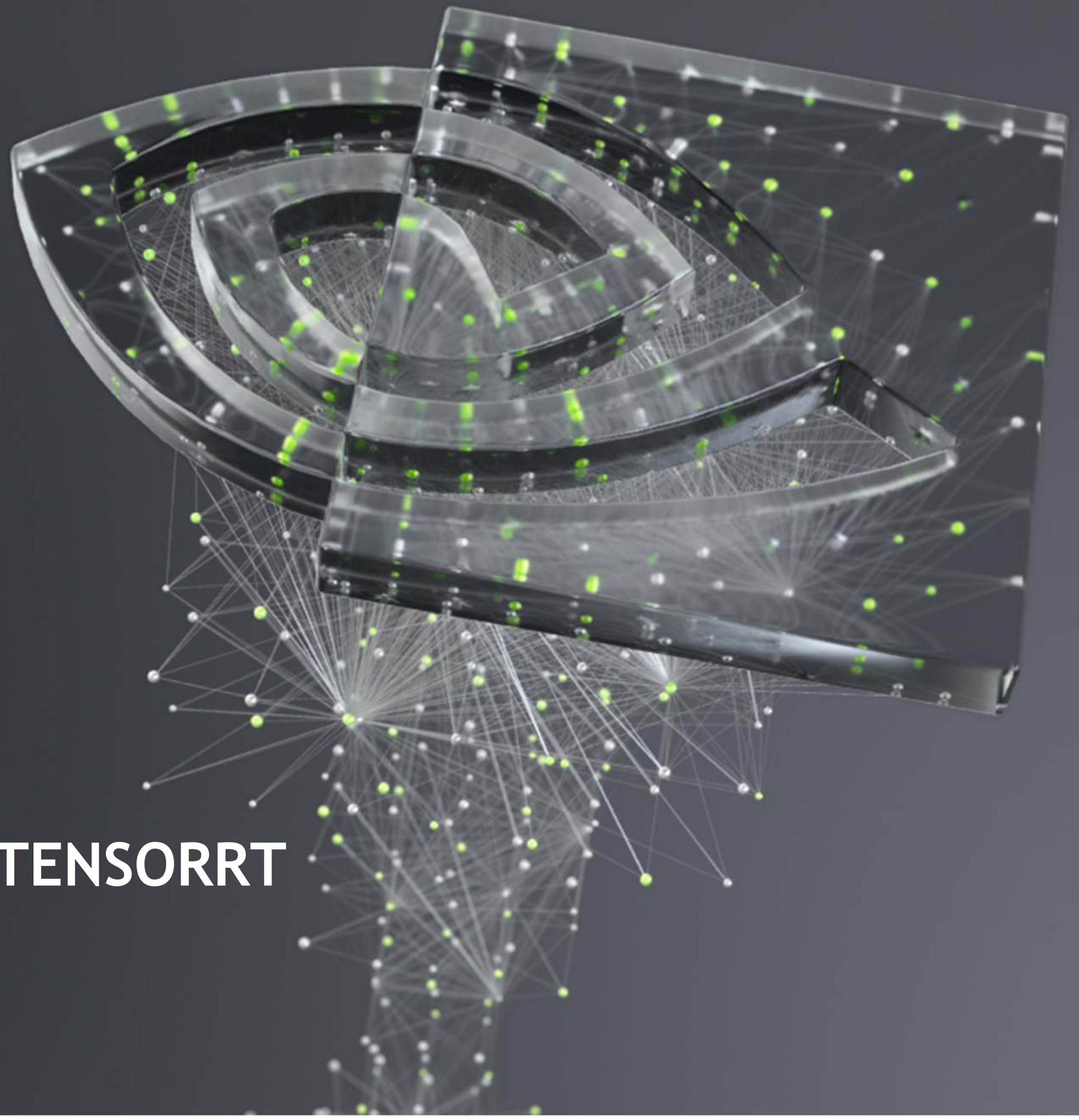
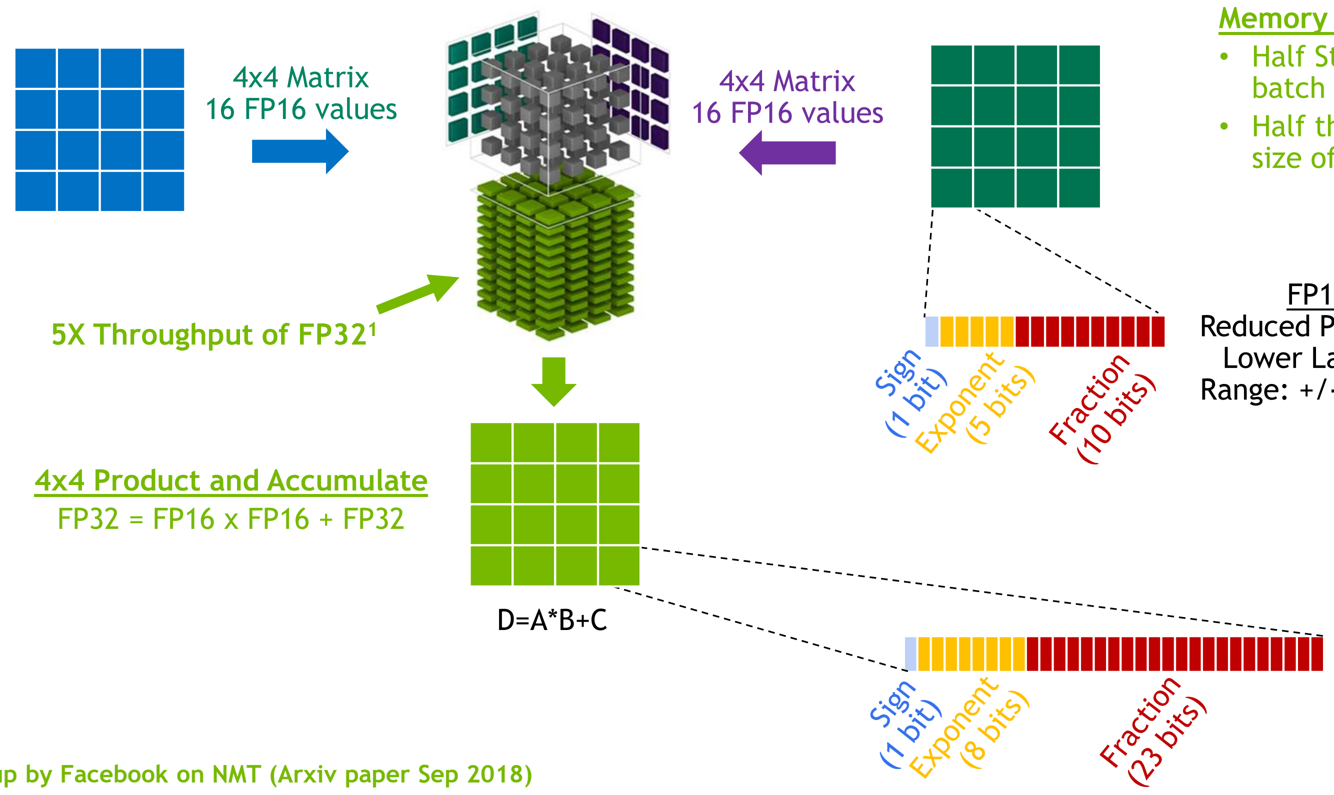


AUTOMATIC MIXED PRECISION & TENSORRT

MIXED PRECISION?

