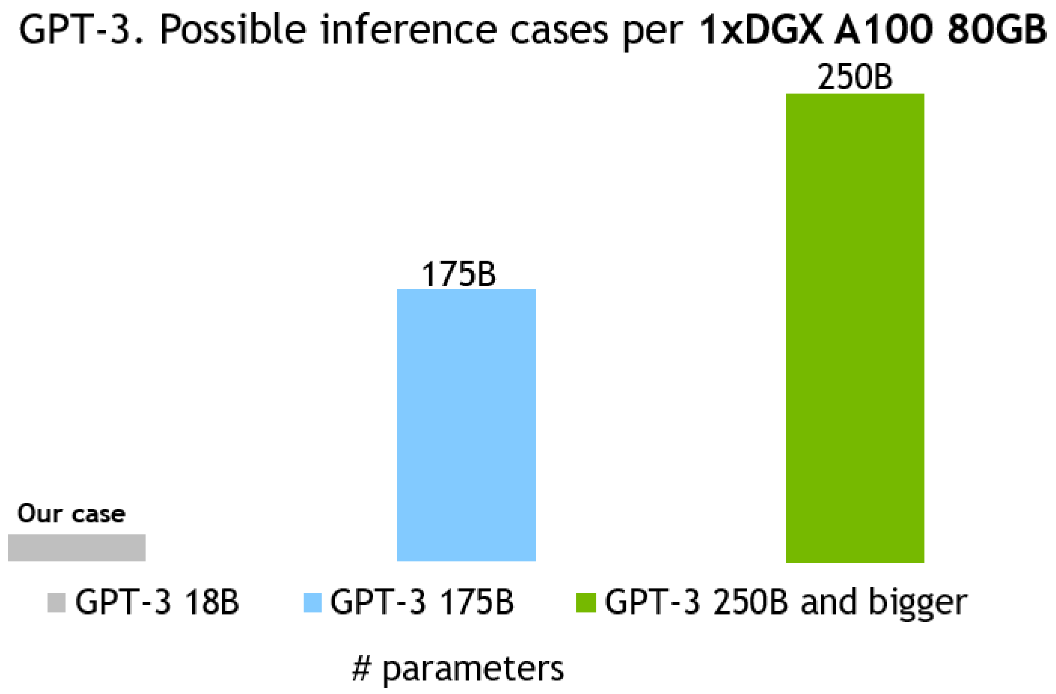


Inference throughput comparisons (requests /per second)
GPT-3 MEGATRON-LM 18B parameters. Seq length=1024.

**CPU case: Dual AMD Rome 7742, 128 cores total, 2.25 GHz (base), 3.4 GHz (max boost), FP32. Container: nvcr.io/nvidia/pytorch:21.03-py3. OnnxRuntime-CPU version.*
Code was not properly optimized for this processor, so with better optimization, the difference in results between GPU and CPU may differ multiple times more.
GPU: 8xA100 for 4x models in parallel. FP16. Container: nvidia/tritonserver:21.03-py3

INFERENCE RESULTS: MEGATRON-LM ON GPT-3

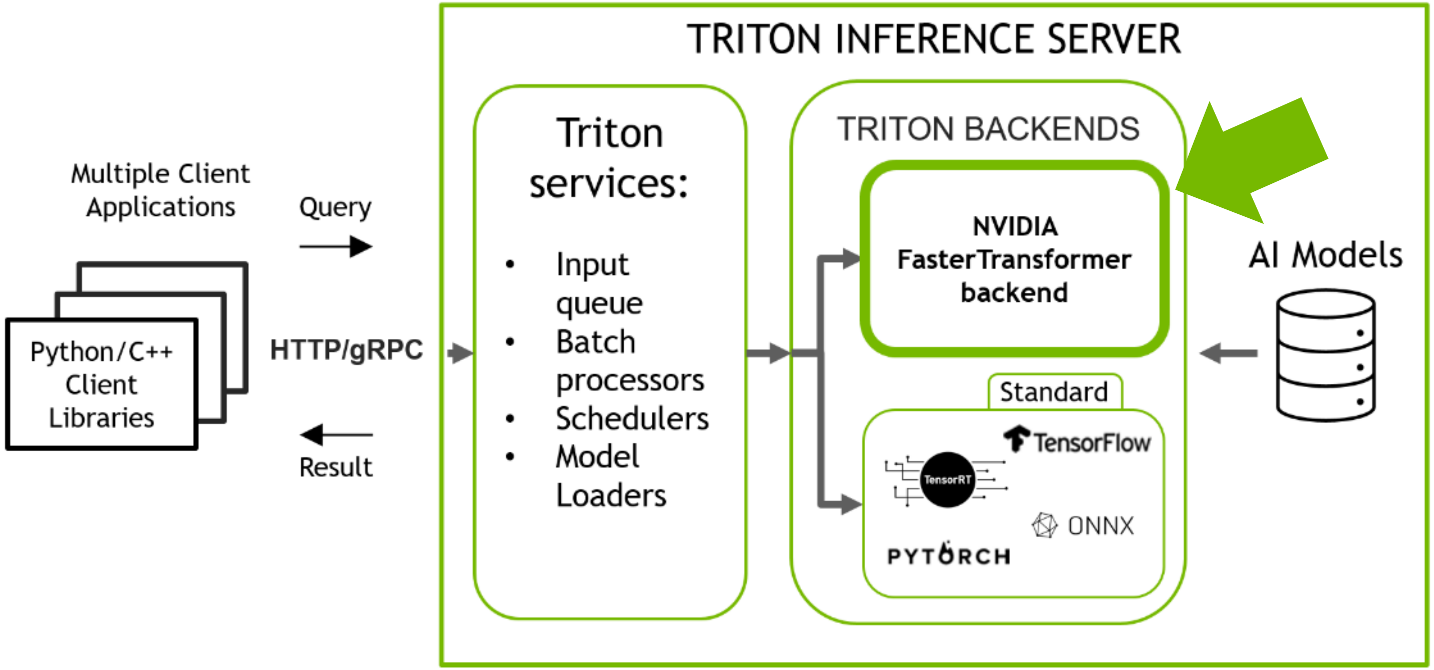
Using our 18B recipe to run Inference on GPT-3 Models 14X parameters
Using 1xDGX A100 640GB



