Part 3: Production Deployment

Building Conversational AI Applications

Course Timeline

8:50	—	9:00	Welcome
9:00	—	10:00	Lecture 1
10:00	—	11:00	Lab 1
11:00	—	11:15	Break
11:15	—	12:15	Lecture 2
12:15	—	13:15	Lab 2
13:15	—	14:00	break
14:00	—	15:00	Lecture 3
15:00	—	16:00	Lab 3
16:00	—	17:00	Assessment support

Course Agenda

Part 1: ASR - Automatic Speech Recognition

Lecture: Challenges of conversational AI applications, building blocks, ASR (automatic speech recognition) focus

Lab: Investigate ASR pipeline with NeMo and Riva

Part 2: TTS – Text To Speech and Conversational AI Customization

Lecture: How can we take advantage of conversational AI platforms to build custom applications?

Lab: Investigate the TTS pipeline with NeMo and Riva; then build a full custom conversational AI pipeline.

Part 3: Production Deployment

Lecture: Discussion of production deployment considerations including an overview of NVIDIA Triton Inference Server and TensorRT.

Lab: Hands-on deployment of an example application using Helm and Kubernetes; includes profiling the application

Deploy (Part 3)

Lecture

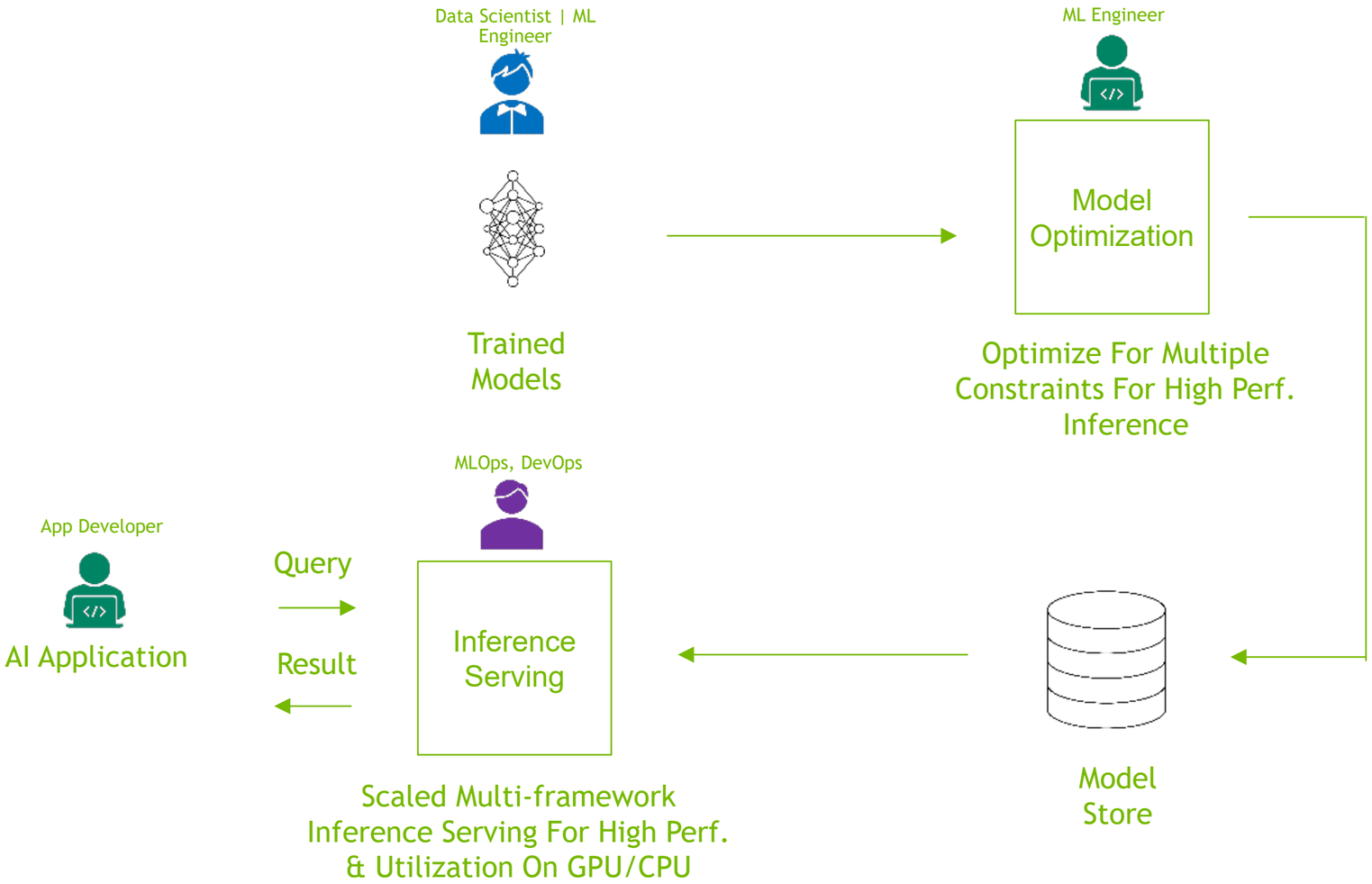
- Beyond Model Accuracy, Meeting Application Requirements
- The Right Model
- The Right Implementation
- Model Optimization Techniques
- Beyond a Single Model
- Real-World Applications

Lab

- Making Kubernetes GPU-Aware
- First GPU-Accelerated Application
- Monitoring Your GPUs
- Deploying Riva
- Using Riva

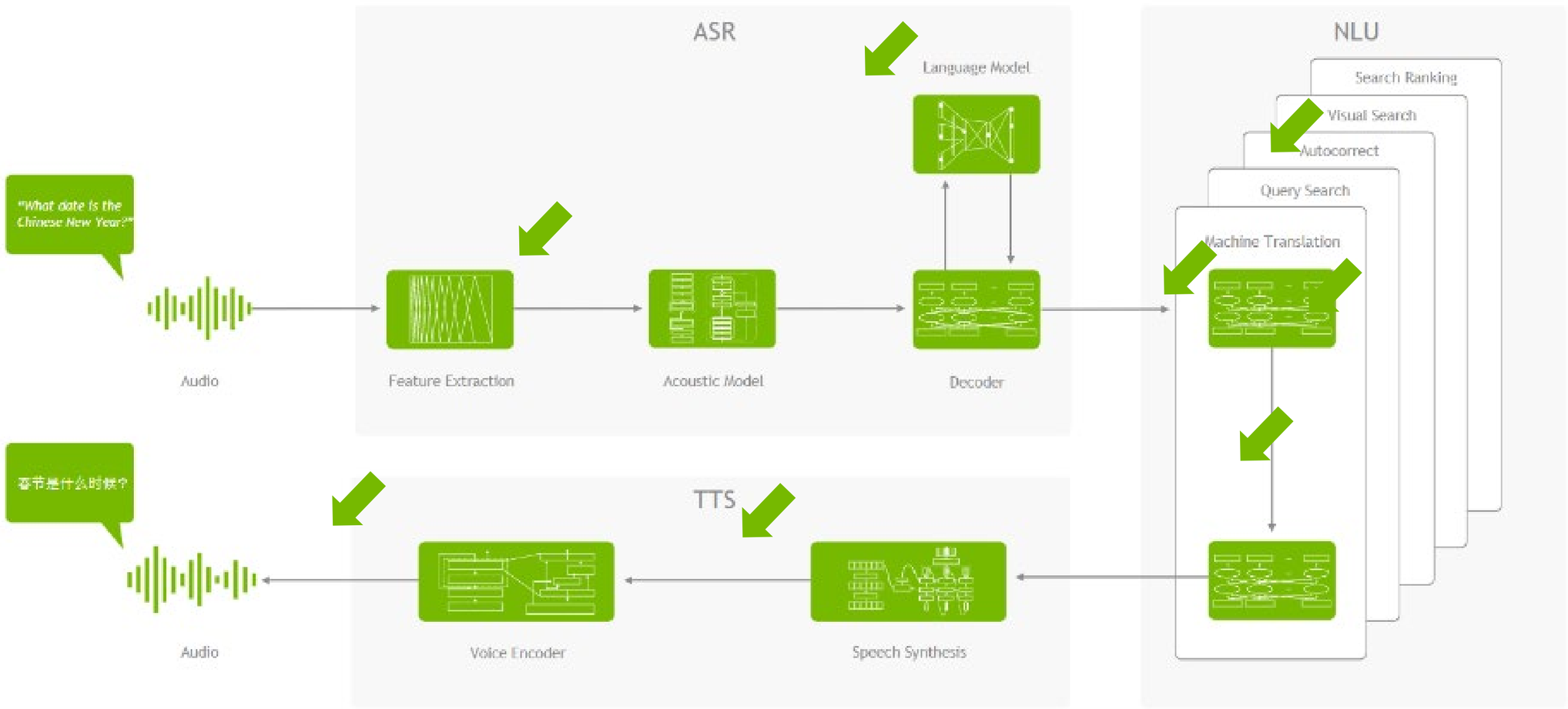
AI Deep Learning Inference Workflow

Two-part process implemented by multiple personas



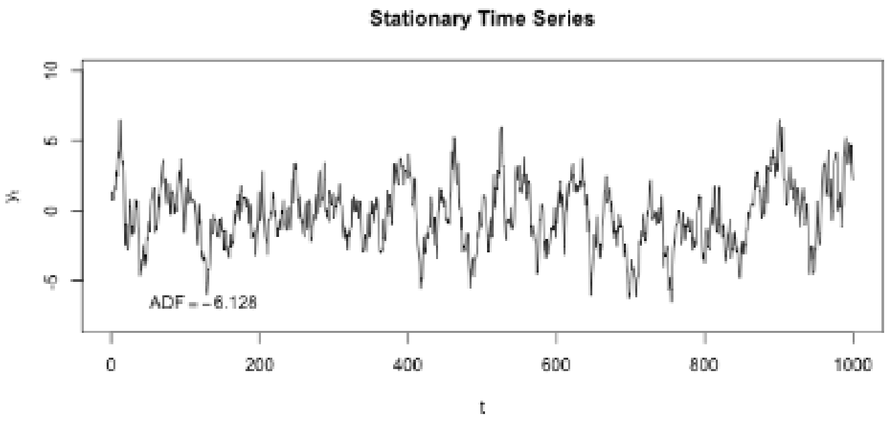
Beyond Model Accuracy

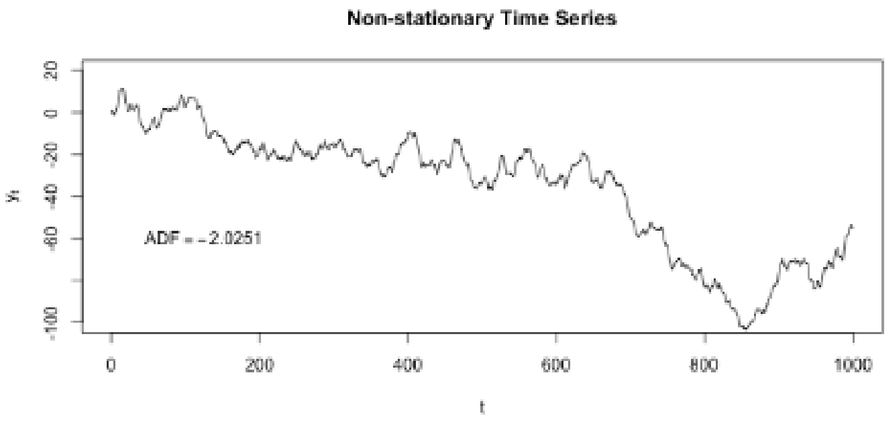
They do not live in isolation



Beyond Model Accuracy

And they need to evolve over time





https://en.wikipedia.org/wiki/Stationary_process

Beyond Model Accuracy

Complexity of real-world applications

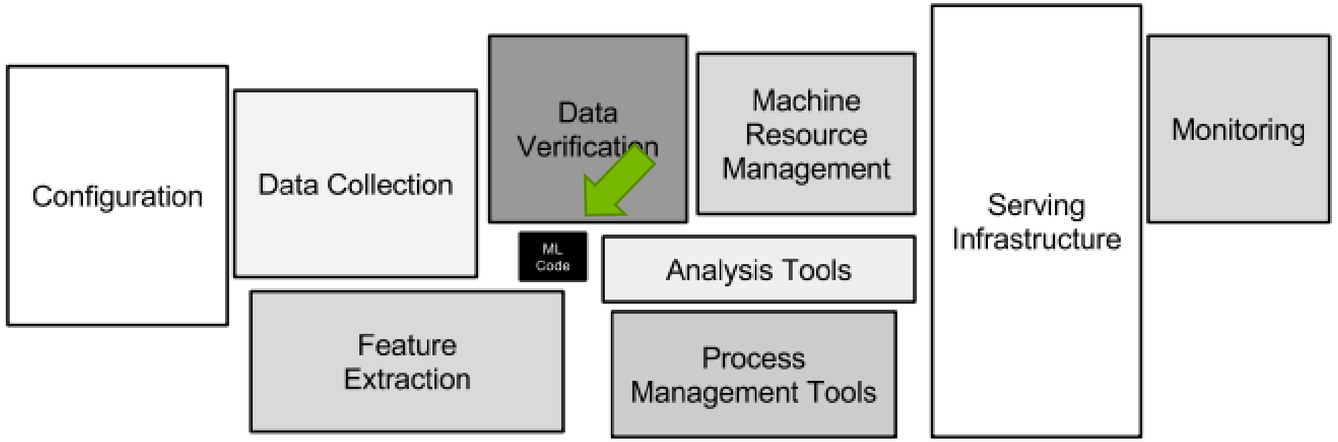


Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., ... & Dennison, D. (2015). Hidden technical debt in machine learning systems. In Advances in neural information processing systems (pp. 2503-2511).