TENSOR CORES BUILT FOR AI AND HPC

Mixed Precision Accelerator – Delivering Up To 5X Throughput of FP32[1]



4x4 Product and Accumulate

FP32 = FP16 x FP16 + FP32

Memory Savings

- Half Storage Requirements (larger batch size)
- Half the memory traffic by reducing size of gradient/activation tensors

FP16
Reduced Precision
Lower Latency
Range: +/- 65,504

FP32
Higher Precision
Range: +/- 3.402823×10^{38}

[1]Fastest Tensor Core Speedup by Facebook on NMT (Arxiv paper Sep 2018)
https://arxiv.org/pdf/1806.00187.pdf

Matching Accuracy for FP32 and Mixed Precision

Model Script	Framework	Data Set	Automatic or Manual Mixed-Precision	FP32 Accuracy	Mixed-Precision Accuracy	FP32 Throughput	Mixed-Precision Throughput	Speedup
BERT Q&A (2)	**TensorFlow**	SQuaD	AMP	90.83 Top 1	90.99 Top 1	66.65 sentences/sec	129.16 sentences/sec	1.94
SSD w/RN50 (1)	**TensorFlow**	COCO 2017	AMP	0.268 mAP	0.269 mAP	569 images/sec	752 images/sec	1.32
GNMT (3)	**PyTorch**	WMT16 English to German	Manual	24.16 BLEU	24.22 BLEU	314,831 tokens/sec	738,521 tokens/sec	2.35
Neural Collaborative Filter (1)	**PyTorch**	MovieLens 20M	Manual	0.959 HR	0.960 HR	55,004,590 samples/sec	99,332,230 items/sec	1.81
U-Net Industrial (1)	**TensorFlow**	DAGM 2007	AMP	0.965-0.988	0.960-0.988	445 images/sec	491 images/sec	1.10
ResNet-50 v1.5 (1)	**MXNet**	ImageNet	Manual	76.67 Top 1%	76.49 Top 1%	2,957 images/sec	10,263 images/sec	3.47
Tacotron 2 / WaveGlow 1.0 (1)	**PyTorch**	LJ Speech Dataset	AMP	0.3629/ -6.1087	0.3645/ -6.0258	10,843 tok/s 257,687 smp/s	12,742 tok/s 500,375 smp/s	1.18/ 1.94

Values are measured with model running on (1) **DGX-1V 8GPU 16G,** (2) **DGX-1V 8GPU 32G or** (3) **DGX-2V 16GPU 32G**

ENABLING AUTOMATIC MIXED PRECISION

Add Just A Few Lines of Code, Get Upto 3X Speedup

TensorFlow

NVIDIA Container 19.07+, TF 1.14+ and TF 2+, explicit optimizer wrapper available:

```
opt = tf.train.experimental.enable_mixed_precision_graph_rewrite(opt)

os.environ['TF_ENABLE_AUTO_MIXED_PRECISION'] = '1'
OR
export TF_ENABLE_AUTO_MIXED_PRECISION=1
```

PyTorch

APEX

```
model, optimizer = amp.initialize(model, optimizer, opt_level="O1")
with amp.scale_loss(loss, optimizer) as scaled_loss:
    scaled_loss.backward()
```

MXNet

```
amp.init()
amp.init_trainer(trainer)
with amp.scale_loss(loss, trainer) as scaled_loss:
   autograd.backward(scaled_loss)
```

More details: https://developer.nvidia.com/automatic-mixed-precision

MIXED PRECISION

What is the benefit?

Using mixed precision and Volta your networks can be:

1. 3-4x faster
2. Reduce memory consumption and bandwidth pressure
3. just as powerful

with no architecture change.