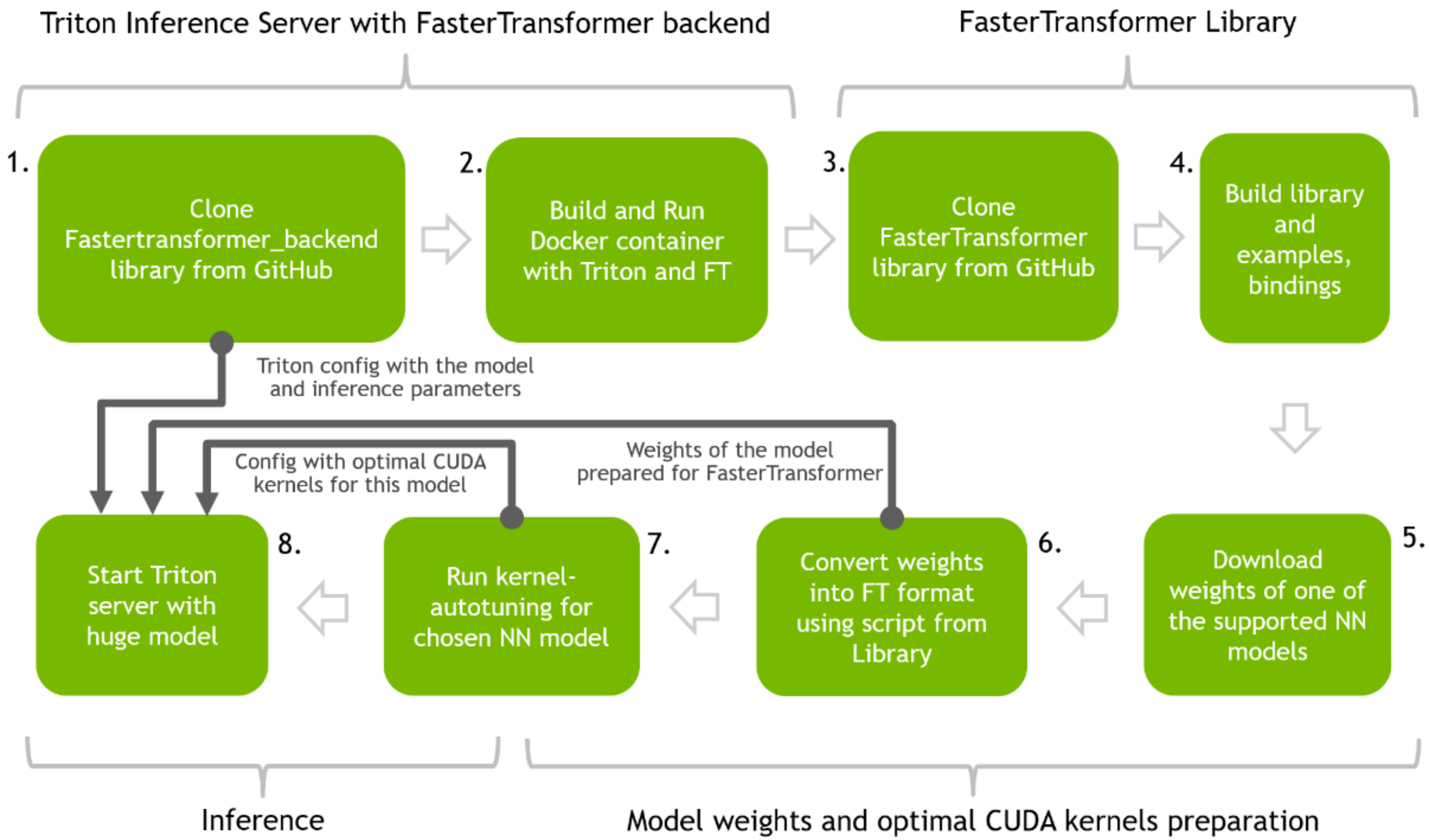
LARGE TRANSFORMERS INFERENCE

FASTER TRANSFORMER + TRITON INFERENCE SERVER

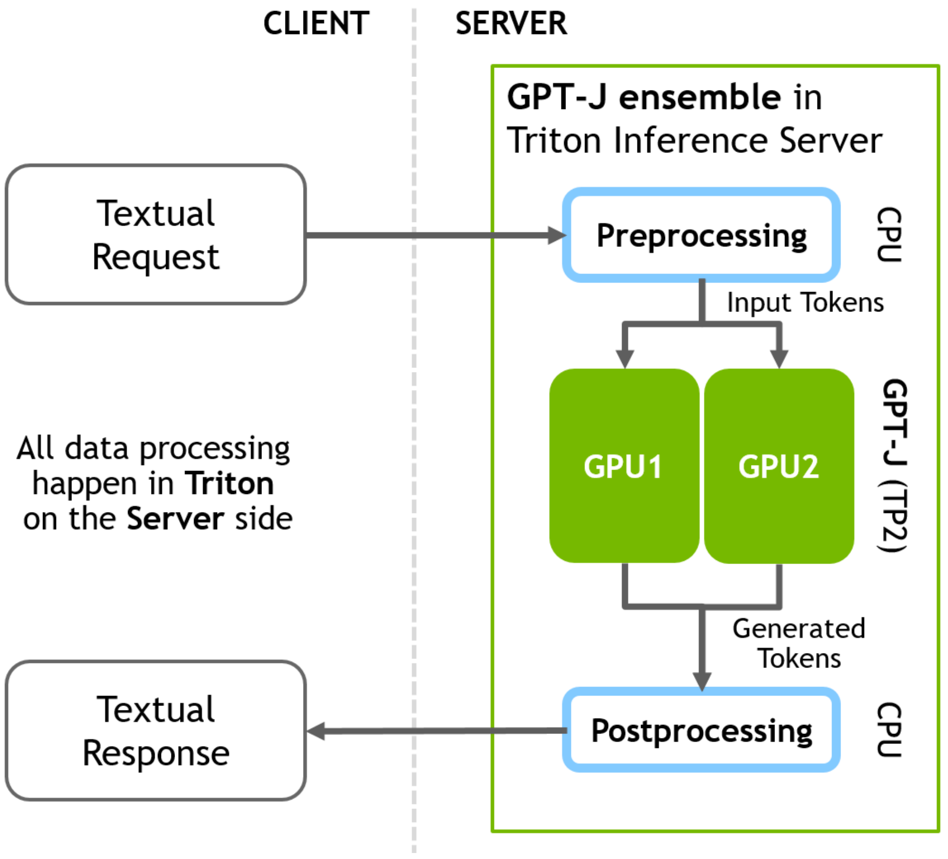
OVERALL ARCHITECTURE



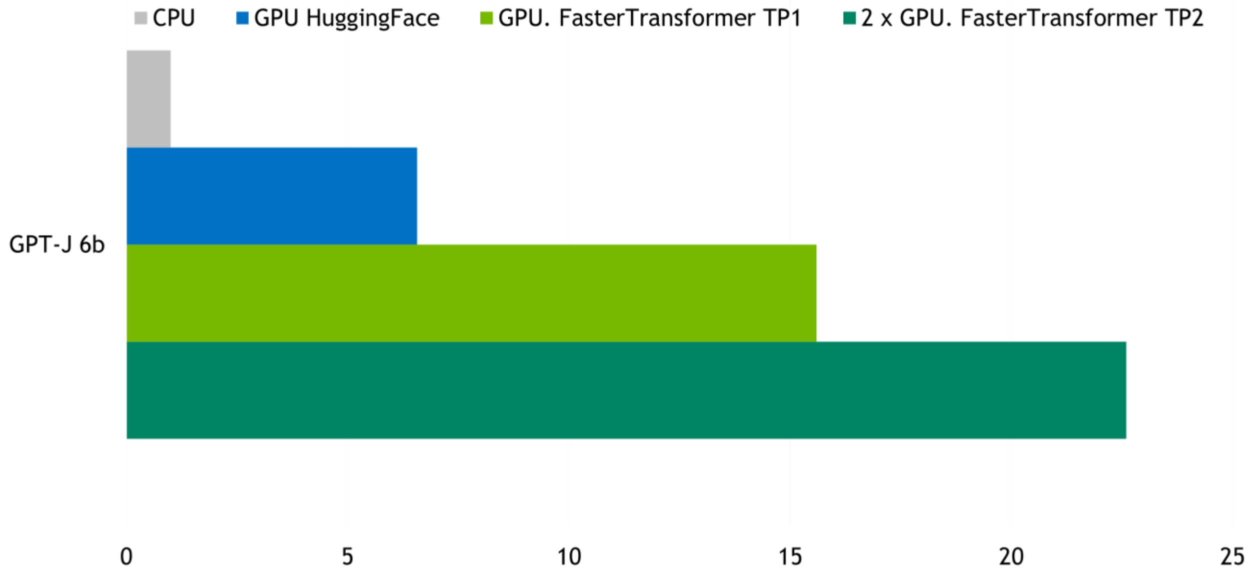
OVERALL ARCHITECTURE



GPT-J TRITON ENSEMBLE



PERFORMANCE



NEMO MERGATRON INFERENCE

NEMO-MEGATRON WITH DGX SUPERPOD

Train what was once impossible

Algorithmic innovation
Train the world's largest transformer-based language models using Megatron's advanced optimizations and parallelization algorithms.