Beyond Model Accuracy

The trade-off | accuracy - latency – throughput – power efficiency

Server Scenario - Closed Division

Network	Throughput	Target Accuracy	MLPerf Server Latency Constraints (ms)	GPU	Server	Dataset	GPU Version
ResNet-50 v1.5	87,984 queries/sec	76.46% Top1	15	8x A10	Supermicro 4029GP-TRT-OTO-28	ImageNet	A10
	115,987 queries/sec	76.46% Top1	15	8x A30	Gigabyte G482-Z54	ImageNet	A30
	3,602 queries/sec	76.46% Top1	15	1x1g.10gb A100	DGX A100	ImageNet	A100 SXM-80GB
	30,794 queries/sec	76.46% Top1	15	1x A100	DGX A100	ImageNet	A100 SXM-80GB
	259,994 queries/sec	76.46% Top1	15	8x A100	DGX A100	ImageNet	A100 SXM-80GB
	207,976 queries/sec	76.46% Top1	15	8x A100	Gigabyte G482-Z54	ImageNet	A100-PCIe-40GB
	106,988 queries/sec	76.46% Top1	15	4x A100	DGX-Station-A100	ImageNet	A100 SXM-80GB

Power Efficiency Offline Scenario - Closed Division

Network	Throughput	Throughput per Watt	GPU	Server	Dataset	GPU Version
ResNet-50 v1.5	213,599 samples/sec	97.31 samples/sec/watt	8x A100	Gigabyte G482-Z54	ImageNet	A100 PCIe-40GB
	270,706 samples/sec	78.27 samples/sec/watt	8x A100	DGX A100	ImageNet	A100 SXM-80GB
	124,529 samples/sec	98.14 samples/sec/watt	4x A100	DGX-Station-A100	ImageNet	A100 SXM-80GB

https://developer.nvidia.com/deep-learning-performance-training-inference

Deploy (Part 3)

Lecture

- Beyond Model Accuracy, Meeting Application Requirements
- The Right Model
- The Right Implementation
- Model Optimization Techniques
- Beyond a Single Model
- Real-World Applications

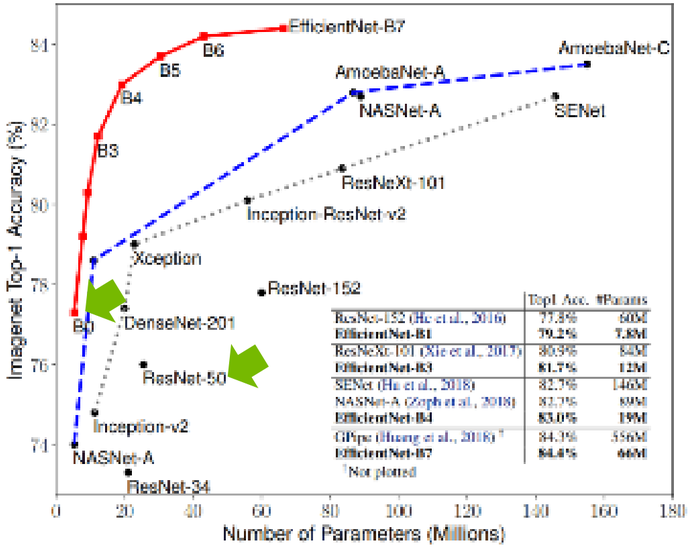
Lab

- Making Kubernetes GPU-Aware
- First GPU-Accelerated Application
- Monitoring Your GPUs
- Deploying Riva
- Using Riva

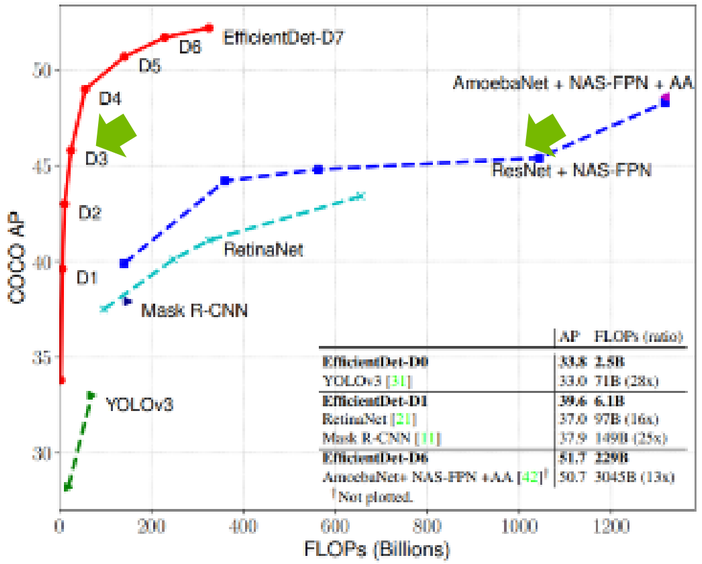
Model Selection

Not all models are created equal

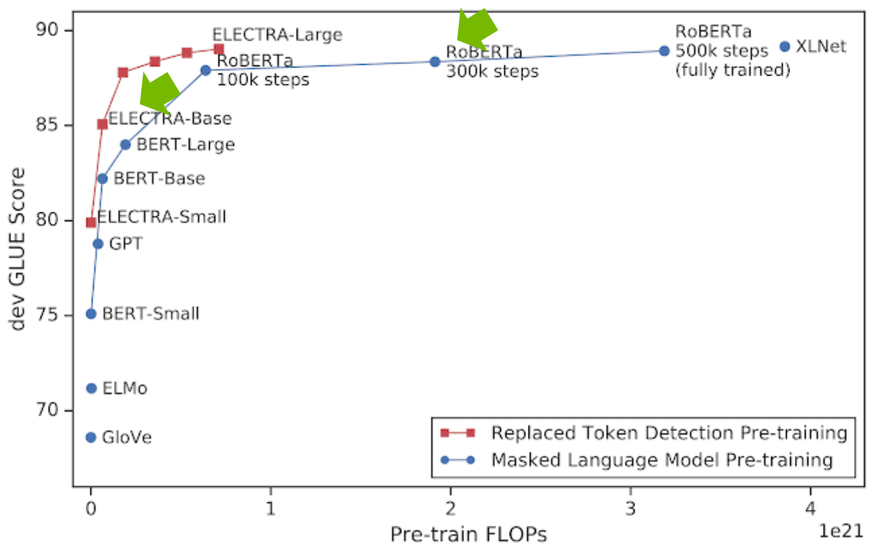
Image Classification



Object detection



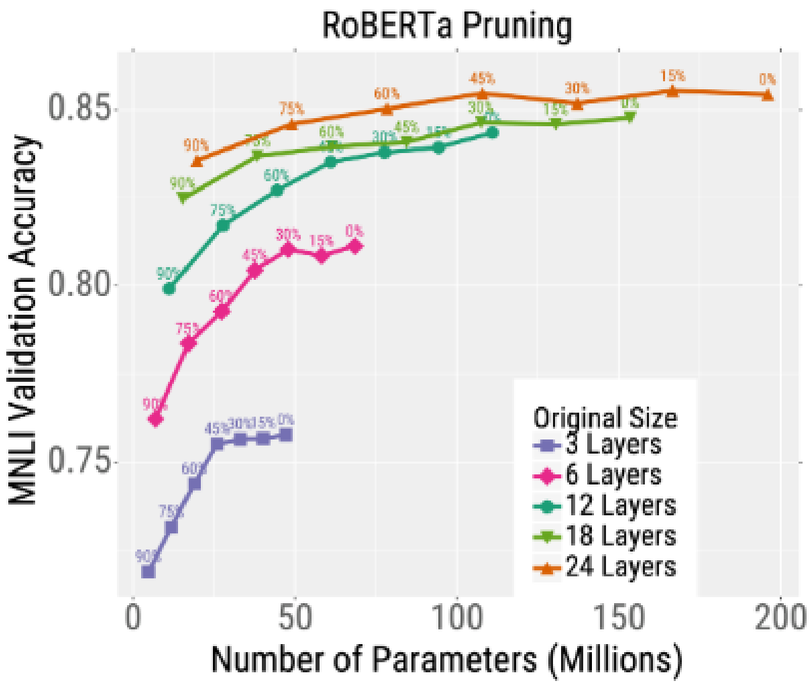
NLP

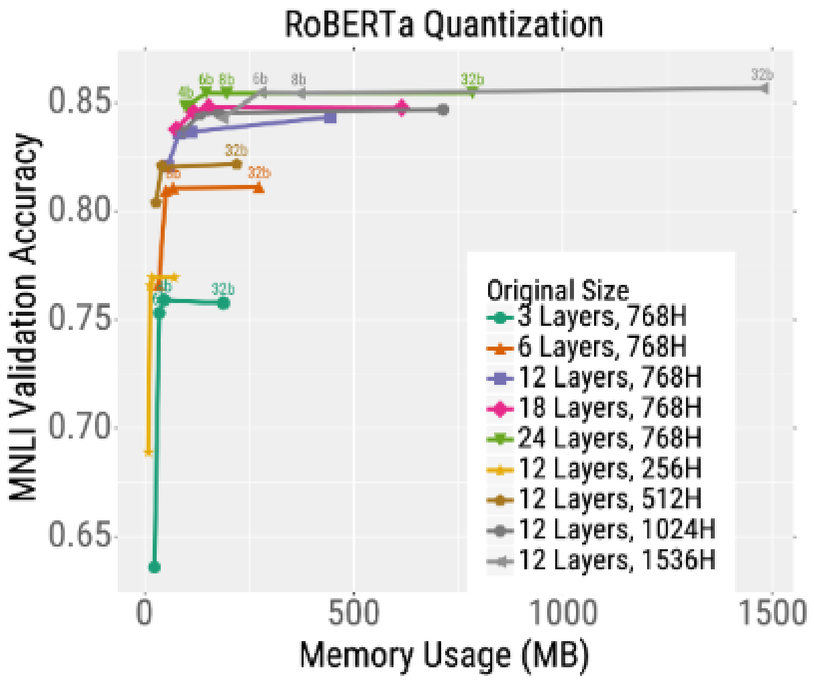


Kevin Clark and Minh-Thang Luong and Quoc V. Le and Christopher D. Manning. ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. ICLR 2020
Mingxing Tan and Quoc V. Le "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". ICML 2019

Model Selection

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



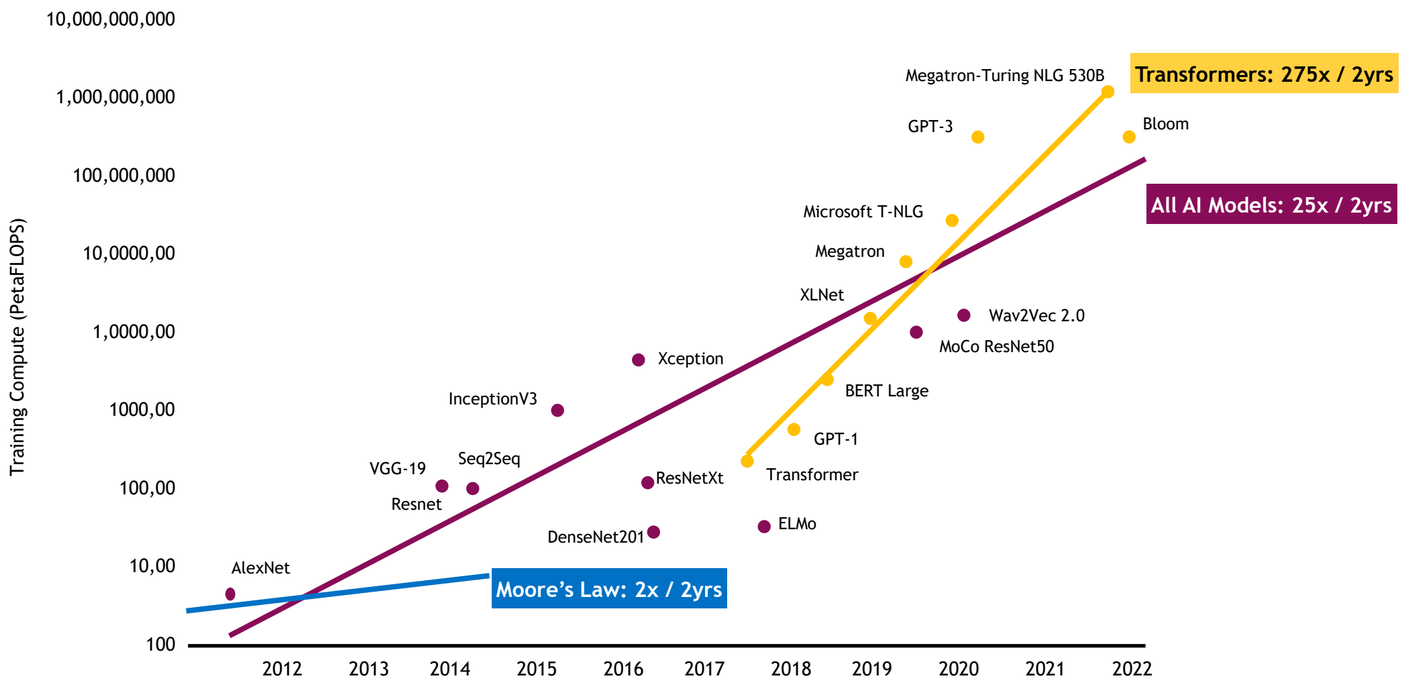


https://bair.berkeley.edu/blog/2020/03/05/compress/

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.

Model Selection

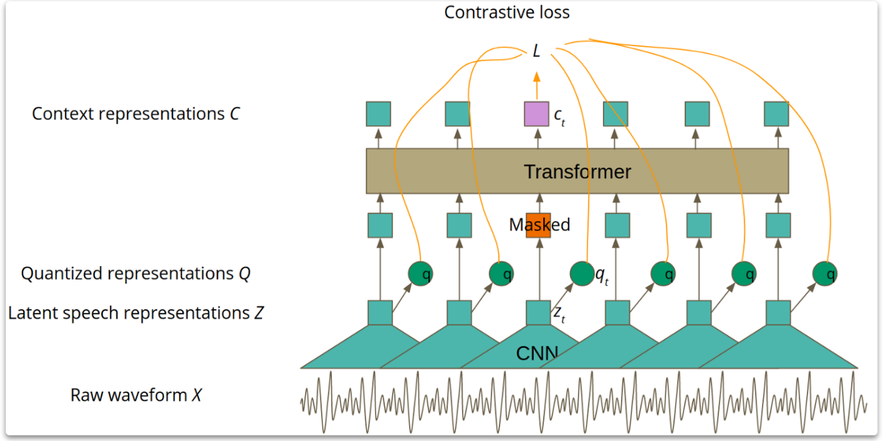
Very large models are and will continue to be prevalent



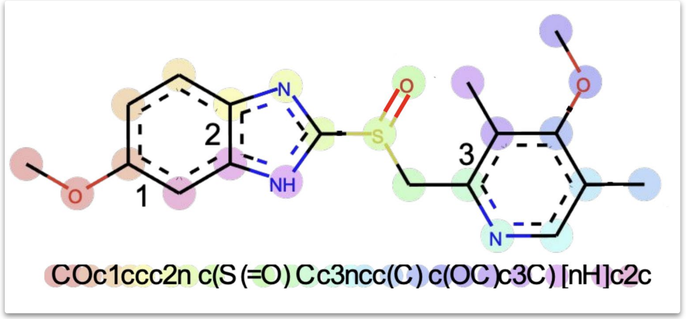
https://developer.nvidia.com/blog/using-deepspeed-and-megatron-to-train-megatron-turing-nlg-530b-the-worlds-largest-and-most-powerful-generative-language-model

Scaling

Beyond Language



wav2vec 2.0



MegaMolBart

Imagen

Deploy (Part 3)

Lecture

- Beyond Model Accuracy, Meeting Application Requirements
- The Right Model
- The Right Implementation
- Model Optimization Techniques
- Beyond a Single Model
- Real-World Applications

Lab

- Making Kubernetes GPU-Aware
- First GPU-Accelerated Application
- Monitoring Your GPUs
- Deploying Riva
- Using Riva

Code Quality Is Key

Dramatic differences in model performance

3-layer BERT with 128 sequence length

		Batch size	Inference on	Throughput (Query per second)	Latency (milliseconds)
CPU	Original 3-layer BERT	1	Azure Standard F16s_v2 (CPU)	6	157
	ONNX Model	1	Azure Standard F16s_v2 (CPU) **with ONNX Runtime**	111	9
GPU	Original 3-layer BERT	4	Azure NV6 GPU VM	200	20
	ONNX Model	4	Azure NV6 GPU VM **with ONNX Runtime**	500	8
	ONNX Model	64	Azure NC6S_v3 GPU VM **with ONNX Runtime + System Optimization** (Tensor Core with mixed precision, Same Accuracy)	10667	6

https://cloudblogs.microsoft.com/opensource/2020/01/21/microsoft-onnx-open-source-optimizations-transformer-inference-gpu-cpu

Deploy (Part 3)