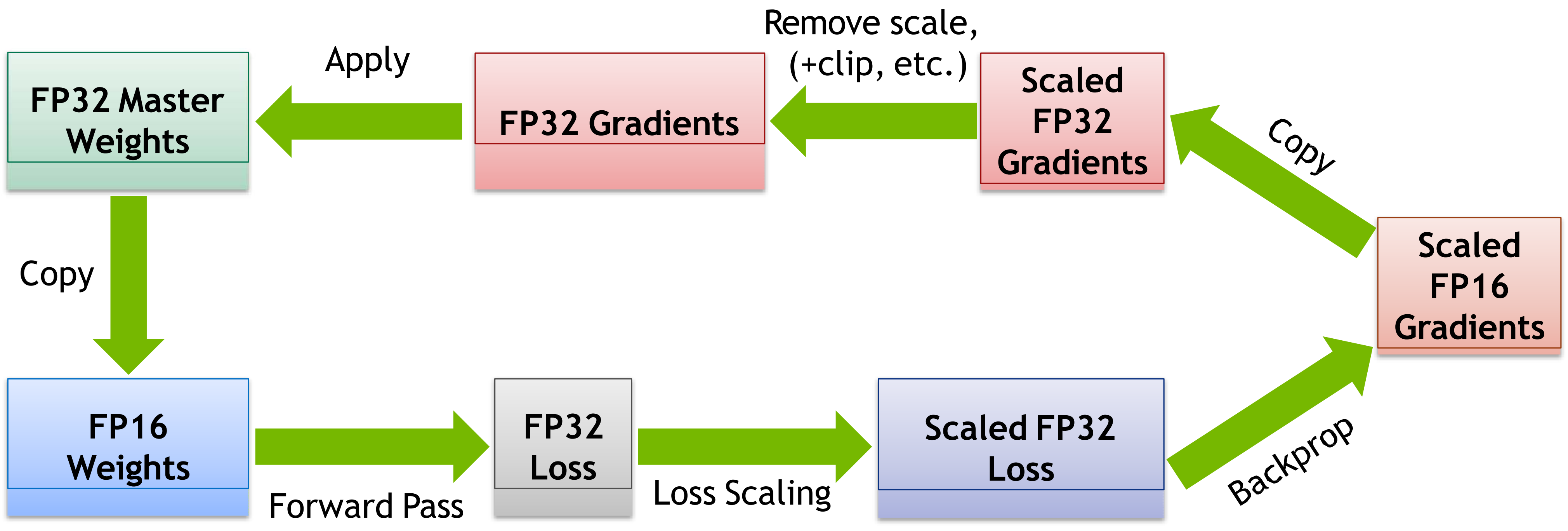
A MIXED PRECISION SOLUTION

Imprecise weight updates $\longrightarrow$ "Master" weights in FP32

Gradients underflow $\longrightarrow$ Loss (Gradient) Scaling

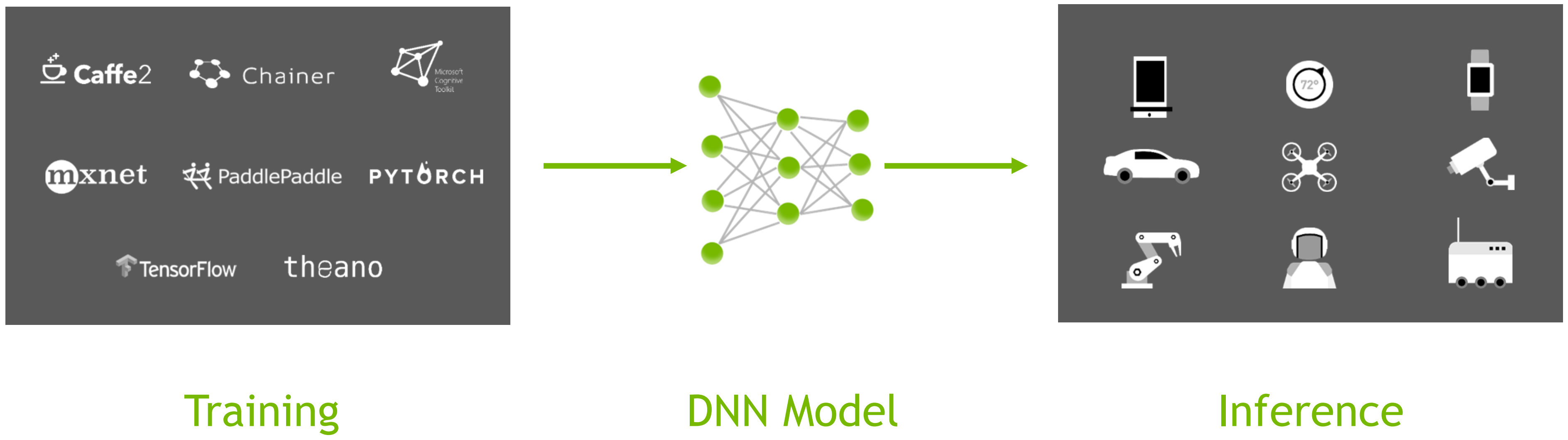
Maintain precision $\longrightarrow$ Accumulate to FP32 (Tensor Cores)

MIXED PRECISION TRAINING



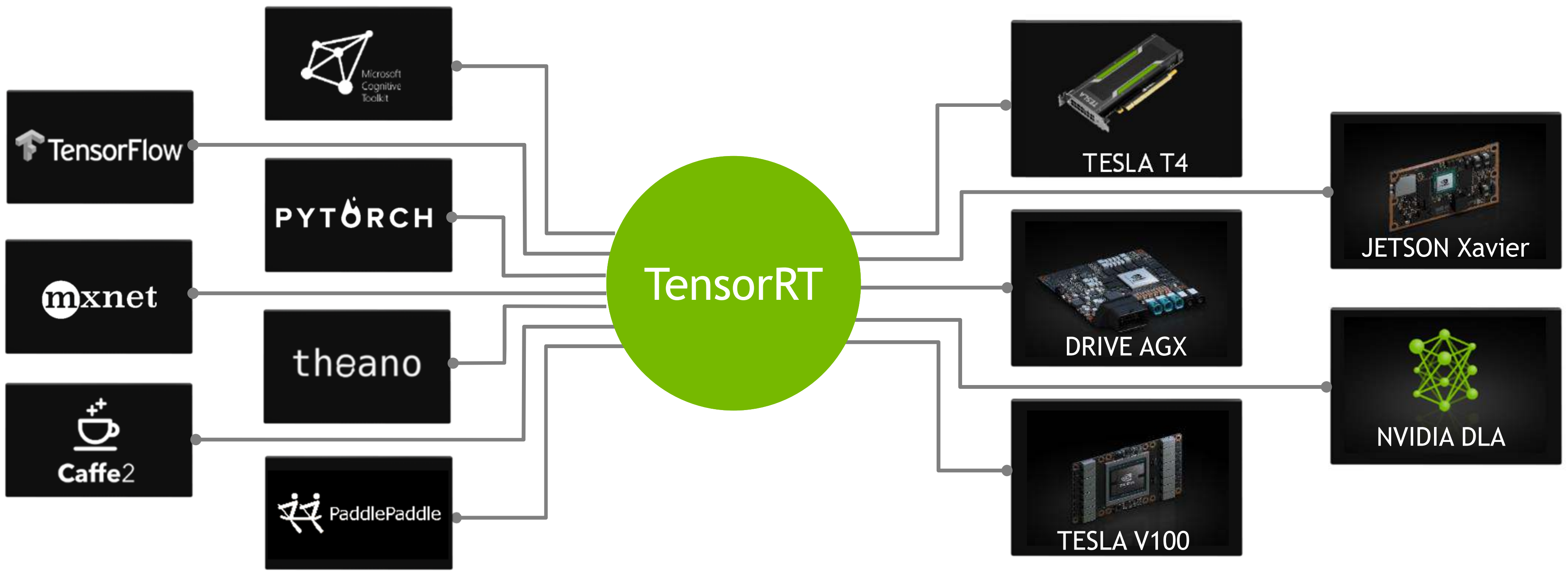
WHY TENSORRT?

AI INFERENCE NEEDS TO RUN EVERYWHERE



NVIDIA TensorRT

From Every Framework, Optimized For Each Target Platform

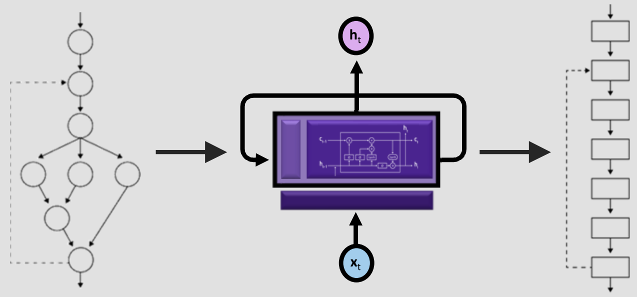


CHALLENGES WITH CURRENT APPROACHES

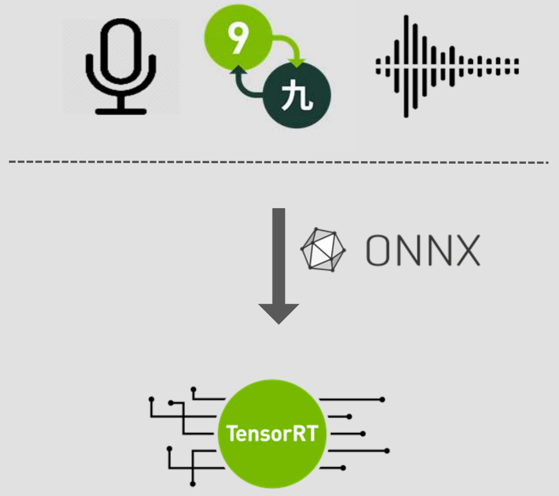
Requirement	Challenges
High Throughput	**Unable to processing high-volume, high-velocity data** ➢ Impact: Increased cost ($, time) per inference
Low Response Time	**Applications don’t deliver real-time results** ➢ Impact: Negatively affects user experience (voice recognition, personalized recommendations, real-time object detection)
Power and Memory Efficiency	**Inefficient applications** ➢ Impact: Increased cost (running and cooling), makes deployment infeasible
Deployment-Grade Solution	**Research frameworks not designed for production** ➢ Impact: Framework overhead and dependencies increases time to solution and affects productivity