Direct access to world-class NLP experts
Access dedicated expertise from install to infrastructure management to scaling workloads to streamlined production AI.

Optimized Topology for Multi-Node Training
Train the largest models using model parallelism, with NVLINK and InfiniBand for fast cross-node communication.

Turnkey Experience for Rapid Deployment
A full-stack data center platform that includes industry-leading computing, storage, networking, software, and management tools.

Efficiency at Extreme Scale
Training GPT-3 175B takes 355 years on a V100, 14.8 years on 1 DGX A100 and about 1 month on a 140-node DGX SuperPOD

GPT-3 | TRITON + FASTERTRANSFORMER

Value: Multi-Node Inference for large scale Transformer Models

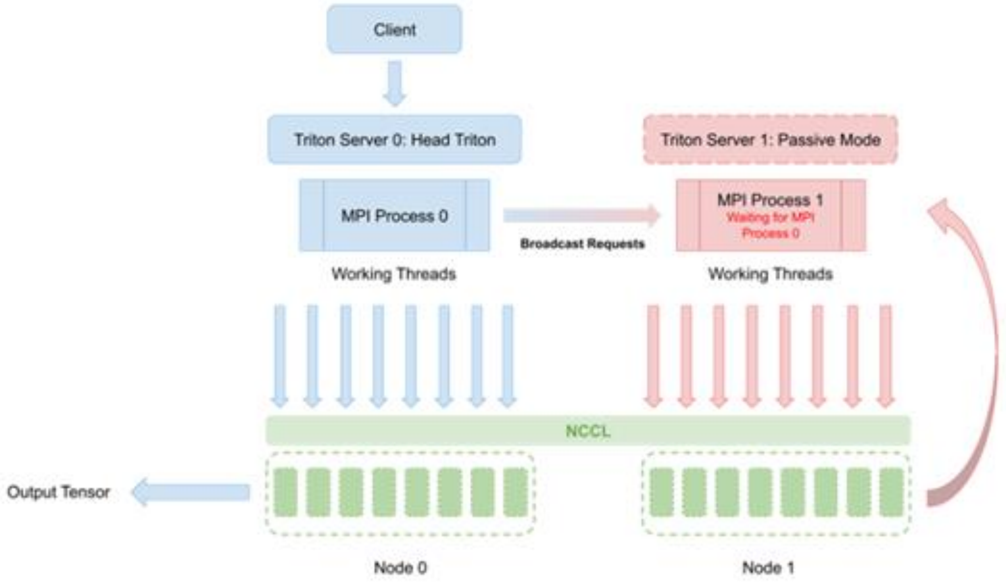
Goal: Serve giant transformer models and accelerate inference performance

Capabilities:

- Written in C++/CUDA and relies on cuBLAS, cuBLASlt, cuSPARSELt
- Optimize kernels for encoder/decoder layers of transformer models
- Integrated as a backend in Triton Inference Server
- Uses tensor/pipeline parallelism for multi-GPU, multi-node inference
- FP16, FP32 supported
- POC of Post-training weight-only INT8 quantization for GPT
  - Only for BS 1-2
- Megatron and HuggingFace converters provided
  - POC of Tensorflow/ONNX converters
- Uses MPI and NCCL to enable inter/intra node communication

Exceptions/Limitations:

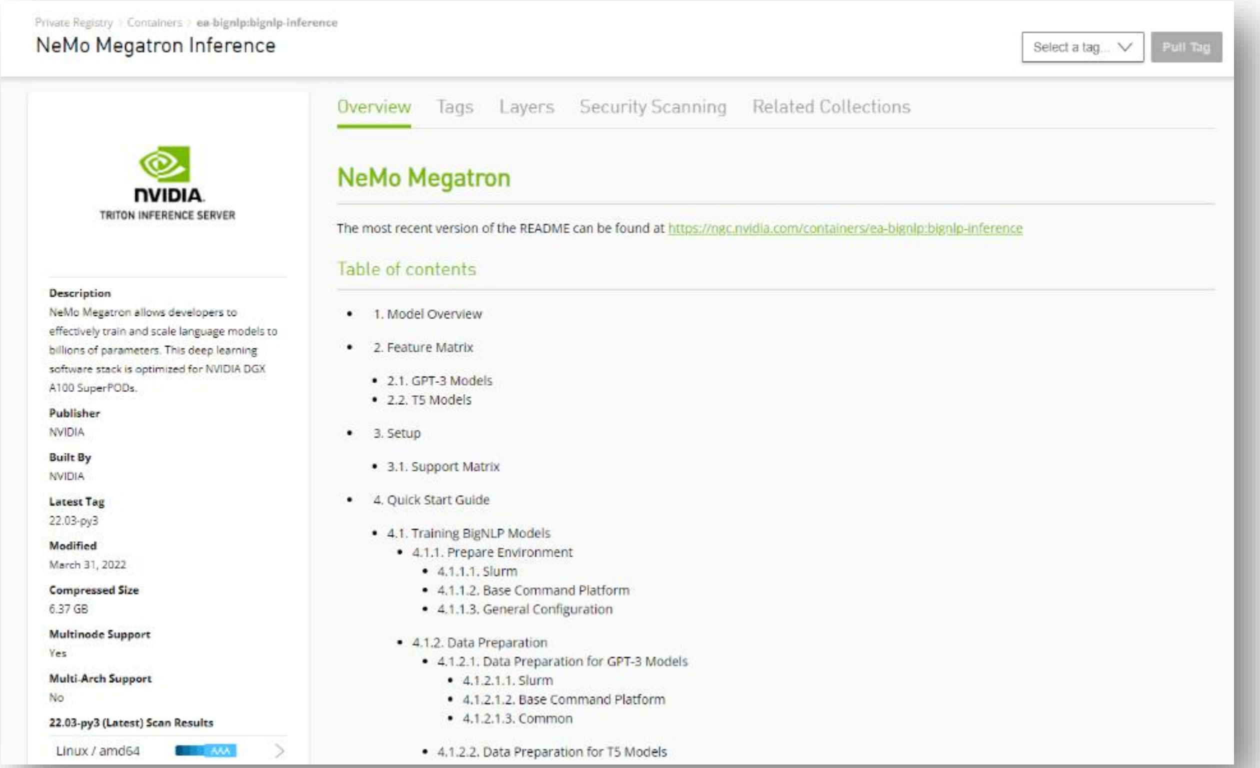
- Supports only GPT and T5 style models currently
- Size-per-head (of the attention head) of the model must be 32, 64, 96, 128, 144, 160, 192, 224 and 256
- Model must be converted to FasterTransformer format
- Currently beta release