Lecture

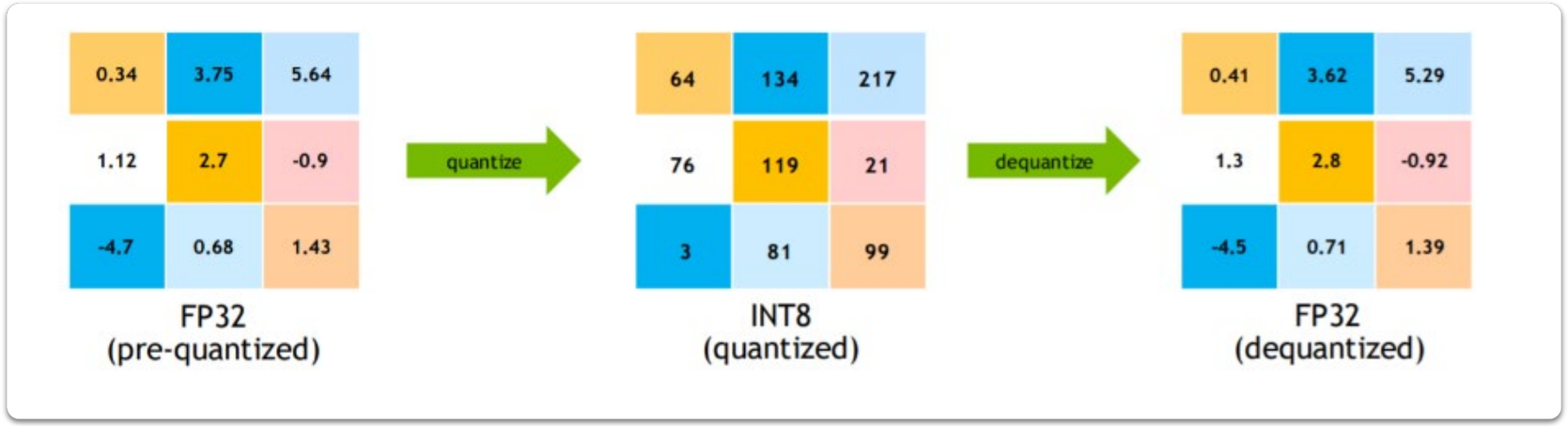
- Beyond Model Accuracy, Meeting Application Requirements
- The Right Model
- The Right Implementation
- Model Optimization Techniques
- Beyond a Single Model
- Real-World Applications

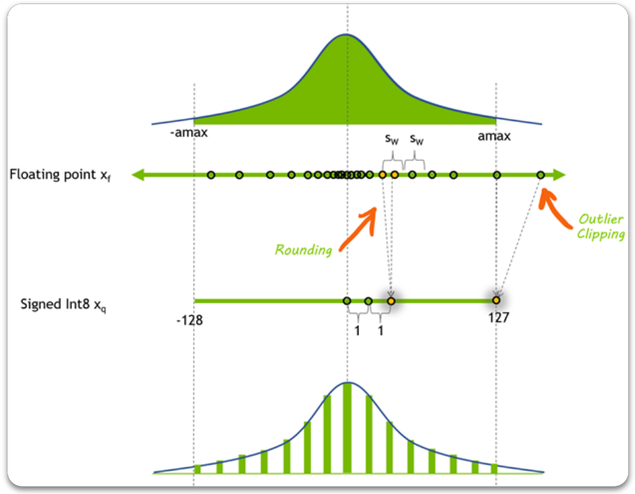
Lab

- Making Kubernetes GPU-Aware
- First GPU-Accelerated Application
- Monitoring Your GPUs
- Deploying Riva
- Using Riva

Quantization

The idea





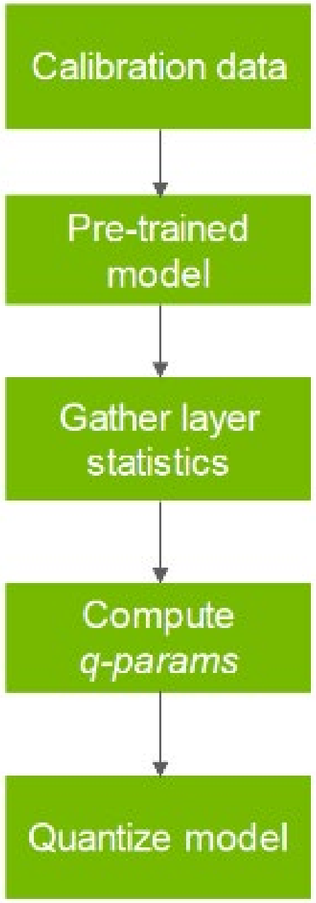
8-bit signed integer quantization of a floating-point tensor x_f. The symmetric dynamic range of $[-amax, amax]$ is mapped through quantization to [-128, 127].

https://developer.nvidia.com/blog/improving-int8-accuracy-using-quantization-aware-training-and-the-tao-toolkit/

Quantization

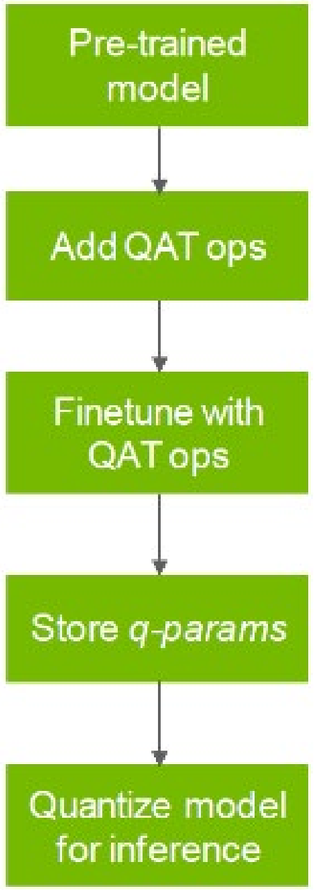
Approaches

Post-training quantization(PTQ)



Post-training quantization (PTQ)		Quantization-aware training (QAT)
Start with a pretrained model and evaluate it on a calibration dataset.		Start with a pretrained model and introduce quantization ops at various layers
Calibration data is used to calibrate the model. It can be a subset of training data.		Fine-tune 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 pretrained model.

Quantization-aware training (QAT)



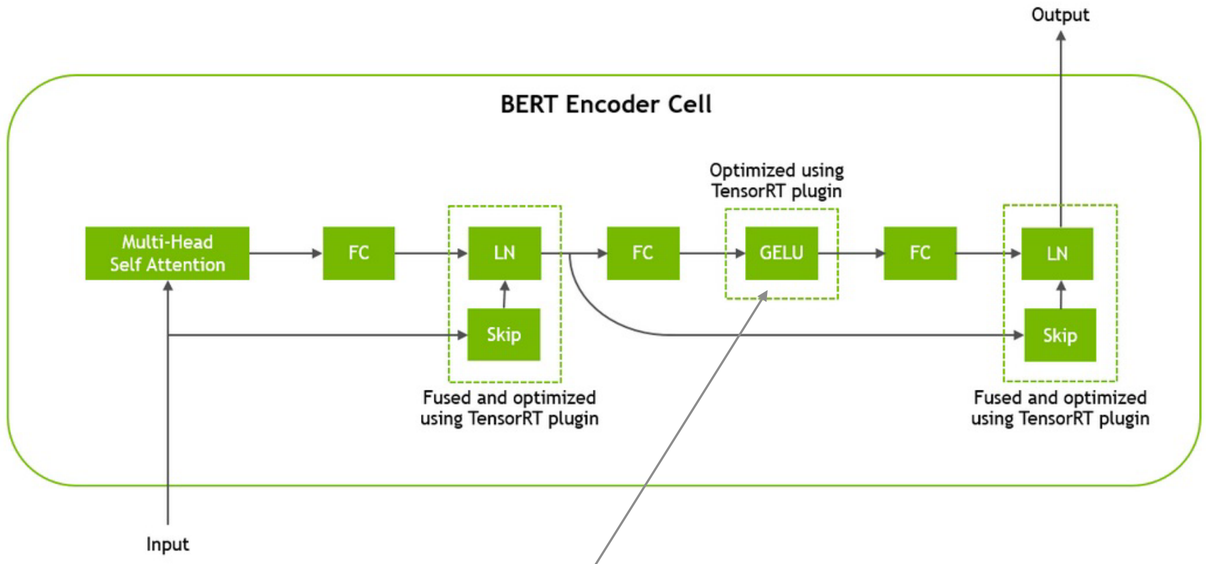
Quantization

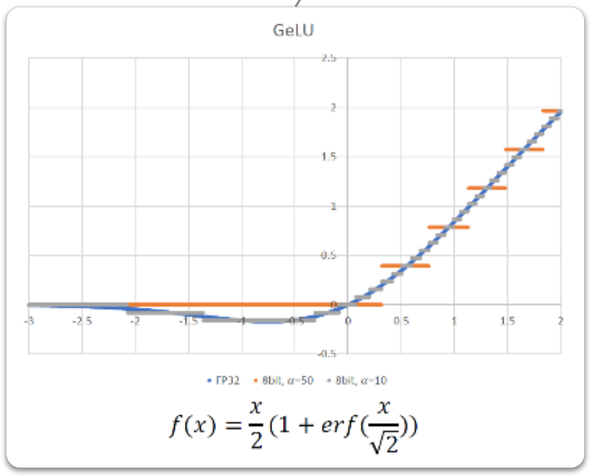
Impact of model design

Model	FP32	Int8 (max)	Int8 (entropy)	Rel Err (entropy)
MobileNet v1	71.01	69.43	69.46	2.18%
MobileNet v2	74.08	73.96	73.85	0.31%
NASNet (large)	82.72	82.09	82.66	0.07%
NASNet (mobile)	73.97	12.95	73.4	0.77%
ResNet50 (v1.5)	76.51	76.11	76.28	0.30%
ResNet50 (v2)	76.37	75.73	76.22	0.20%
ResNet152 (v1.5)	78.22	5.29	77.95	0.35%
ResNet152 (v2)	78.45	78.05	78.15	0.38%
VGG-16	70.89	70.75	70.82	0.10%
VGG-19	71.01	70.91	70.85	0.23%
Inception v3	77.99	77.7	77.85	0.18%
Inception v4	80.19	1.68	80.16	0.04%

Quantization

Impact of model design





- GeLU produces highly asymmetric range
- Negative values between [-0.17,0]
- All negative values clipped to 0
- GeLU10 allows to maintain negative values

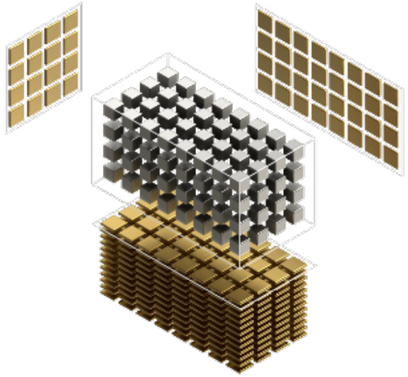
Bert large uncased	FP32	Int8	Rel Err %
MRPC	0.855	0.823	3.74%
SQuAD 1.1 (F1)	91.01	85.16	6.43%

Bert large uncased	FP32	Int8 (GeLU10)	Rel Err %
MRPC	0.855	0.843	0.70%
SQuAD 1.1 (F1)	91.01	90.40	0.67%

Model Pruning

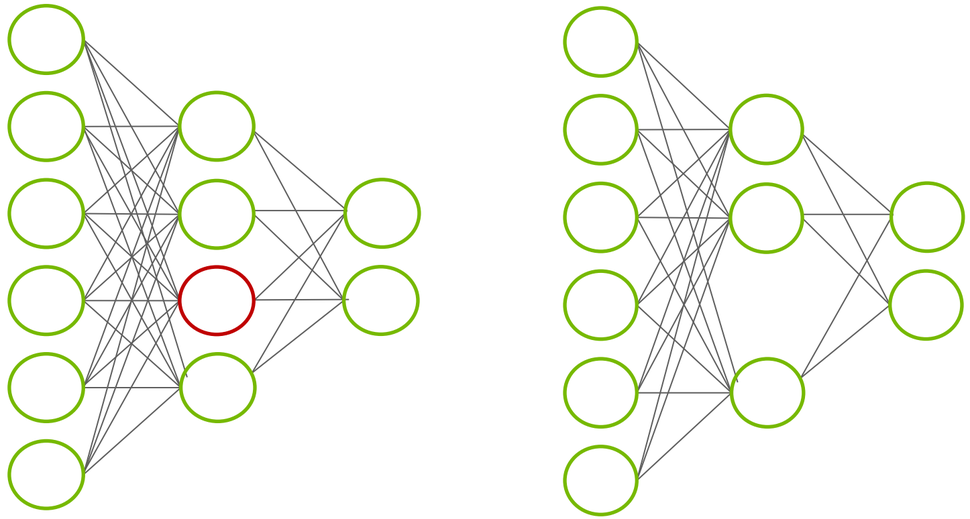
The opportunity:

- Reduced memory bandwidth
- Reduced memory footprint
- Acceleration (especially in presence of hardware acceleration)



2 step process:

1. Reduce model size
2. Incrementally retrain model after pruning to recover accuracy



6 inputs, 6 neurons,
32 connections

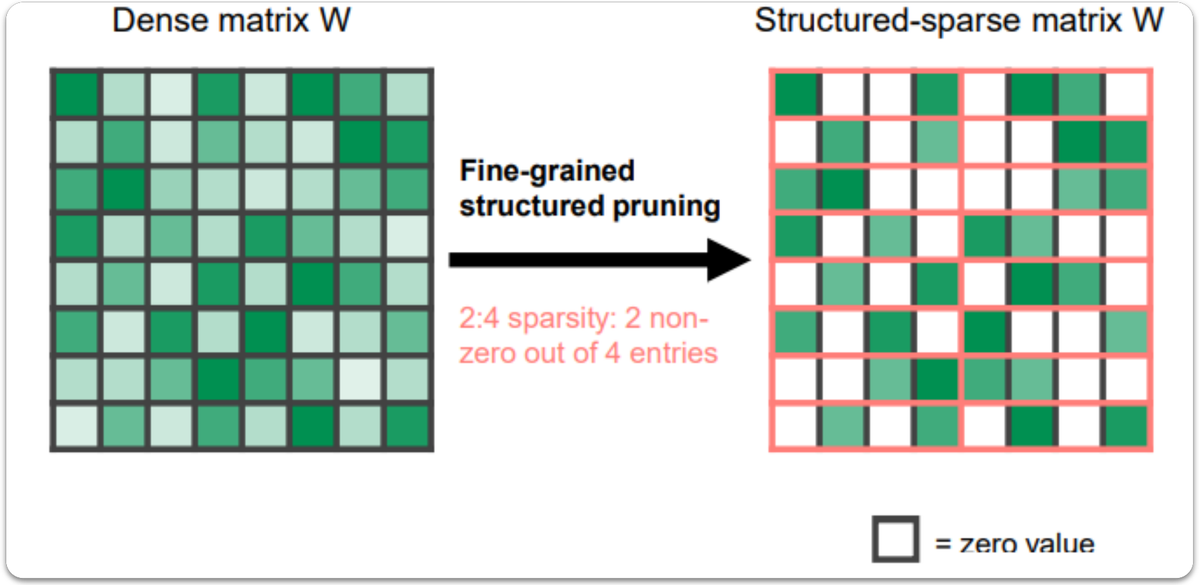
6 inputs, 5 neurons,
24 connections

Model Pruning

Accelerate with Ampere sparse tensor core

- Fine-grained structured sparsity for Tensor Cores
  - 50% fine-grained sparsity
  - 2:4 pattern: 2 values out of each contiguous block of 4 must be 0

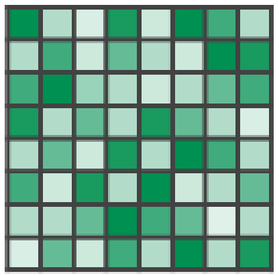
- Addresses the 3 challenges:
  - Accuracy: maintains accuracy of the original, unpruned network
    - Medium sparsity level (50%), fine-grained
  - Training: a recipe shown to work across tasks and networks
  - Speedup:
    - Specialized Tensor Core support for sparse math
    - Structured: lends itself to efficient memory utilization