ANNOUNCING TensorRT 7

ASR, NLU & TTS | 1000+ Kernels | FP32, FP16, INT8



Compiler Supports RNNs, Transformers and CNNs



20+ ONNX Ops & Dynamic Shapes Enhancements Accelerating Speech

ASR With Jasper Example

NLU With BERT Example

TTS With Tacotron 2+Waveglow Blog & Example

Get Started with ASR, NLU, TTS Today

TensorRT INTEGRATED WITH TENSORFLOW

Speed Up TensorFlow Inference With TensorRT Optimizations

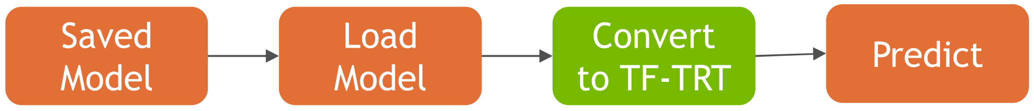
Speed up TensorFlow model inference with TensorRT with new TensorFlow APIs

Simple API to use TensorRT within TensorFlow easily

Sub-graph optimization with fallback offers flexibility of TensorFlow and optimizations of TensorRT

Optimizations for FP32, FP16 and INT8 with use of Tensor Cores automatically

TensorFlow-TensorRT Inference Workflow



```
# Set Precision
conversion_params = trt.DEFAULT_TRT_CONVERSION_PARAMS._replace(
    precision_mode=trt.TrtPrecisionMode.INT8)

# Convert to TF-TRT Graph
converter = trt.TrtGraphConverterV2(
   input_saved_model_dir=input_saved_model_dir,
   conversion_params=conversion_params)

# INT8 Calibration
converter.convert(calibration_input_fn=my_calibration_fn)

# Run Inference
converter.save(output_saved_model_dir)
```

Available in TensorFlow 2.0 and 1.15

https://github.com/tensorflow/tensorflow

TensorRT ONNX PARSER

High-Performance Inference for ONNX Models

Optimize and deploy models from ONNX-supported frameworks to production

Apply TensorRT optimizations to any ONNX framework (Caffe 2, Microsoft Cognitive Toolkit, MxNet & PyTorch)

Import TensorFlow and Keras through converters (tf2onnx, keras2onnx)

Use with C++ and Python apps

20+ New Ops in TensorRT 7

Support for Opset 11 (See List of Supported Ops)

ONNX

Caffe2 Chainer Cognitive Toolkit

mxnet MATLAB® PyTorch

PaddlePaddle

TensorRT Optimization

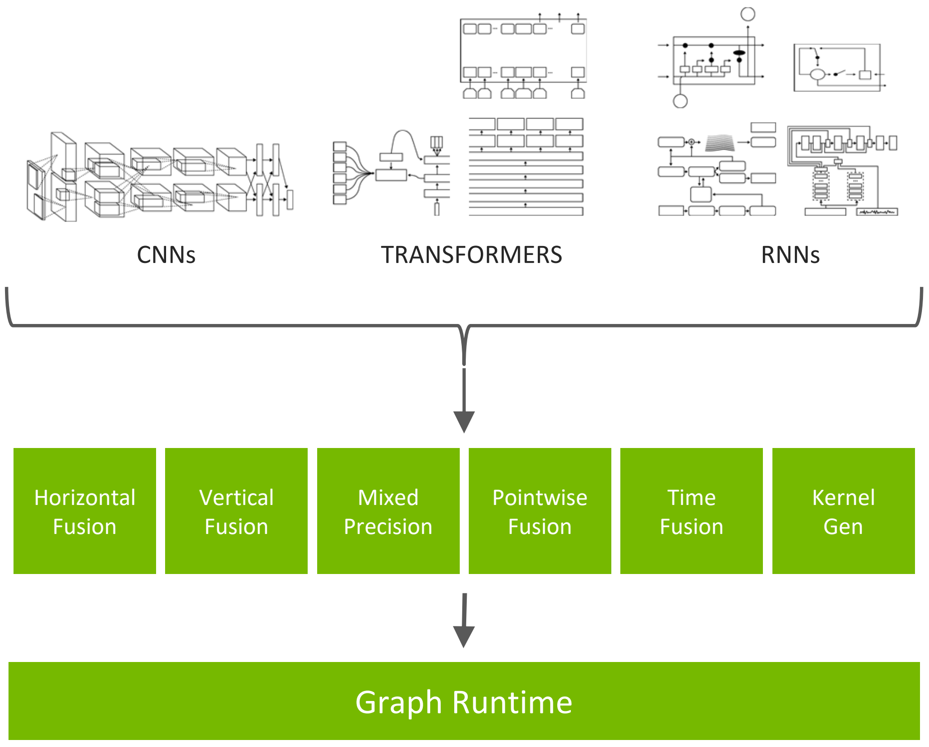
Deploy highly-optimized Conversational AI apps in production environments

New API to define loops found in RNNs

Compiler fuses pointwise ops, generates optimized kernels, and fuses ops across time steps

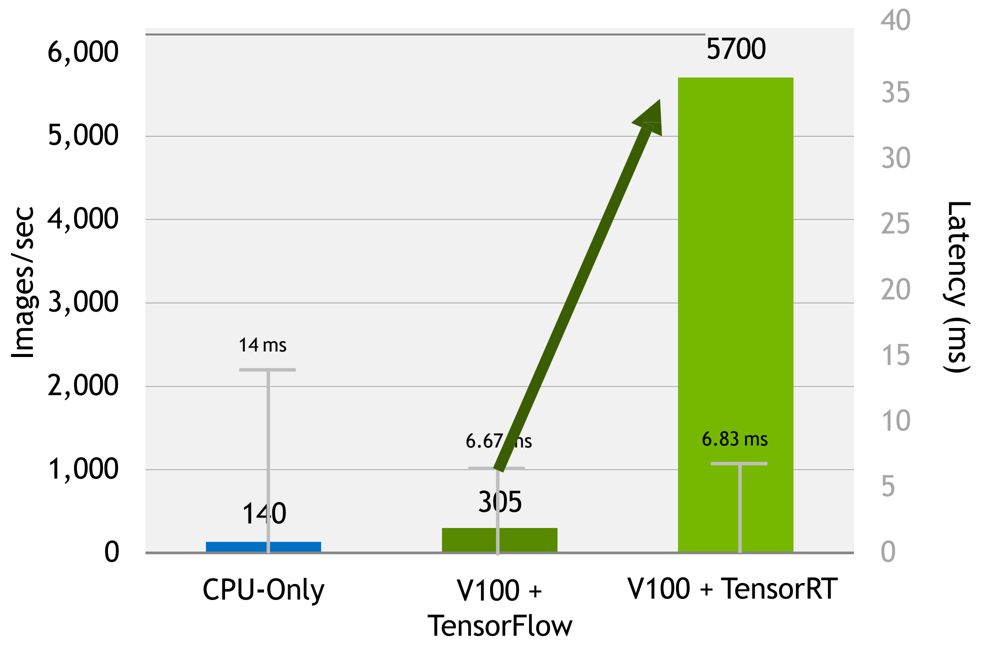
Run ASR, NLU and TTS within 300 ms, a requirement for real time apps, 10x perf vs CPU

Models Supported: BERT, MT-DNN, RoBERTa, Tacotron 2, WaveRNN, DeepASR, GNMT, LSTM Peephole, LSTM Autoencoder