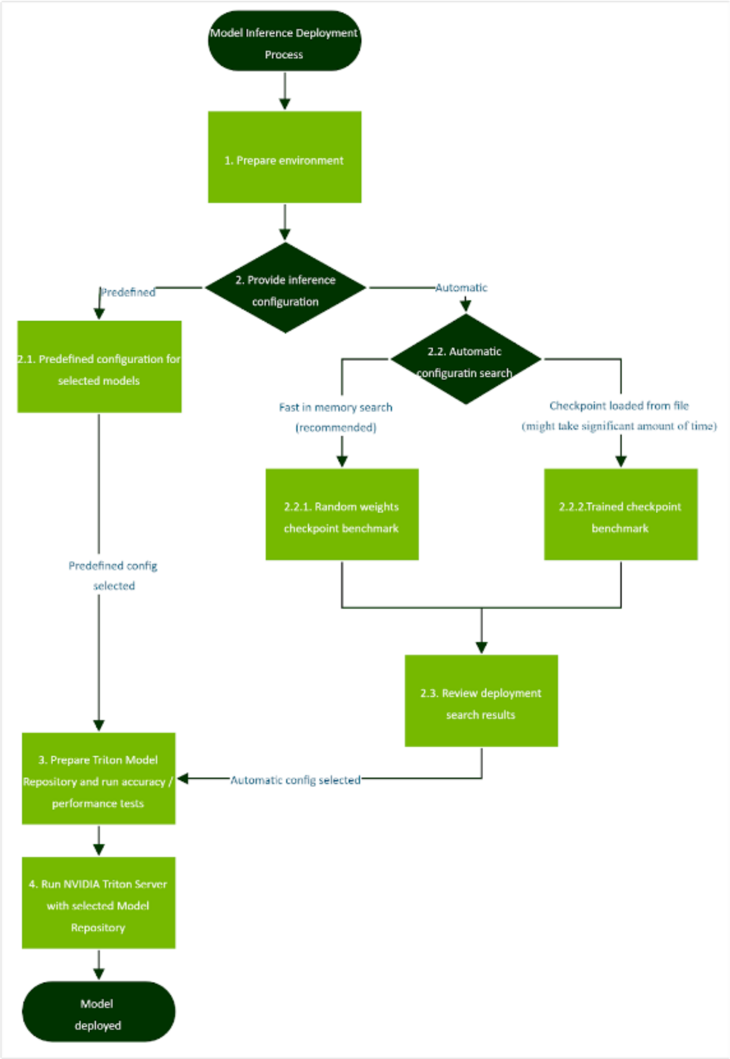


https://developer.nvidia.com/nemo-megatron-early-access

GPT-3 | TRITON + FASTERTRANSFORMER

Value: Multi-Node Inference for large scale Transformer Models

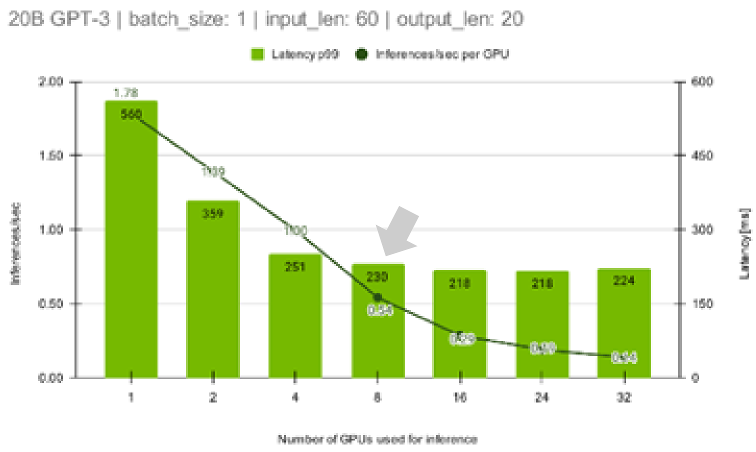


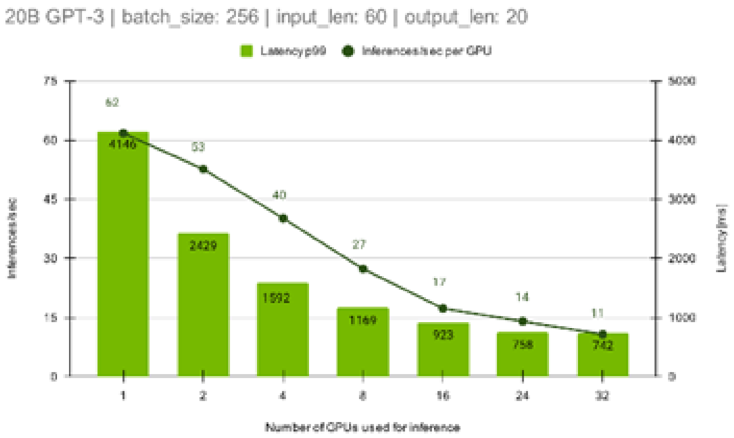


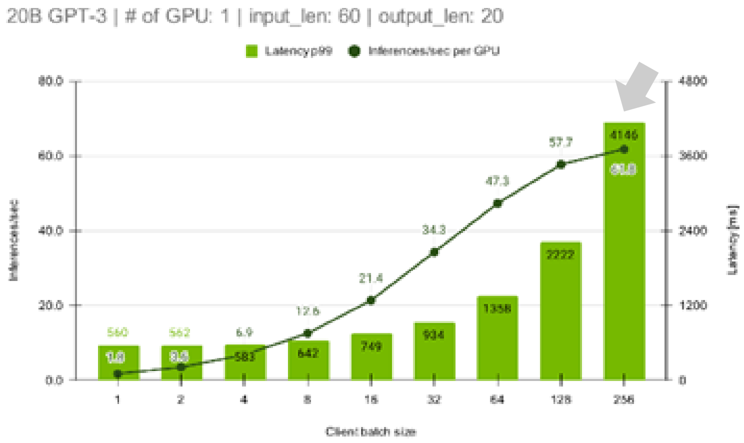
https://developer.nvidia.com/nemo-megatron-early-access

INFERENCE BENCHMARKS

Model Size	Input Length	Output Length	Batch Size	# of GPUs	Min of P99 (ms)	Max of P99 (ms)
1.3B	60	20	1 - 256	1 – 8	74	437
5.1B	60	20	1 - 256	1 – 8	94	1,143
20B	60	20	1 - 256	1 – 8	230	4,146
175B	60	20	1 - 256	8 – 32	649	5,731
530B	60	20	1 - 256	16 - 32	1,054	8,326