Model Pruning

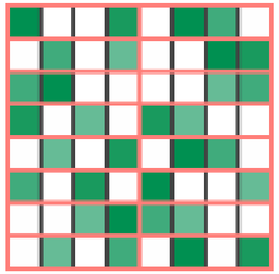
Recipe for 2:4 sparse network training

Dense weights



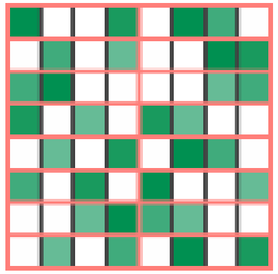
Train (or obtain) a dense network

2:4 sparse weights



Prune for 2:4 sparsity

Retrained 2:4 sparse weights

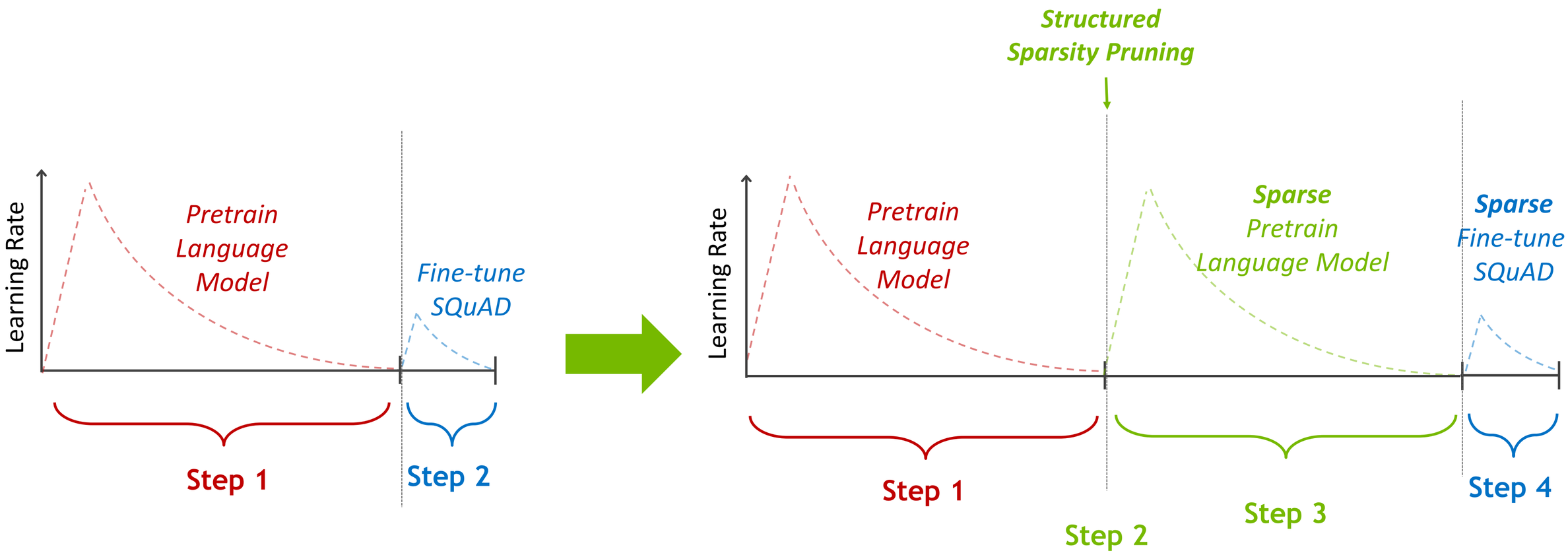


Repeat the original training procedure

- Same hyperparameters as in step-1
- Initialize to weights from step-2
- Maintain the 0 pattern from step-2: no need to recompute the mask

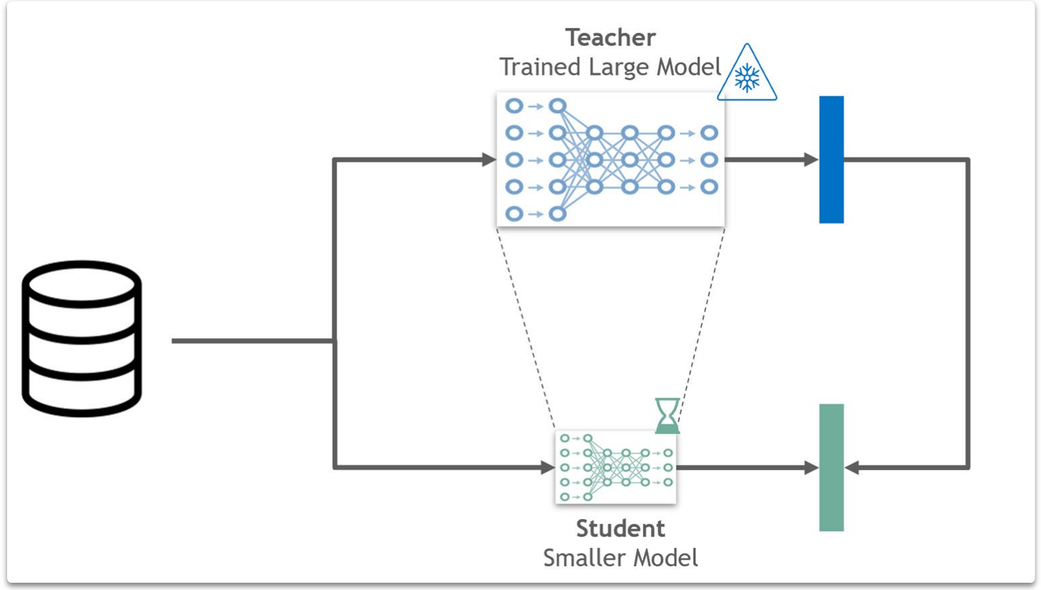
Bert Squad Example

SQuAD dataset and fine-tuning is too small to compensate for pruning on its own



Knowledge Distillation

The idea



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.

Model Optimizations

Some papers

Training with Quantization Noise for Extreme Model Compression

Angela Fan *
Facebook AI Research, LORIA

Pierre Stock *†
Facebook AI Research, Inria

Benjamin Graham
Facebook AI Research

Edouard Grave
Facebook AI Research

Rémi Gribonval †
Inria

Hervé Jégou
Facebook AI Research

Armand Joulin
Facebook AI Research

Abstract

We tackle the problem of producing compact models, maximizing their accuracy for a given model size. A standard solution is to train networks with Quantization Aware Training (Jacob et al., 2018), where the weights are quantized during training and the gradients approximated with the Straight-Through Estimator (Bengio et al., 2013). In this paper, we extend this approach to work beyond `int8` fixed-point quantization with extreme compression methods where the approximations introduced by STE are severe, such as Product Quantization. Our proposal is to only quantize a different random subset of weights during each forward, allowing for unbiased gradients to flow through the other weights. Controlling the amount of noise and its form allows for extreme compression rates while maintaining the performance of the original model. As a result we establish new state-of-the-art compromises between accuracy and model size both in natural language processing and image classification. For example, applying our method to state-of-the-art Transformer and ConvNet architectures, we can achieve 82.5% accuracy on MNLI by compressing RoBERTa to 14 MB and 80.0% top-1 accuracy on ImageNet by compressing an EfficientNet-B3 to 3.3 MB.[1]

"We used Quant-Noise to compress Facebook AI's state-of-the-art RoBERTa Base model from 480 MB to 14 MB while achieving 82.5 percent on MNLI, compared with 84.8 percent for the original model."

SparseGPT: Massive Language Models Can Be Accurately Pruned in One-Shot

Preprint. Version January 19, 2023.

Elias Frantar
IST Austria
elias.frantar@ist.ac.at

Dan Alistarh
IST Austria & Neural Magic
dan.alistarh@ist.ac.at

Abstract

We show for the first time that large-scale generative pretrained transformer (GPT) family models can be pruned to at least 50% sparsity in *one-shot, without any retraining*, at minimal loss of accuracy. This is achieved via a new pruning method called `SparseGPT`, specifically designed to work efficiently and accurately on massive GPT-family models. When executing `SparseGPT` on the largest available open-source models, OPT-175B and BLOOM-176B, we can reach 60% sparsity with negligible increase in perplexity: remarkably, more than 100 billion weights from these models can be ignored at inference time. `SparseGPT` generalizes to semi-structured (2:4 and 4:8) patterns, and is compatible with weight quantization approaches.

GLM-130B: An Open Bilingual Pre-Trained Model

Aohan Zeng‡*, Xiao Liu‡*, Zhengxiao Du‡, Zihan Wang‡, Hanyu Lai‡, Ming Ding‡, Zhuoyi Yang‡, Yifan Xu†, Wendi Zheng‡, Xiao Xia‡, Weng Lam Tam‡, Zixuan Ma‡, Yufei Xue§, Jidong Zhai‡, Wenguang Chen‡, Peng Zhang§, Yuxiao Dong‡, Jie Tang‡†

Tsinghua University‡ Zhipu.AI§

Abstract

We introduce GLM-130B, a bilingual (English and Chinese) pre-trained language model with 130 billion parameters. It is an attempt to open-source a 100B-scale model at least as good as GPT-3 and unveil how models of such a scale can be successfully pre-trained. Over the course of this effort, we face numerous unexpected technical and engineering challenges, particularly on loss spikes and disconvergence. In this paper, we introduce the training process of GLM-130B including its design choices, training strategies for both efficiency and stability, and engineering efforts. The resultant GLM-130B model offers significant outperformance over GPT-3 175B on a wide range of popular English benchmarks while the performance advantage is not observed in OPT-175B and BLOOM-176B. It also consistently and significantly outperforms ERNIE TITAN 3.0 260B—the largest Chinese language model—across related benchmarks. Finally, we leverage a unique scaling property of GLM-130B to reach INT4 quantization, without quantization aware training and with almost no performance loss, making it the first among 100B-scale models. More importantly, the property allows its effective inference on 4×RTX 3090 (24G) or 8×RTX 2080 Ti (11G) GPUs, the most ever affordable GPUs required for using 100B-scale models. The GLM-130B model weights are publicly accessible and its code, training logs, related toolkit, and lessons learned are open-sourced at `https://github.com/THUDM/GLM-130B`.

8-bit Optimizers via Block-wise Quantization

Tim Dettmers*‡ Mike Lewis* Sam Shleifer* Luke Zettlemoyer*‡
Facebook AI Research*, {mikelewis,sshleifer}@fb.com
University of Washington‡, {dettmers,lsz}@cs.washington.edu

Abstract

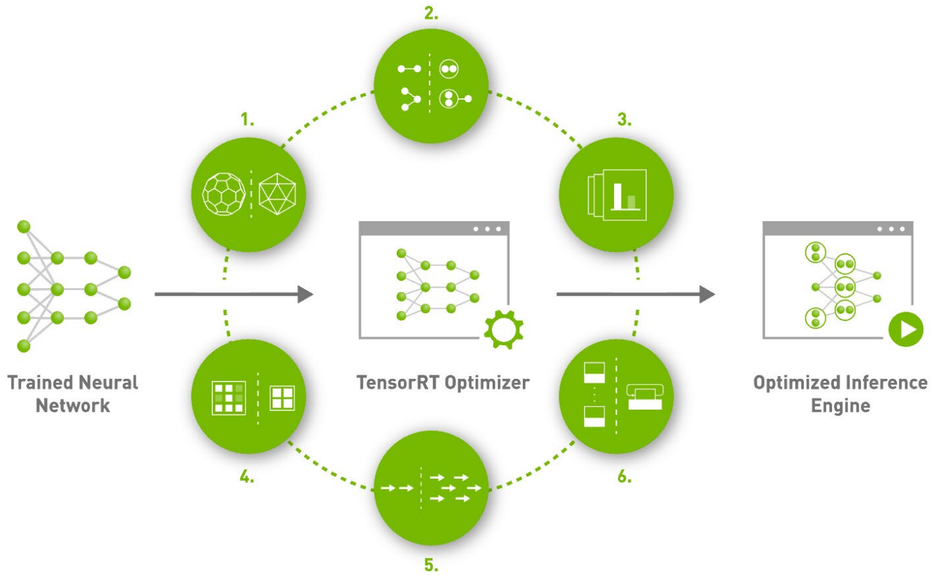
Stateful optimizers maintain gradient statistics over time, e.g., the exponentially smoothed sum (SGD with momentum) or squared sum (Adam) of past gradient values. This state can be used to accelerate optimization compared to plain stochastic gradient descent but uses memory that might otherwise be allocated to model parameters, thereby limiting the maximum size of models trained in practice. In this paper, we develop the first optimizers that use 8-bit statistics while maintaining the performance levels of using 32-bit optimizer states. To overcome the resulting computational, quantization, and stability challenges, we develop block-wise dynamic quantization. Block-wise quantization divides input tensors into smaller blocks that are independently quantized. Each block is processed in parallel across cores, yielding faster optimization and high precision quantization. To maintain stability and performance, we combine block-wise quantization with two additional changes: (1) dynamic quantization, a form of non-linear optimization that is precise for both large and small magnitude values, and (2) a stable embedding layer to reduce gradient variance that comes from the highly non-uniform distribution of input tokens in language models. As a result, our 8-bit optimizers maintain 32-bit performance with a small fraction of the memory footprint on a range of tasks, including 1.5B parameter language modeling, GLUE finetuning, ImageNet classification, WMT'14 machine translation, MoCo v2 contrastive ImageNet pretraining+finetuning, and RoBERTa pretraining, without changes to the original optimizer hyperparameters. We open-sourceour 8-bit optimizers as a drop-in replacement that only requires a two-line code change.

!

Choice of the Model Optimization Tool

TensorRT

Key features



1. Reduce Mixed Precision

Maximizes throughput by quantizing models to INT8 while preserving accuracy

2. Layer and Tensor Fusion

Optimizes use of GPU memory and bandwidth by fusing nodes in a kernel

3. Kernel Auto-Tuning

Selects best data layers and algorithms based on the target GPU platform

4. Dynamic Tensor Memory

Minimizes memory footprint and reuses memory for tensors efficiently

5. Multi-Stream Execution

Uses a scalable design to process multiple input streams in parallel

6. Time Fusion

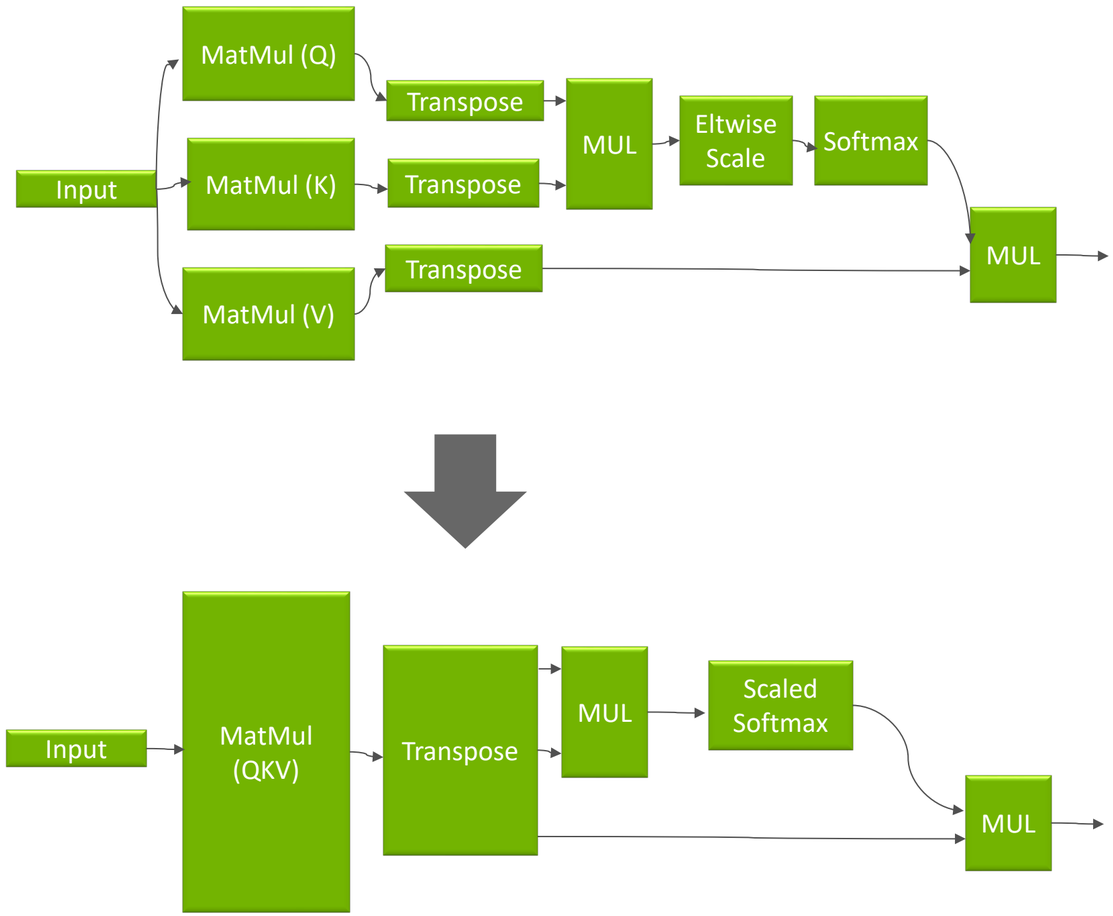
Optimizes recurrent neural networks over time steps with dynamically generated kernels