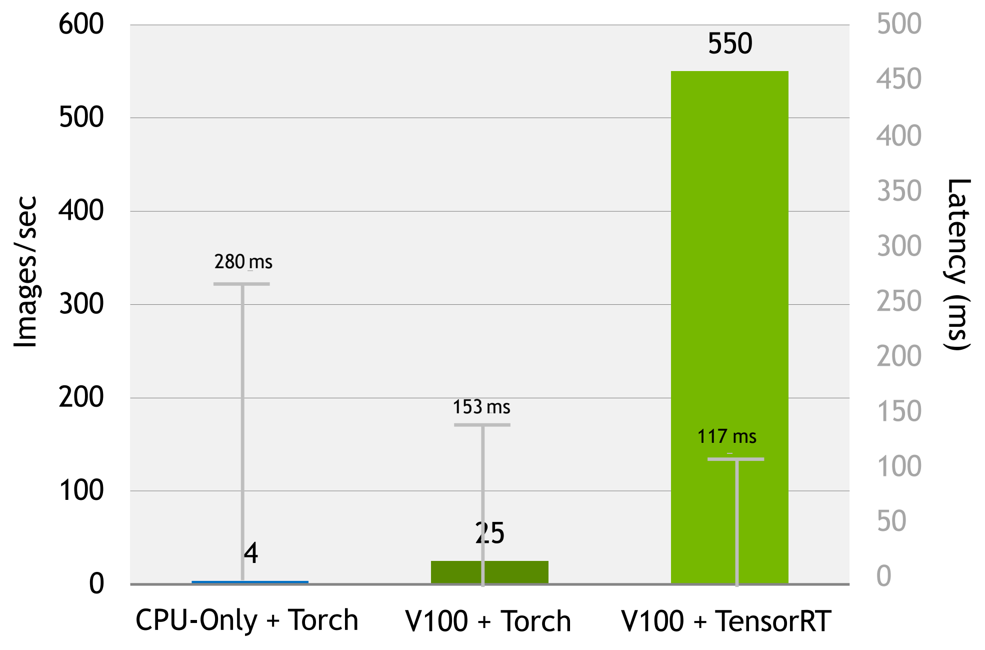
TENSORRT PERFORMANCE

40x Faster CNNs on V100 vs. CPU-Only Under 7ms Latency (ResNet50)



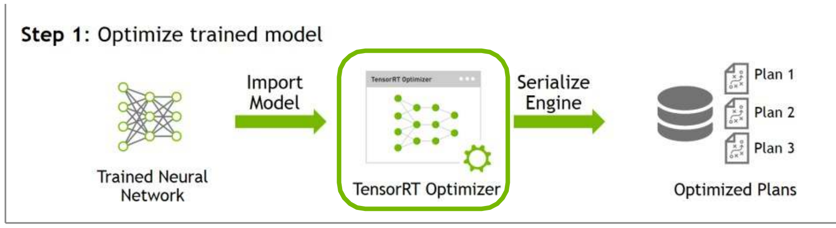
Inference throughput (images/sec) on ResNet50. **V100 + TensorRT**: NVIDIA TensorRT (FP16), batch size 39, Tesla V100-SXM2-16GB, E5-2690 v4@2.60GHz 3.5GHz Turbo (Broadwell) HT On **V100 + TensorFlow**: Preview of volta optimized TensorFlow (FP16), batch size 2, Tesla V100-PCIE-16GB, E5-2690 v4@2.60GHz 3.5GHz Turbo (Broadwell) HT On. **CPU-Only:** Intel Xeon-D 1587 Broadwell-E CPU and Intel DL SDK. Score doubled to comprehend Intel's stated claim of 2x performance improvement on Skylake with AVX512.

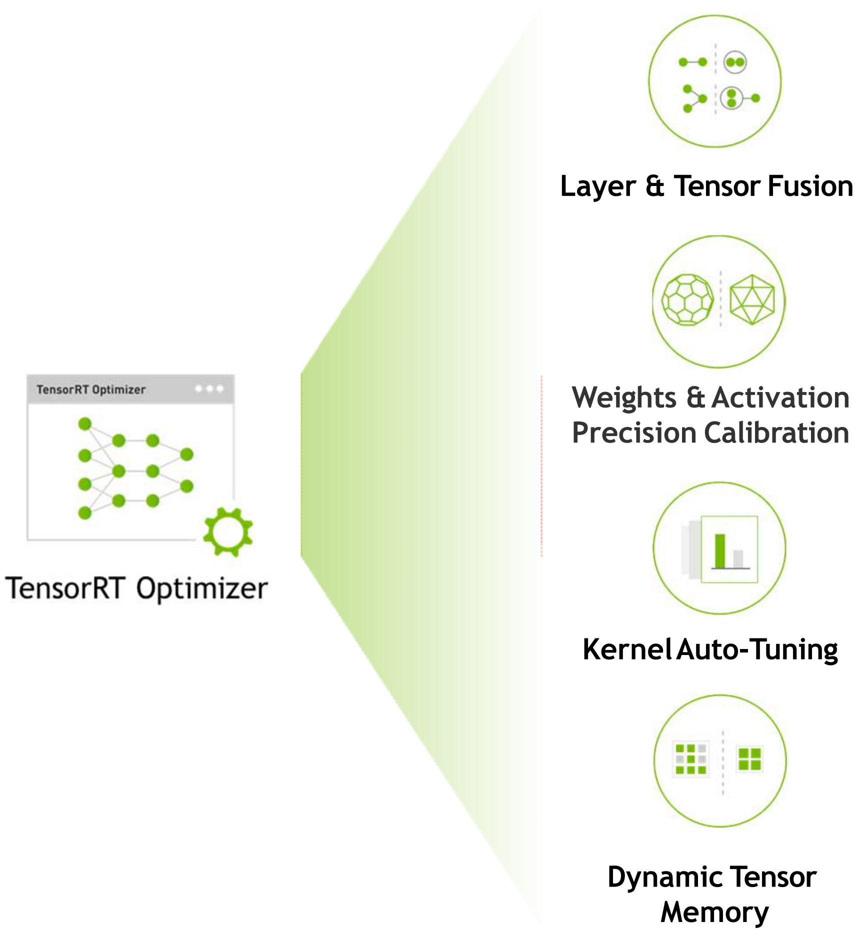
140x Faster Language Translation RNNs on V100 vs. CPU-Only Inference (OpenNMT)



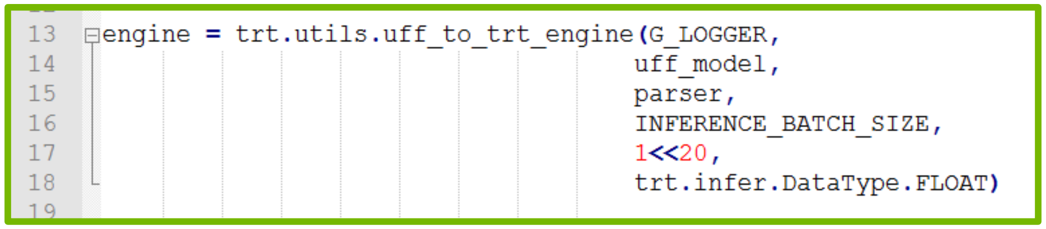
Inference throughput (sentences/sec) on OpenNMT 692M. **V100 + TensorRT**: NVIDIA TensorRT (FP32), batch size 64, Tesla V100-PCIE-16GB, E5-2690 v4@2.60GHz 3.5GHz Turbo (Broadwell) HT On. **V100 + Torch**: Torch (FP32), batch size 4, Tesla V100-PCIE-16GB, E5-2690 v4@2.60GHz 3.5GHz Turbo (Broadwell) HT On. **CPU-Only:** Torch (FP32), batch size 1, Intel E5-2690 v4@2.60GHz 3.5GHz Turbo (Broadwell) HT On