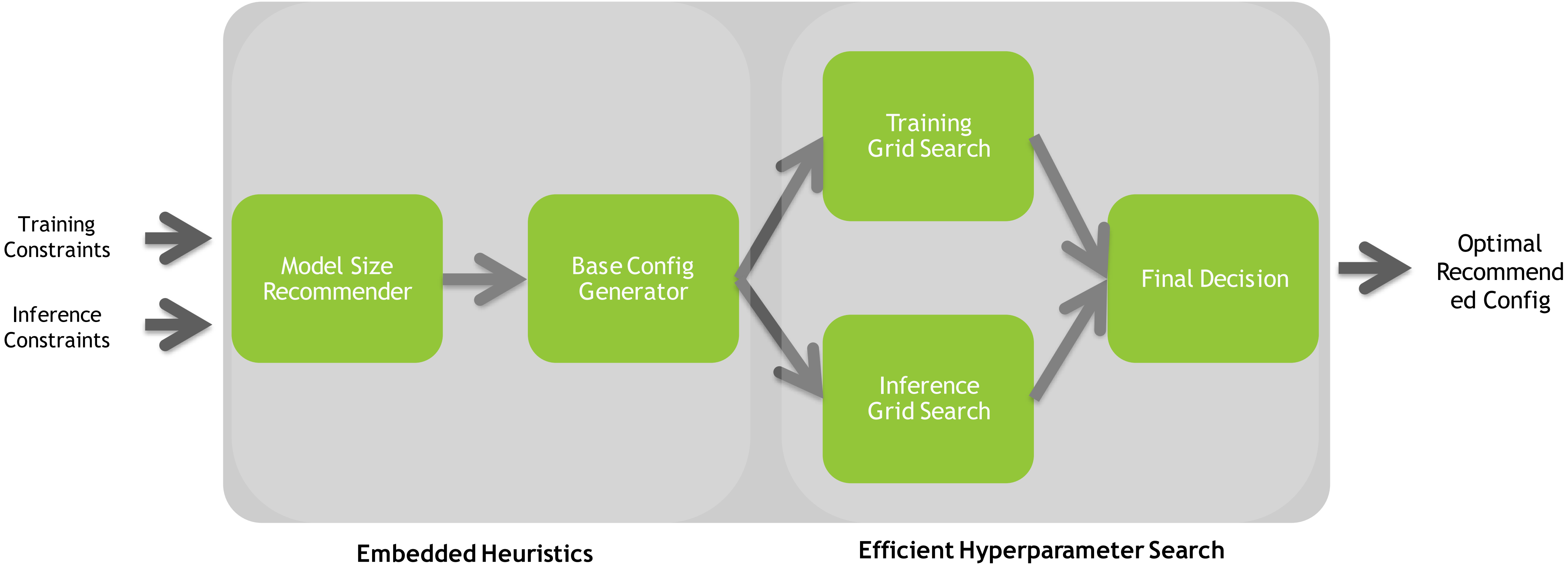






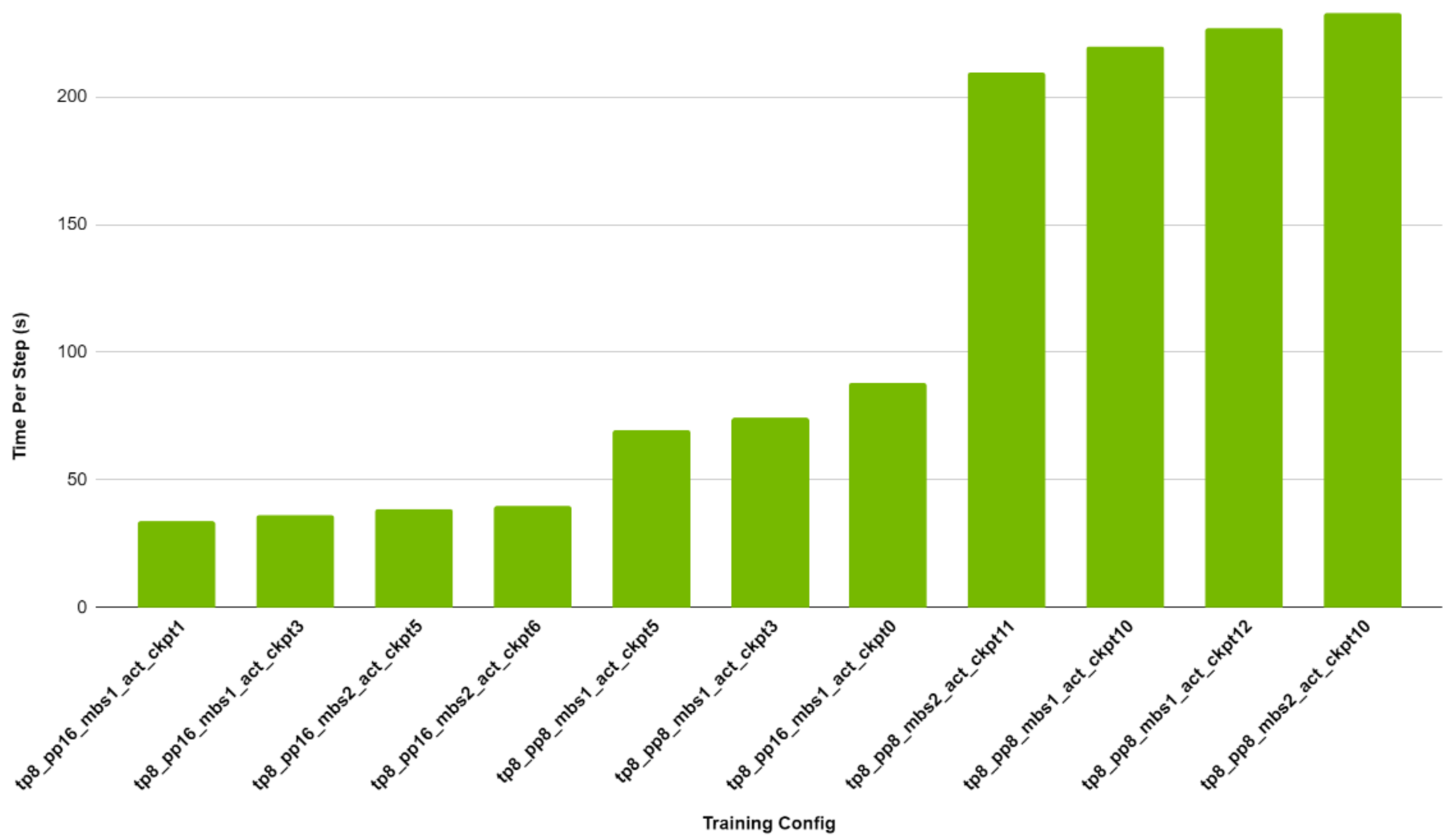
OVERVIEW OF THE TOOLING

Efficient Hyperparameter Search With Embedded Heuristics



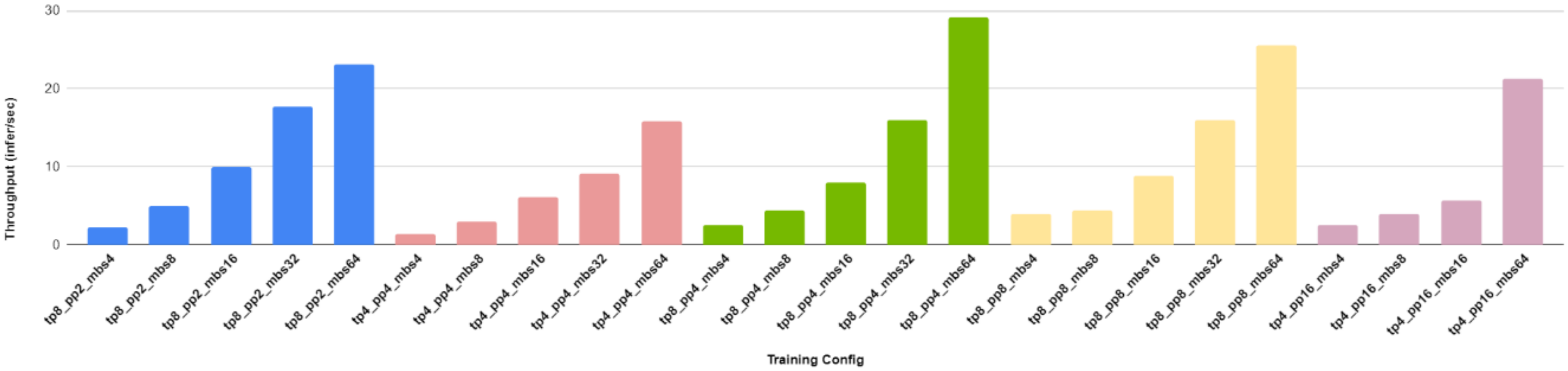
PERFORMANCE GAINS

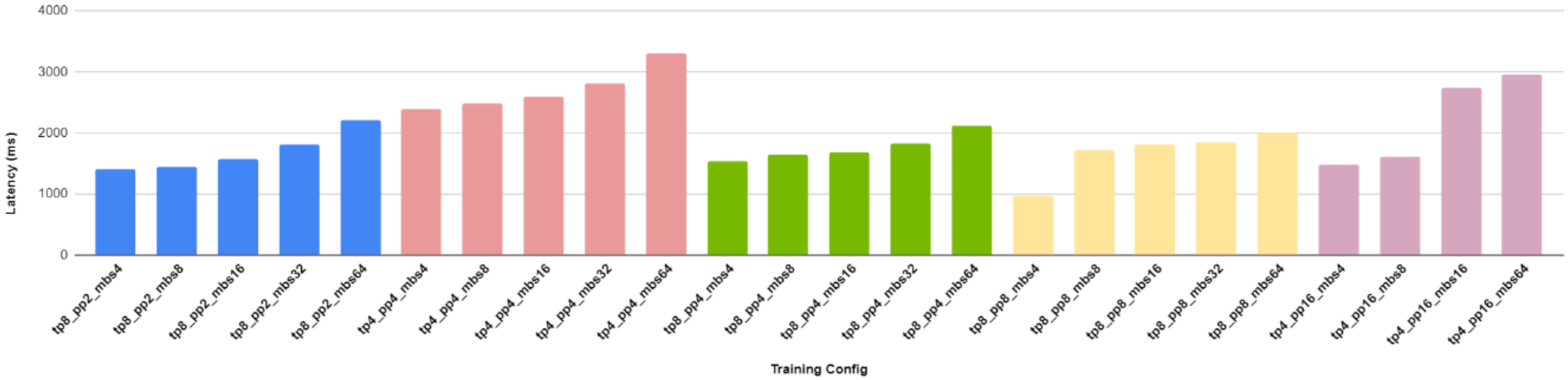
175B GPT-3 Model: 6.85x training speedup



PERFORMANCE GAINS

Inference 175B GPT-3 Model: Optimizing throughput and latency





Each color shows a model config, with different MBS values

LAB OVERVIEW

LAB

Overview

- Baseline inference Of GPT-J with 6B parameters using the Hugging Face library and PyTorch
- Inference Of GPT-J with Faster Transformers
- Distributed inference: Tensor Parallel (TP) and/or Pipeline Parallel (PP)
- Deployment of GPT-J with Triton Inference Server