TensorRT

Layer & tensor fusion

Optimizes use of GPU memory and bandwidth by fusing nodes in a kernel

- Combines successive nodes into a single node, making single kernel execution
- Significantly reduces number of layers to compute, resulting in faster performance
- Eliminates unnecessary memory traffic by removing concat/slice layers

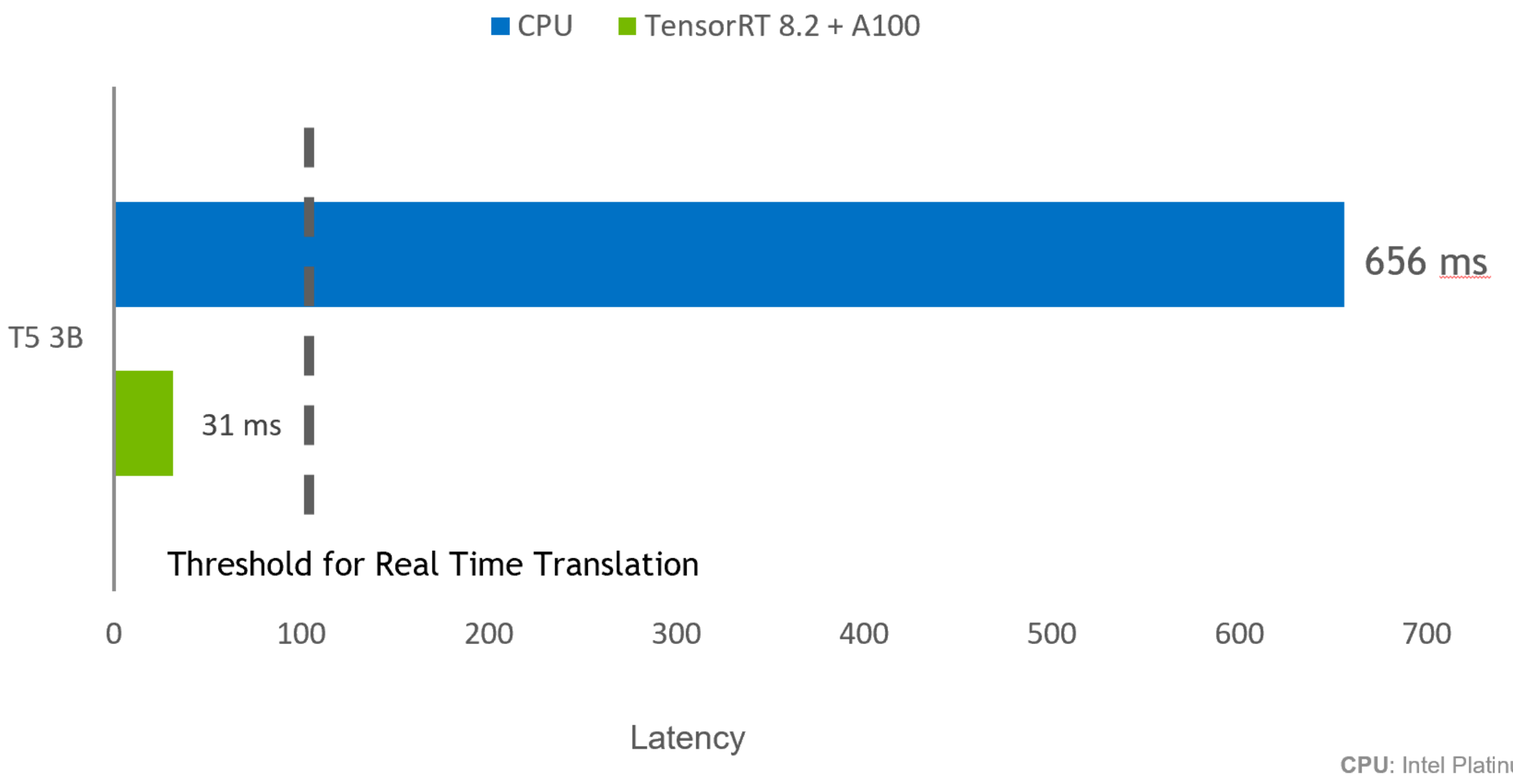


Supported fusion list

developer.nvidia.com/tensorrt

TensorRT

Transformer



CPU: Intel Platinum 8380, 2 socket
GPU: NVIDIA A100 PCIE 80GB
S/W: PyTorch 1.9, TensorRT 8.2.0 EA

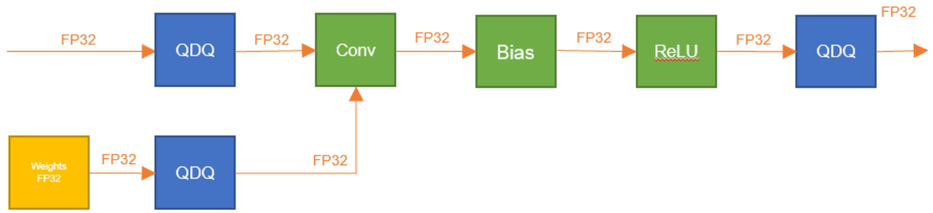
T5/GPT-2 demo code in TensorRT OSS github repo

TensorRT

Quantization-aware training (QAT)

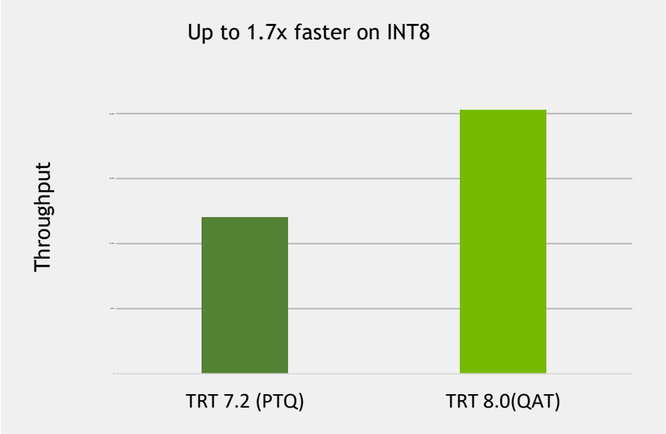
Improved accuracy for INT8 inference

- Better accuracy compared to Post Training Quantization (PTQ)
- Quantize state of the art models with minimal loss of accuracy
- TensorRT optimizes the Q/DQ graph for inference without compromising performance



Training with Q/DQ

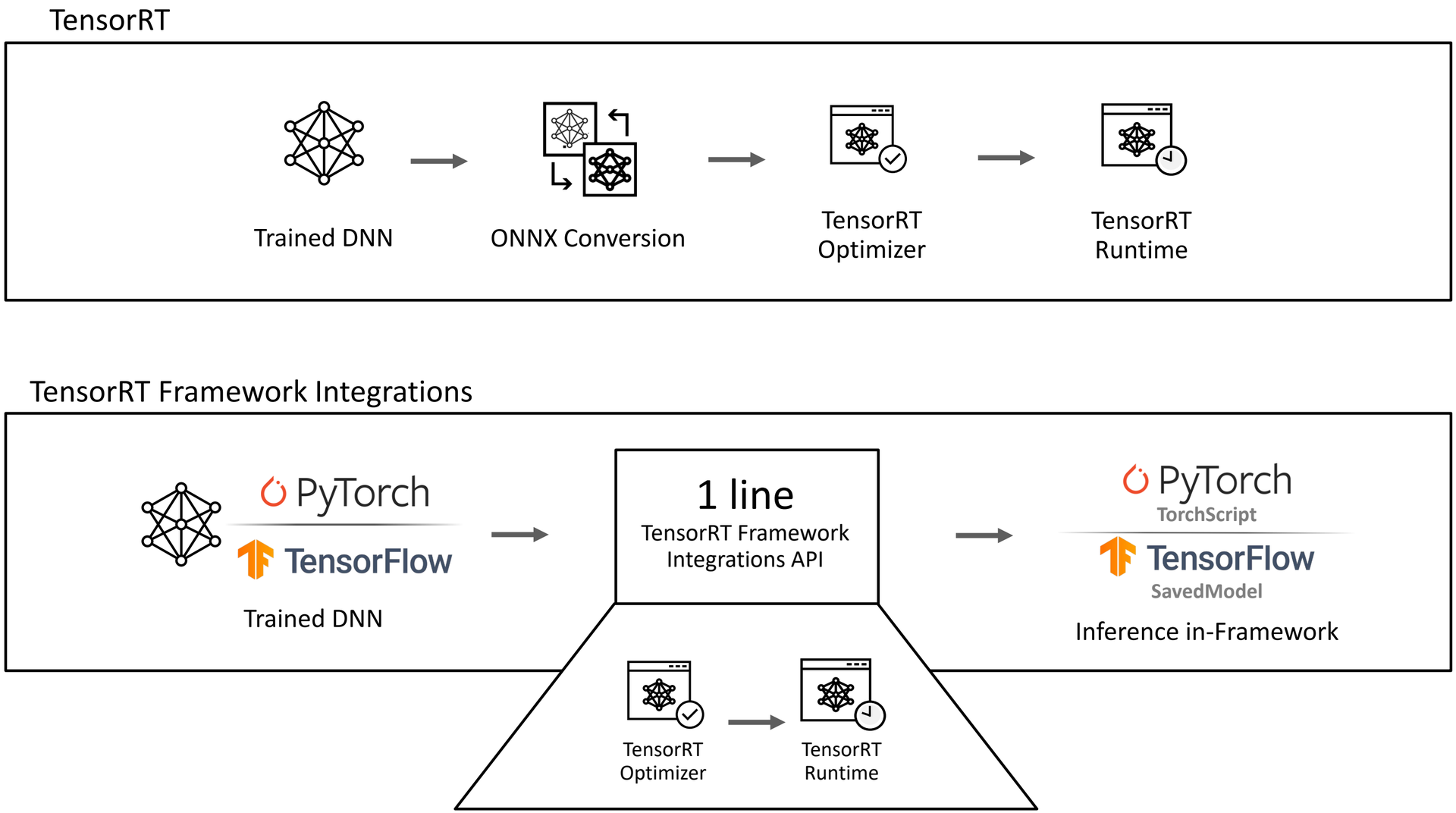
	FP32 & FP16	INT8 PTQ	INT8 QAT
Top-1 acc	77.4	33.93	76.8



GPU: A100, Network: EfficientNet-B0, BS=64

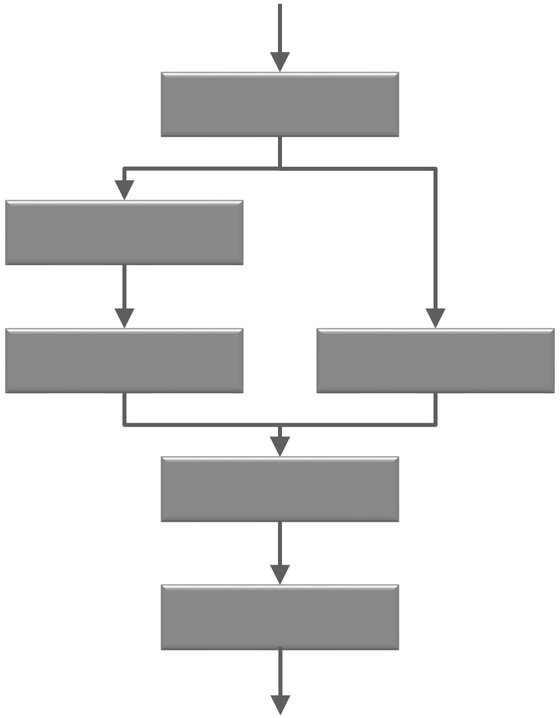
https://github.com/NVIDIA/TensorRT/tree/master/tools/pytorch-quantization

Inference Optimization Workflow for TensorRT

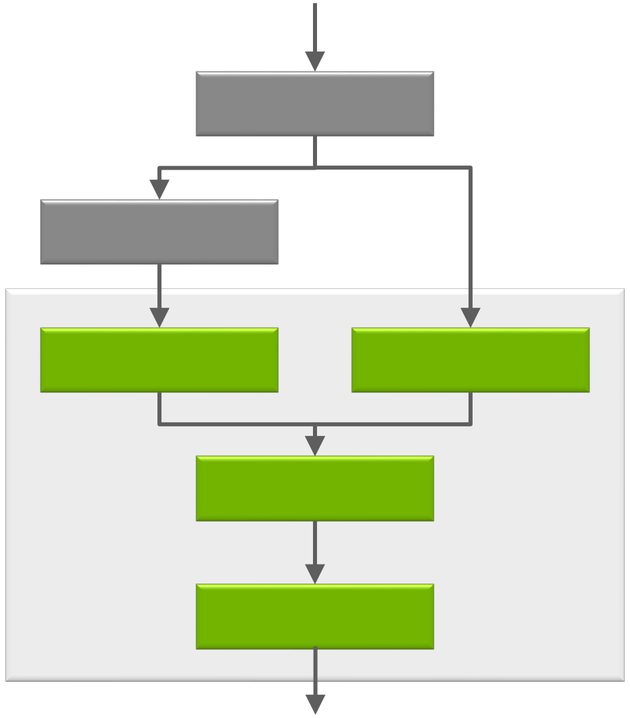


Framework Integrations

Conversion workflow

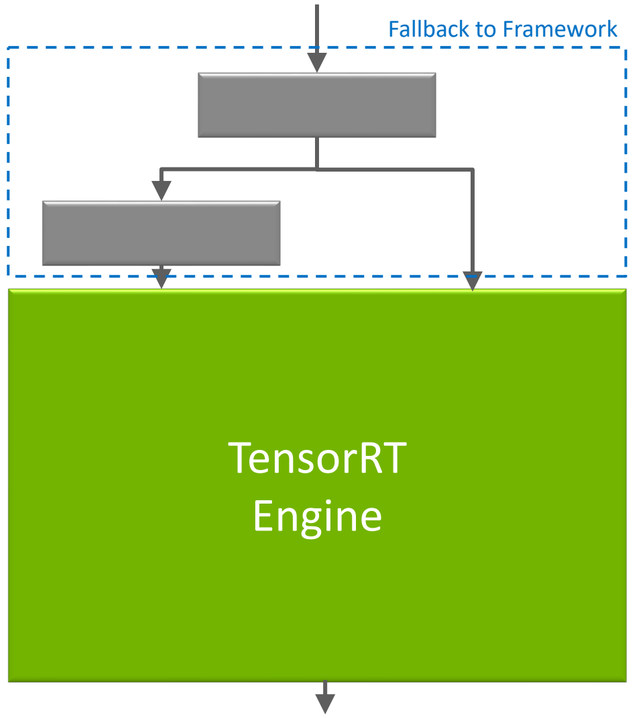


Native Model



1. Partition Model

Identify supported nodes



2. Convert Model

Convert nodes to TensorRT

TensorRT Integrated With PyTorch and TensorFlow

Up to 4X faster inference with one 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