TENSORRT OPTIMIZATIONS



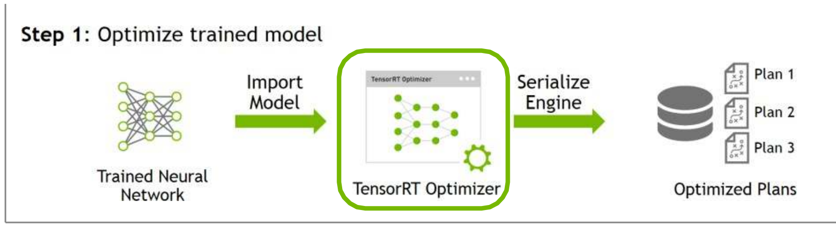


- ➢ Optimizations are completely automatic
- ➢ Performed with a single function call

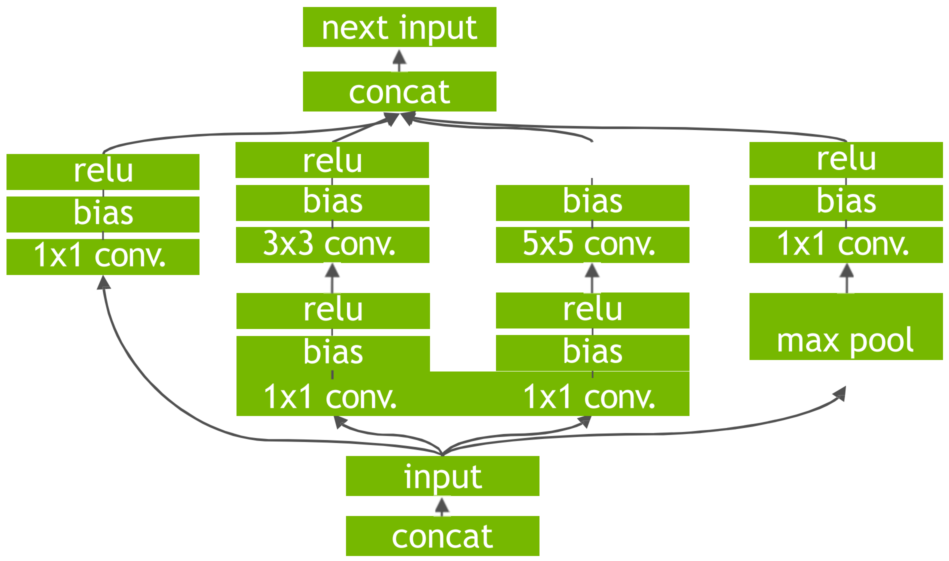


```
engine = trt.utils.uff_to_trt_engine(G_LOGGER,
                                     uff_model,
                                     parser,
                                     INFERENCE_BATCH_SIZE,
                                     1<<20,
                                     trt.infer.DataType.FLOAT)
```

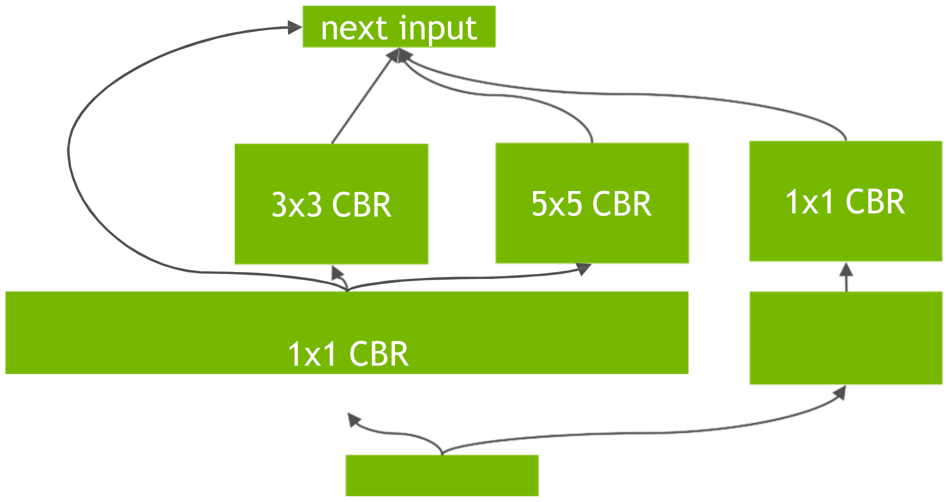
LAYER & TENSOR FUSION



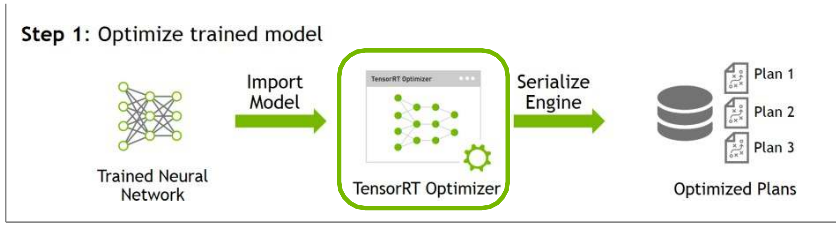
Un-Optimized Network



TensorRT Optimized Network



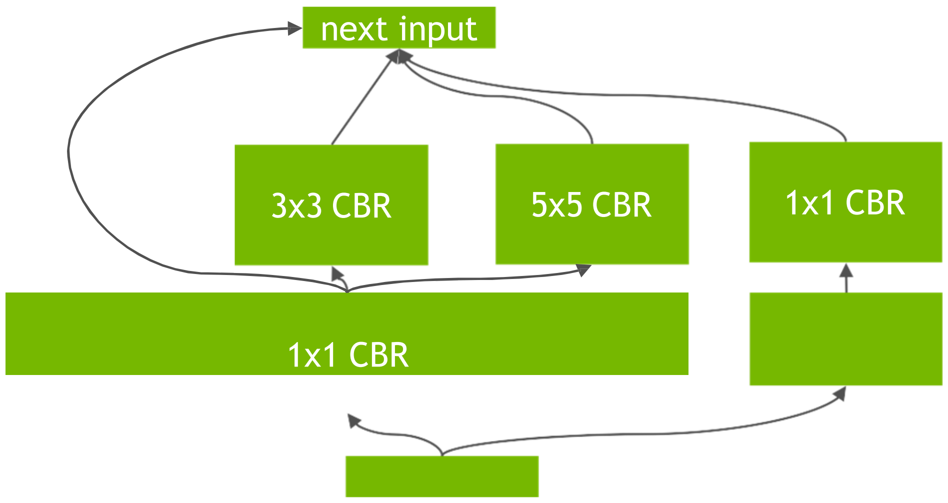
LAYER & TENSOR FUSION



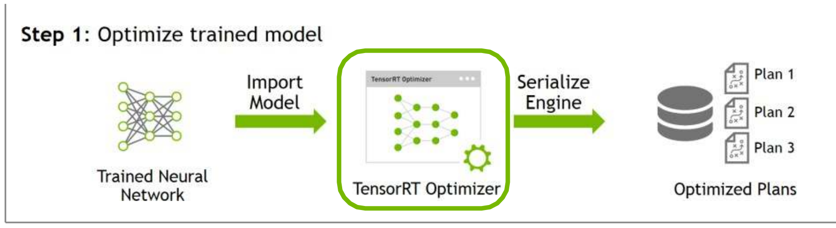
- Vertical Fusion
- Horizonal Fusion
- Layer Elimination