Sparsity Support

APEX automatic sparsity - TensorRT

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

Training Phase

```
# 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)
```

Inference Phase

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

ASP: https://github.com/NVIDIA/apex/tree/master/apex/contrib/sparsity

TensorRT

Widely adopted

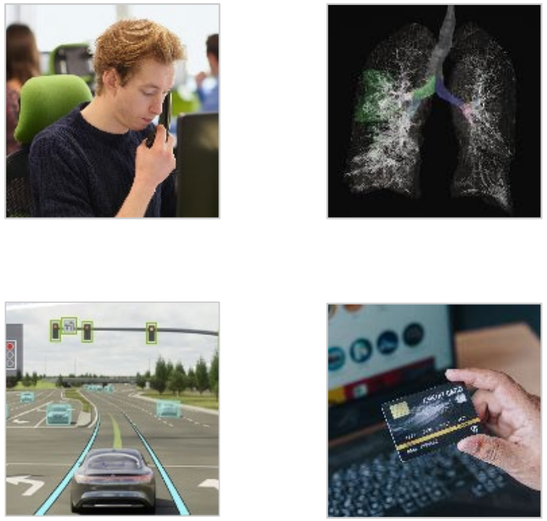
2.5M
Downloads

350K+
Developers

600+
Inference Services

Growing Ecosystem
3X developers in 2020 vs. 2019

Customers
Trusted by 27,500 Companies



Cloud to Edge
Delivers across Industries

Deploy (Part 3)

Lecture

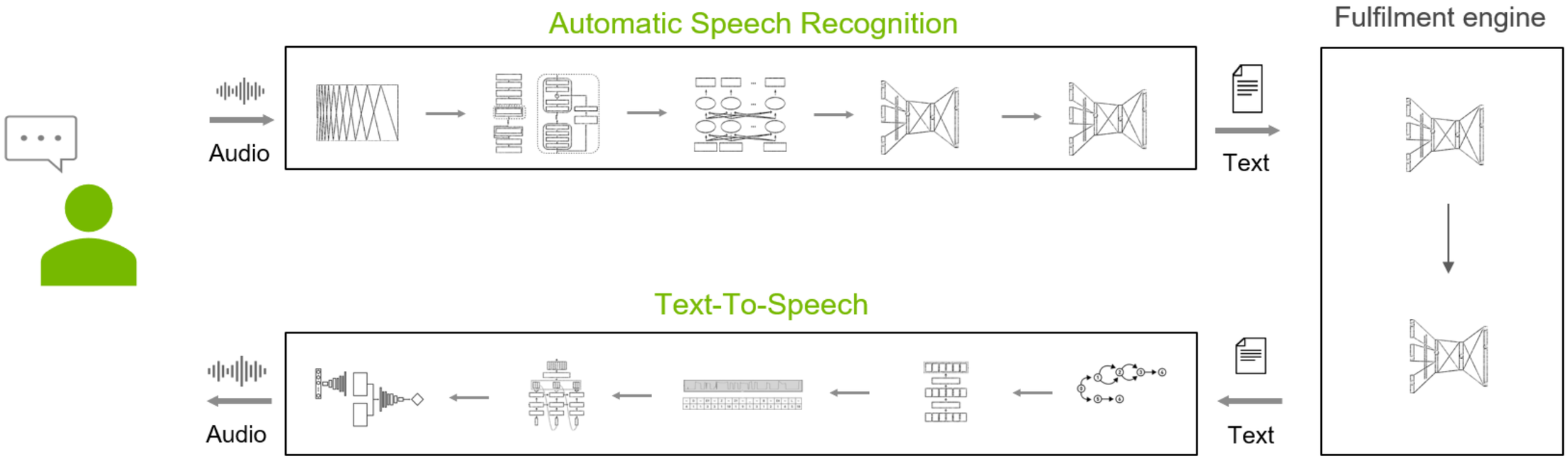
- Beyond Model Accuracy, Meeting Application Requirements
- The Right Model
- The Right Implementation
- Model Optimization Techniques
- Beyond a Single Model
- Real-World Applications

Lab

- Making Kubernetes GPU-Aware
- First GPU-Accelerated Application
- Monitoring Your GPUs
- Deploying Riva
- Using Riva

Build the Application

Deploy several neural networks



Challenges In Model Deployment

Ease of Use

Multiple Frameworks

Requires Retooling In Production

Different Types Of Use Cases:

Real Time, Batch, Streaming, Ensembles
Each Requires Serving Optimizations

Cost of Deployment

Need For Mixed Infrastructure - CPU & GPU on Cloud, Data Center & Edge

Disparate Serving Solutions Increase Cost

High Performance, Low Latency Inference on GPU and CPU

Hardware Underutilization Increases Cost

Ecosystem Tools

Scaled Deployment Needs Cluster Orchestration Tools

Lack of Integration w/ Standard Tools Drives Costly/Non-scalable Implementations

NVIDIA
TRITON INFERENCE SERVER

Nvidia Triton Inference Server

Triton takes care of plumbing to deploy models for inference

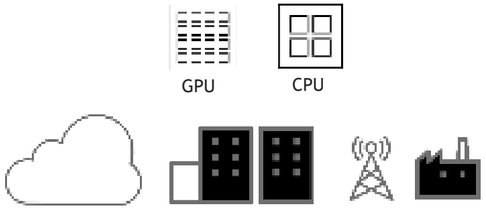
Standardized Inference Platform

TensorFlow
TensorRT
OpenVINO
PYTORCH
Custom Backends
ONNX RUNTIME
XGBoost
learn
(Random Forest)
LightGBM
(For non-categorical features)

Support for All Major Framework Backends for Flexibility & Consistency

Standard HTTP, gRPC, and C++ Communication

Inferencing on GPU and CPU

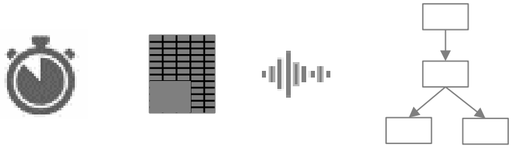


Inference Serving on GPU, CPU, and Mixed Workloads

Cloud | Data Center | Edge

Bare metal | Virtualization

Different Types of Queries



Real time Batch Stream Ensemble

Support for Different Types of Inference Queries and Use Cases

Maximized Hardware Utilization

Concurrent Model Execution Increases Throughput & Utilization, lowering TCO

Dynamic Batching Maximizes Throughput Under Latency Constraint

NVIDIA
TRITON INFERENCE SERVER

Nvidia Triton Inference Server

Triton integrates easily in organization workflows for all ai use cases

Kubernetes Integration

Scalable Microservice In Kubernetes

Helm Chart For Fast Deployment

KFServing Integration

MLOps

Live Model Updates | Dynamic Model Loading | Triton Model Analyzer

Integrated with Azure ML, Seldon, ClearML, and AWS SageMaker and Google CAIP as Custom Container

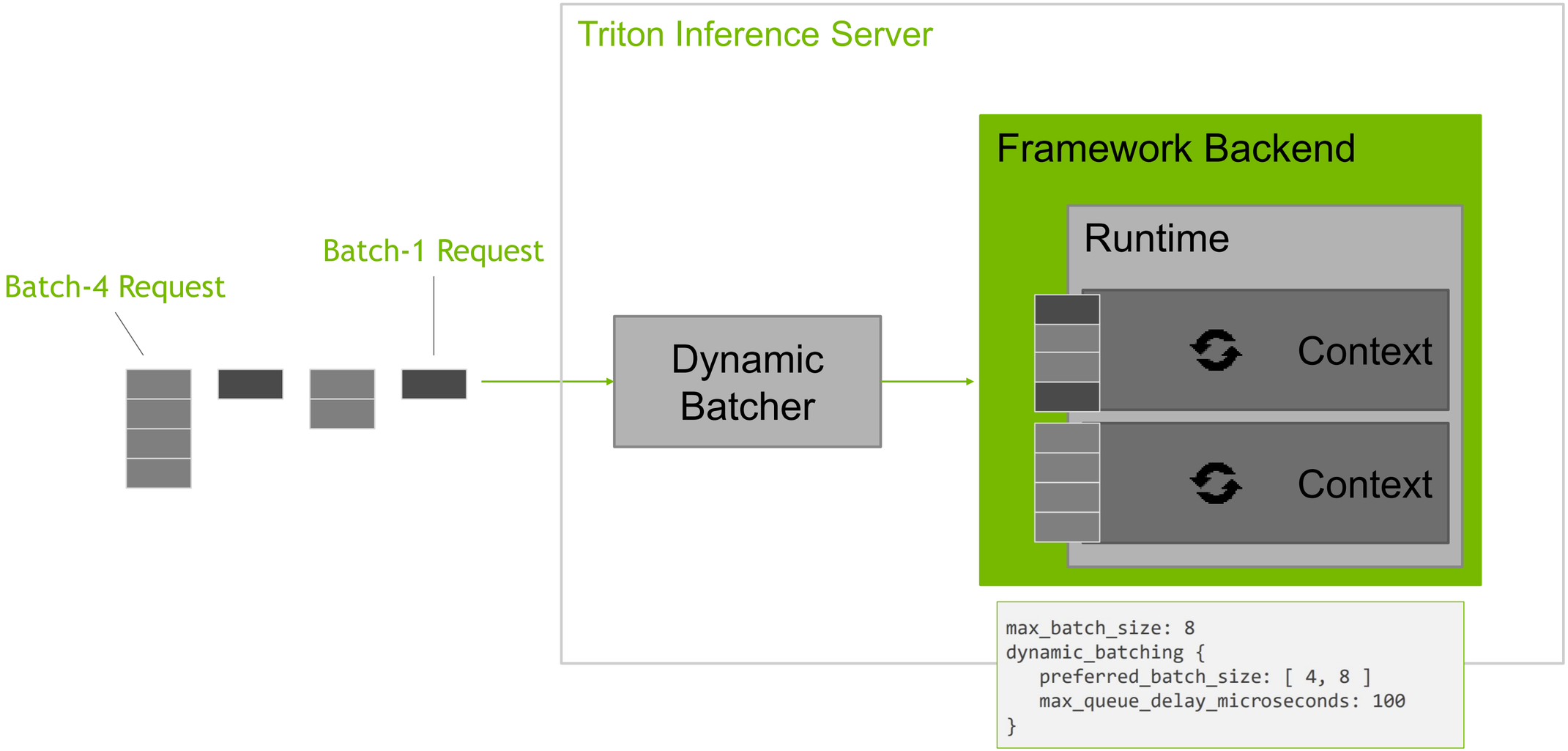
Open Source & Customizable

Completely Open Source – Inspect, Customize & Extend

Customizable Container – Modular Backends For Low Memory Footprint

Nvidia Triton Inference Server

Dynamic batching scheduler

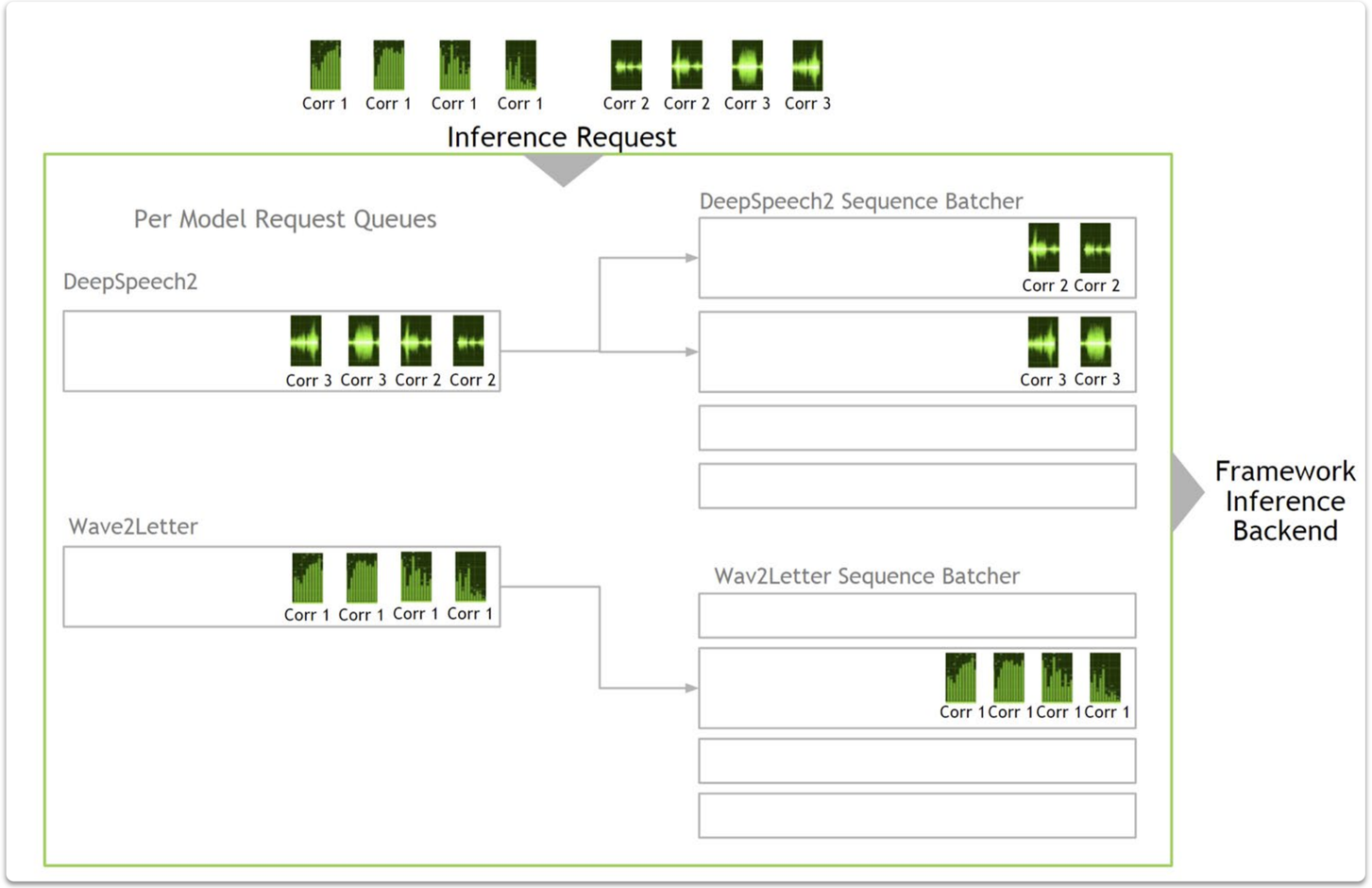


Nvidia Triton Inference Server

Streaming inference requests

Streaming API

Based on the correlation ID, the audio requests are sent to the appropriate batch slot in the sequence batcher*

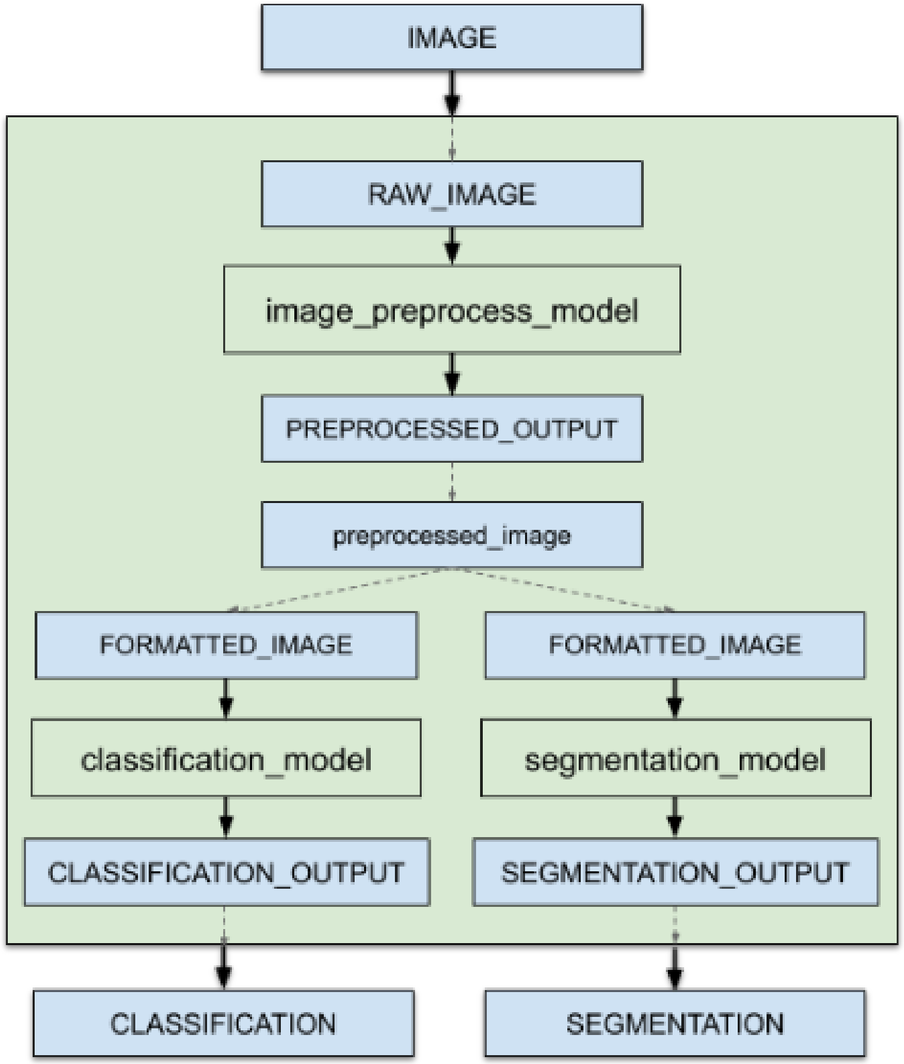


*Correct order of requests is assumed at entry into the endpoint
Note: Corr = Correlation ID

Nvidia Triton Inference Server

Model ensembling

- Pipeline of one or more models and the connection of input and output tensors between those models
- Use for model stitching or data flow of multiple models such as data preprocessing → inference → data post-processing
- Collects the output tensors in each step, provides them as input tensors for other steps according to the specification
- Ensemble models will inherit the characteristics of the models involved, so the meta-data in the request header must comply with the models within the ensemble

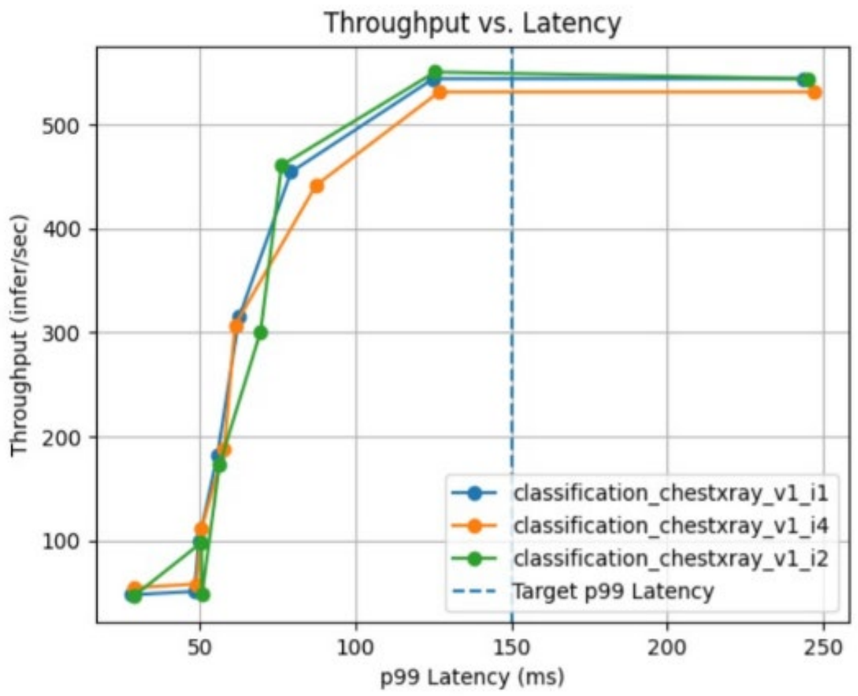


Nvidia Triton Inference Server

Model analyzer

Tool to automatically identify optimal model configurations to maximize performance

1. Specify constraints on performance results, latency and throughput
1. Model Analyzer automatically sweeps through different configurations (batch sizes, model instances, etc.)
1. Finds the best configuration for optimal performance under constraints
1. Analyze results through reports and charts



Throughput vs. Latency curves for 3 configurations of classification_chestxray_v1

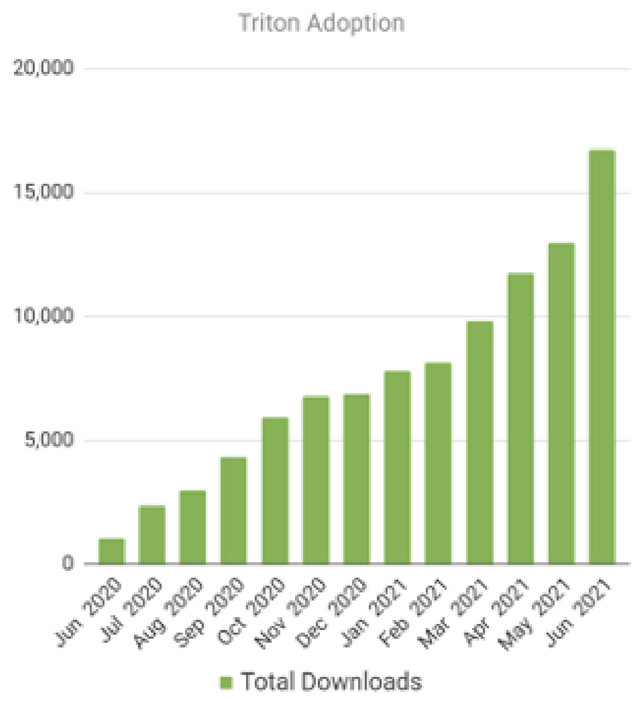
GitHub repo and docs: https://github.com/triton-inference-server/model_analyzer

NVIDIA
TRITON INFERENCE SERVER

Nvidia Triton Inference Server

Broad use cases and integrations

Adoption



Unique NGC pulls + GitHub clones

Publicly Referenceable Success Stories

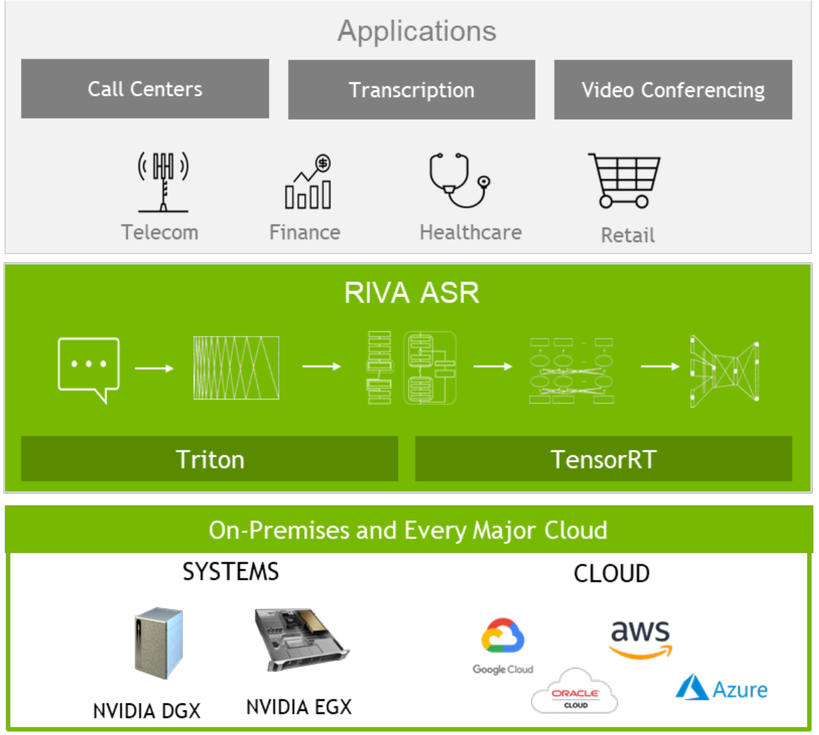
- American Express — Fraud Detection
- Ant Group — Fraud Detection*
- NAVER — Image Classification & Recommendation
- Office 365 — Grammar Check
- Pinterest — Image Segmentation
- BPIL — Defect Detection
- Tencent 腾讯 — Video Content Audit
- TracXpoint — Product Identification
- United States Postal Service — Package Analytics

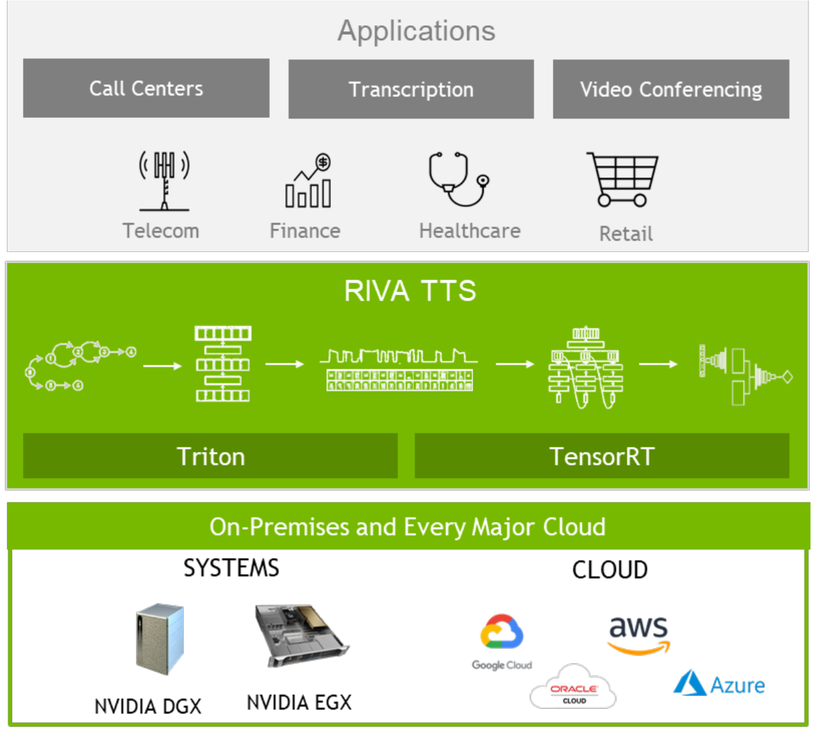
Platform Integrations

- Amazon EKS
- Azure AKS
- Azure Machine Learning
- Google GKE
- Google Vertex AI Custom Container
- ClearML
- Kubeflow/KFServing
- Prometheus
- Red Hat OpenShift
- Seldon
- VMware vSphere*

NVIDIA Riva

Uses TensorRT and Triton Inference Server





Deploy (Part 3)

Lecture

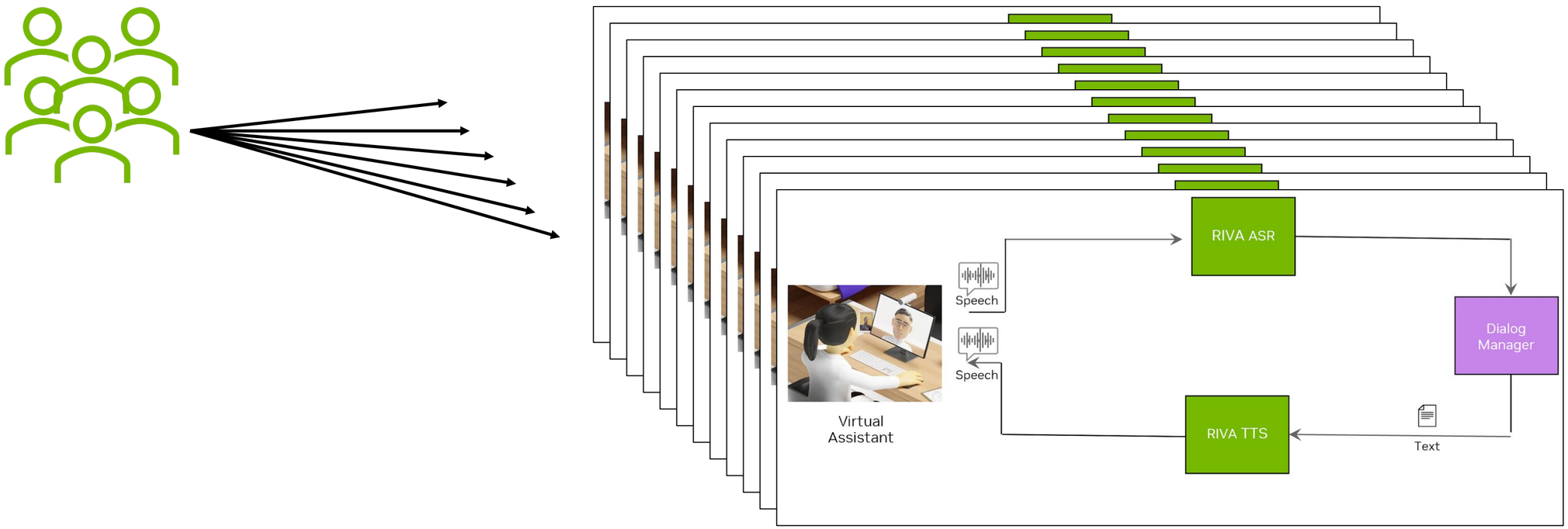
- Beyond Model Accuracy, Meeting Application Requirements
- The Right Model
- The Right Implementation
- Model Optimization Techniques
- Beyond a Single Model
- Real-World Applications

Lab

- Making Kubernetes GPU-Aware
- First GPU-Accelerated Application
- Monitoring Your GPUs
- Deploying Riva
- Using Riva