Network	Layers before	Layers after
VGG19	43	27
Inception V3	309	113
ResNet-152	670	159

TensorRT Optimized Network



FP16, INT8 PRECISION CALIBRATION

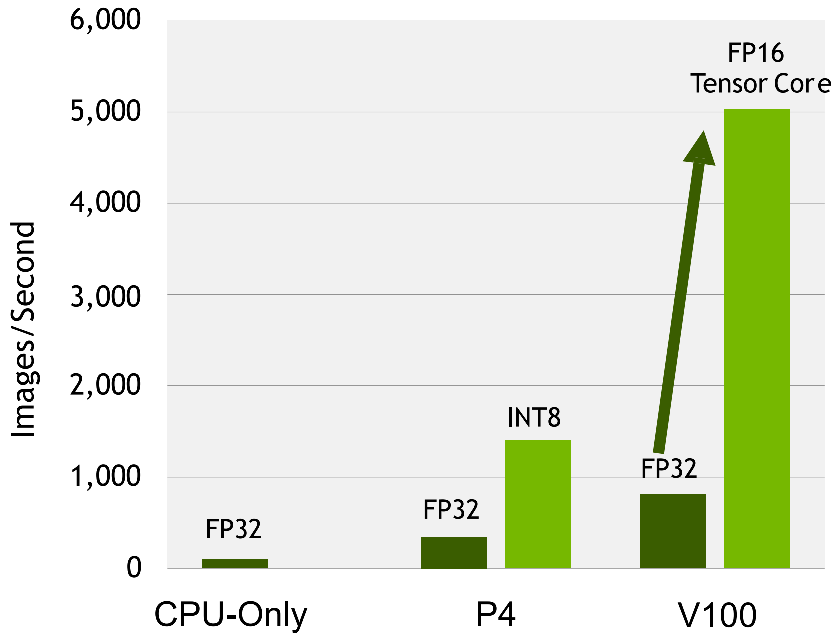


Precision	Dynamic Range	
FP32	-3.4×10^{38} ~ $+3.4 \times 10^{38}$	Training precision
FP16	-65504 ~ +65504	No calibration required
INT8	-128 ~ +127	Requires calibration

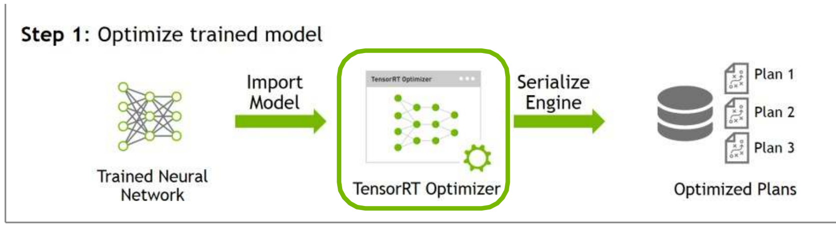
Precision calibration for INT8 inference:

- ➢ Minimizes information loss between FP32 and INT8 inference on a calibration dataset
- ➢ Completely automatic

Reduced Precision Inference Performance (ResNet50)



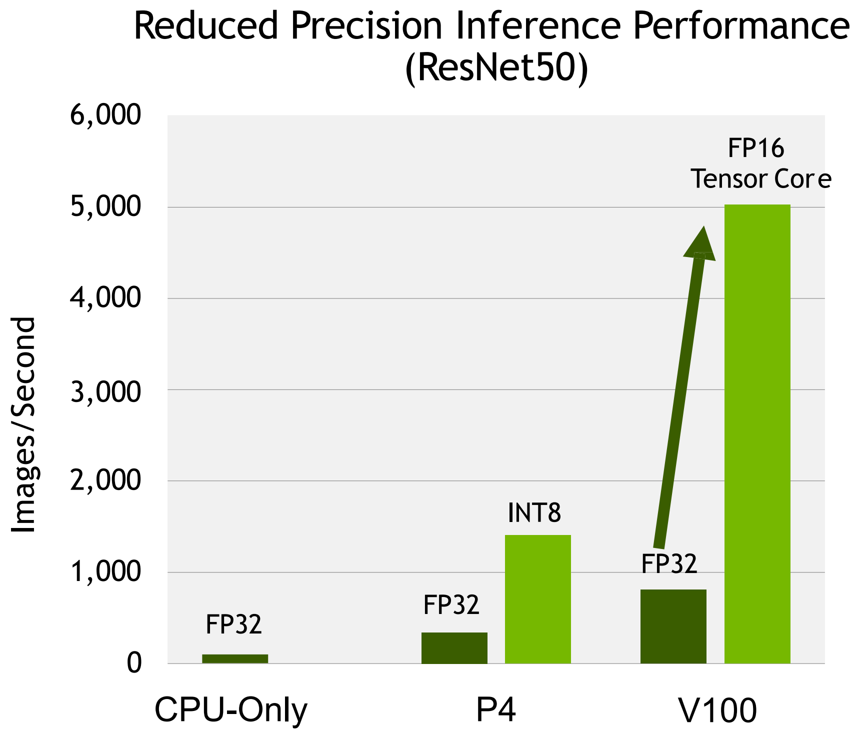
FP16, INT8 PRECISION CALIBRATION



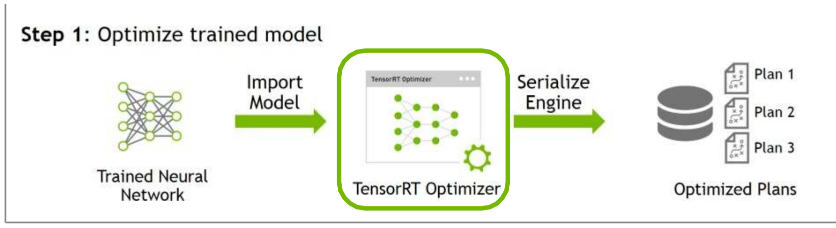
	FP32 Top 1	INT8 Top 1	Difference
Googlenet	68.87%	68.49%	0.38%
VGG	68.56%	68.45%	0.11%
Resnet-50	73.11%	72.54%	0.57%
Resnet-152	75.18%	74.56%	0.61%

Precision calibration for INT8 inference:

- Minimizes information loss between FP32 and INT8 inference on a calibration dataset
- Completely automatic



KERNEL AUTO-TUNING
DYNAMIC TENSOR MEMORY



Kernel Auto-Tuning

- 100's of custom built kernel tuning based on target GPU Architecture.