Real-World Applications Challenges

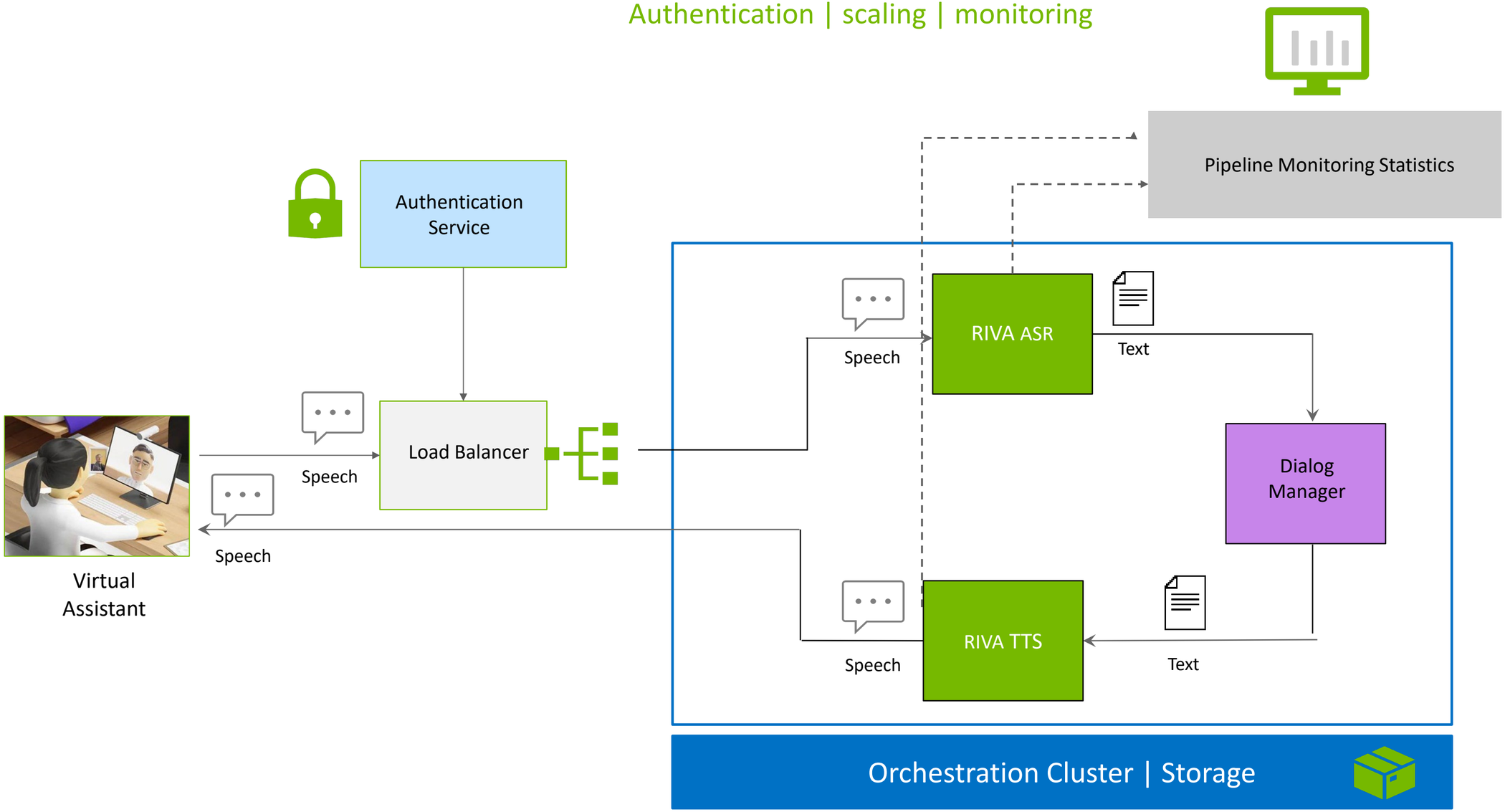
Authentication | scaling | monitoring



Application: N
Version: K

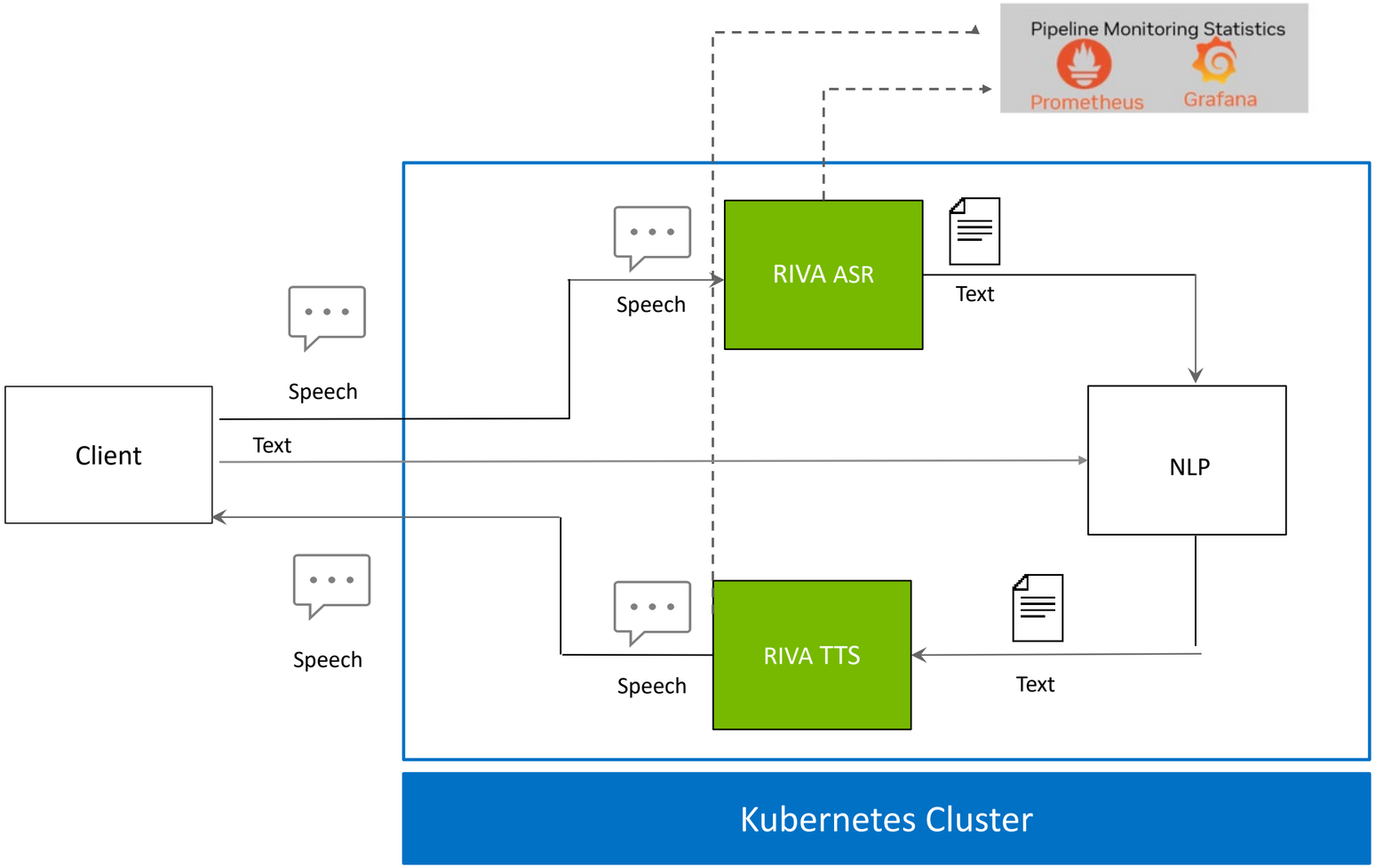
Real-World Applications Challenges

Authentication | scaling | monitoring



Real-World Applications Challenges

Authentication | scaling | monitoring



Monitoring

Telemetry

- Telemetry:
  - Involves automatically collecting and analyzing data received from a distance
  - Measure physical quantities; hardware counters; and software specific metrics
  - Related to **logging** used to more in depth diagnose errors and examine state flows

- Why?
  - Identify bottlenecks in the pipeline
  - Alert notifications (out of memory, thread trashing, swapping, …)
  - Calculating and measuring utilization

Monitoring

Data center GPU manager - DCGM

GPU Utilization
for capacity planning

Tensor core Utilization
for higher app performance

Memory & Bandwidth Usage
for optimal app performance

NVLink/PCIe Usage
for optimal app performance

GPU Power consumption
for GPU health

GPU Temperature
for GPU health

GPU errors
for GPU health

NVIDIA DCGM

7 GPU METRICS TO MONITOR FOR HIGHER AVAILABILITY, UTILIZATION & APPLICATION PERFORMANCE

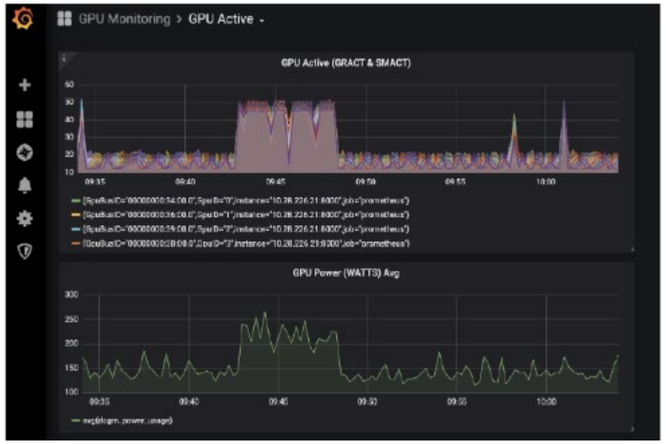
Monitoring

DCGM integration with monitoring tools

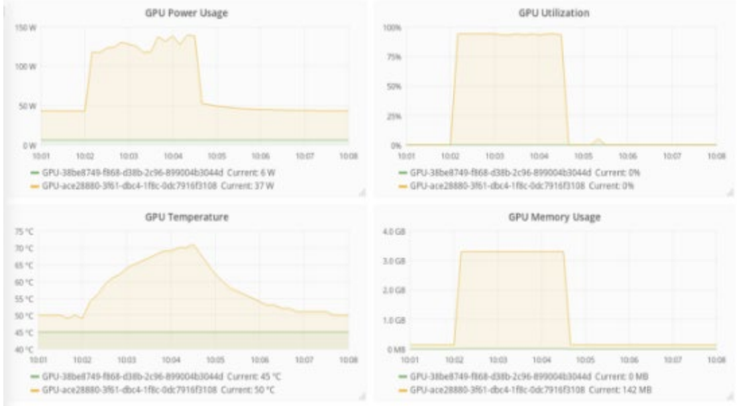
GPU metrics from DCGM can be exported into *Prometheus* *Kubernetes* *collectd* *Telegraf* monitoring/metrics

Easy visualization e.g. Grafana Prometheus/collectd

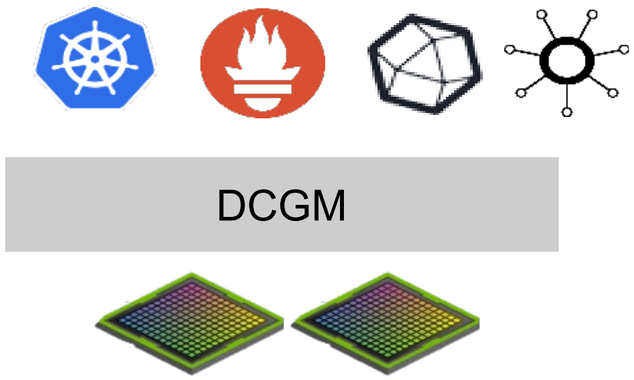
DCGM available for GPUs on-premises data center and public clouds



Visualization via Prometheus & Grafana



Visualization via collectd & Grafana



Monitoring

General dashboard

Monitoring

Riva Metric Examples

Metric name	Metrics formula	Definition
Average queue time	avg(delta(nv_inference_queue_duration_us[1m])/(1+delta (nv_inference_request_success[1m]))/1000)	Time in milliseconds a request stays in the Triton Inference Server queue averaged over all the requests in a one-minute time window. This is a measure of your server's computing capability; if this increases over a threshold, consider scaling your server to more replicas
Number of successful requests per minute	sum(delta(nv_inference_request_success[1m]))	Total number of successful inference requests captured over a one minute time window
Number of failed requests per minute	sum(delta(nv_inference_request_failure[1m]))	Total number of successful inference requests captured over a one minute time window
P99 latency is seconds	quantile_over_time(0.99, nv_inference_compute_infer_duration_us[1m]) / 1000 / 1000	The p99 ASR latency of all the request samples captured over a one minute time window
P95 latency is seconds	quantile_over_time(0.95, nv_inference_compute_infer_duration_us[1m]) / 1000 / 1000	The p95 ASR latency of all the request samples captured over a one minute time window
GPU memory utilization	avg(nv_gpu_memory_used_bytes / 1024 / 1024 / 1024)	GPU memory used by the Riva server
Number of Riva Servers	count (count by (instance) (nv_cache_hit_lookup_duration_per_model))	Riva server replicas on the Kubernetes cluster.
GPU Utilization	avg(nv_gpu_utilization)	Average GPU utilization
GPU Power Utilization	avg(nv_gpu_power_usage)	GPU power consumption over a period of time

Deploy (Part 3)

Lecture

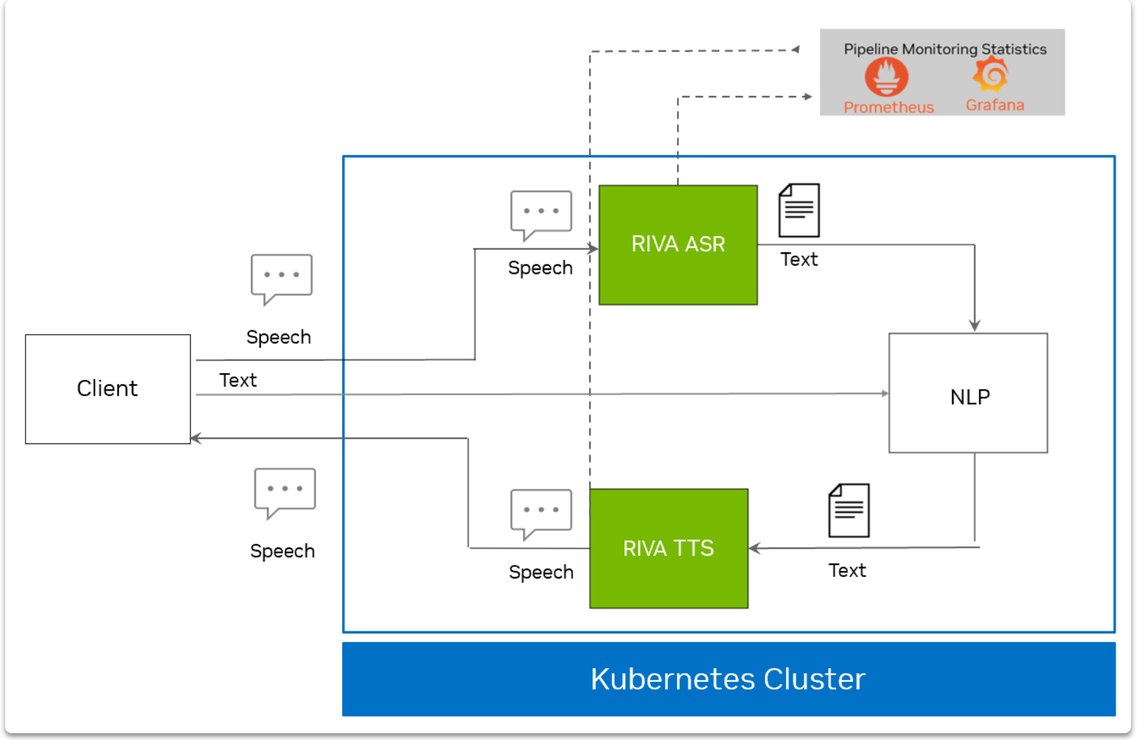
- Beyond Model Accuracy, Meeting Application Requirements
- The Right Model
- The Right Implementation
- Model Optimization Techniques
- Beyond a Single Model
- Real-World Applications

Lab

- Making Kubernetes GPU-Aware
- First GPU-Accelerated Application
- Monitoring Your GPUs
- Deploying Riva
- Using Riva

Lab Overview

1. Deploy a Kubernetes cluster and make Kubernetes GPU-Aware
2. First GPU-accelerated application
3. Deploy Riva ASR and TTS on the Kubernetes cluster
4. Request Riva services from Kubernetes cluster
5. Monitoring your Application with Prometheus and Grafana,
    1. Riva provides Prometheus metrics indicating GPU and request statistics in plain text. Metrics exposed on server:8002/metrics.
    2. Metrics are viewed using a sample Grafana dashboard.



!

Lab

NVIDIA