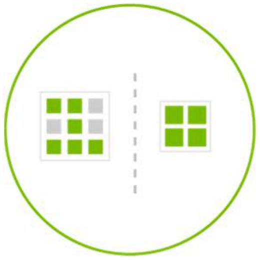
Tesla V100

Jetson TX2

Drive PX2

Multiple parameters:
- Batch size
- Input dimensions
- Filter dimensions

…



Dynamic Tensor Memory

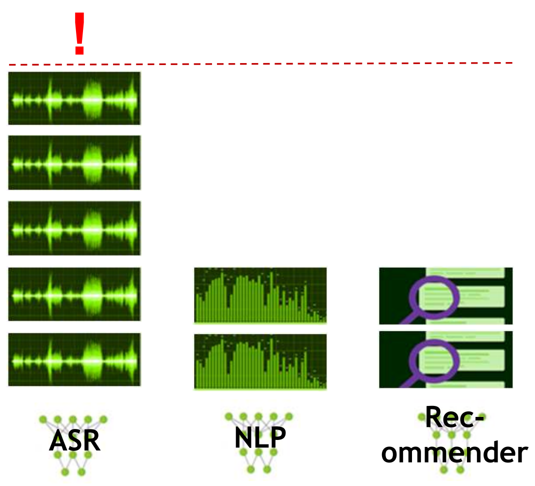
- Reduces memory footprint and improves memory re-use
- Manages memory allocation for each tensor only for the duration of its usage

WHY TENSORRT INFERENCE SERVER?

INEFFICIENCY LIMITS INNOVATION

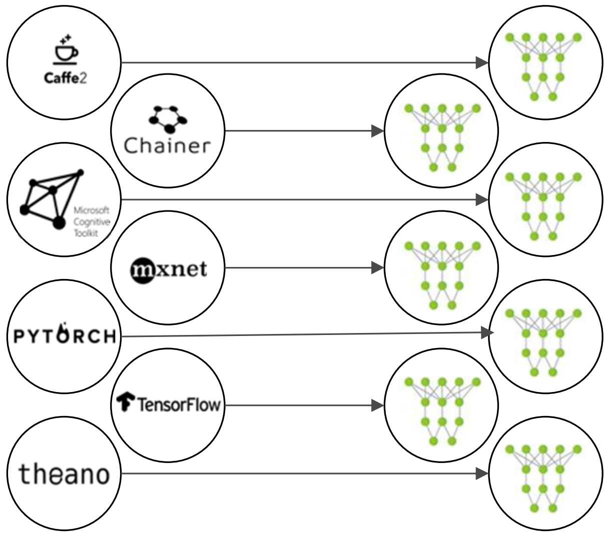
Difficulties with Deploying Data Center Inference

Single Model Only



Some systems are overused while others are underutilized

Single Framework Only



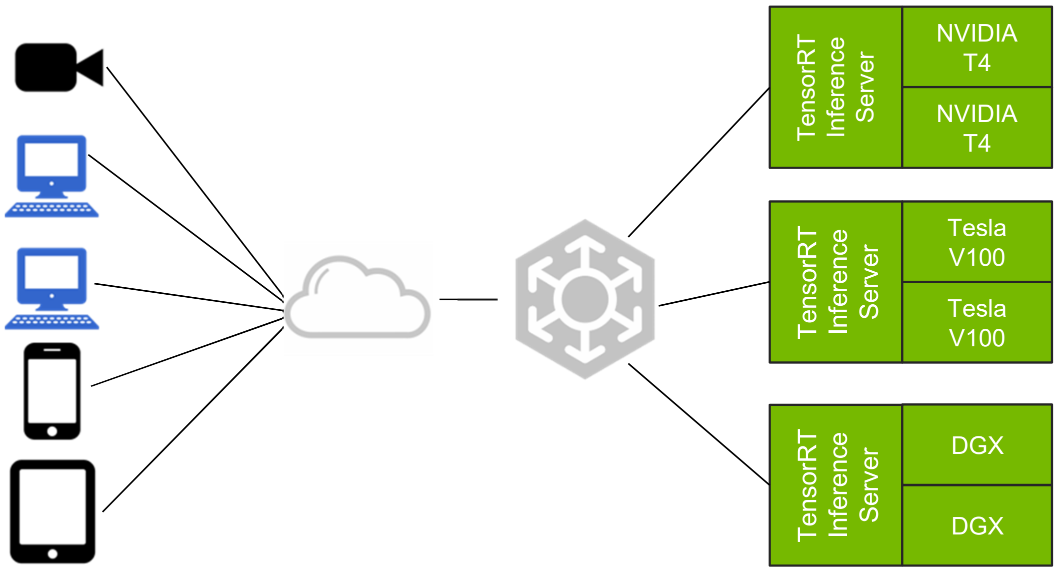
Solutions can only support models from one framework

Custom Development

Developers need to reinvent the plumbing for every application

NVIDIA TENSORRT INFERENCE SERVER

Production Data Center Inference Server


TensorRT Inference Server
NVIDIA T4
NVIDIA T4
TensorRT Inference Server
Tesla V100
Tesla V100
TensorRT Inference Server
DGX
DGX

Maximize real-time inference performance of GPUs

Quickly deploy and manage multiple models per GPU per node

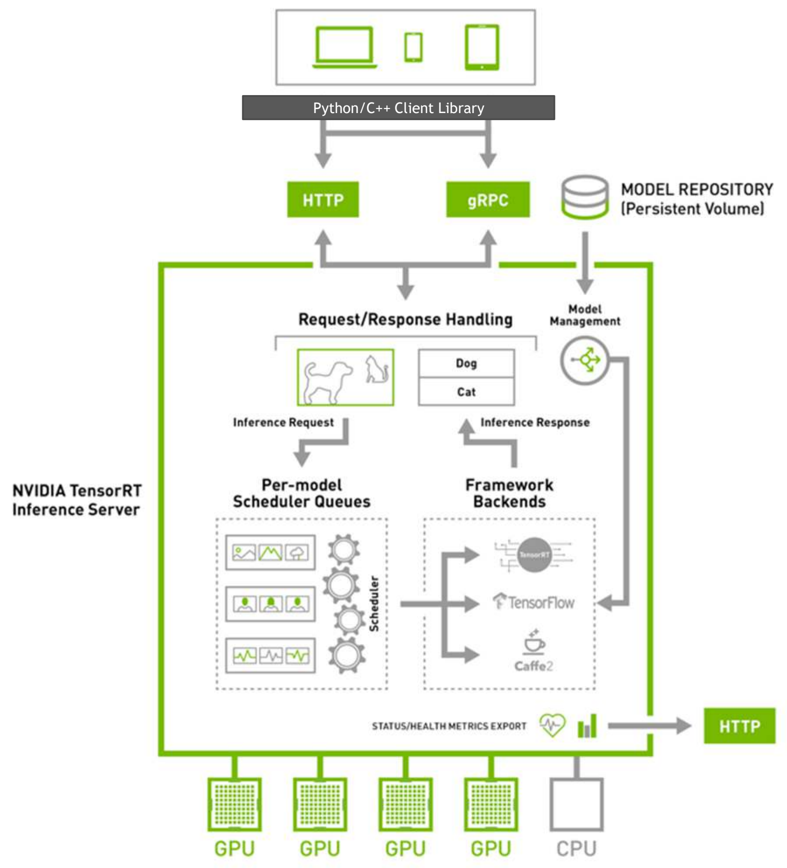
Easily scale to heterogeneous GPUs and multi GPU nodes

Integrates with orchestration systems and auto scalers via latency and health metrics

Now open source for thorough customization and integration

INFERENCE SERVER ARCHITECTURE

Available with Monthly Updates



Python/C++ Client Library
HTTP
gRPC
MODEL REPOSITORY (Persistent Volume)
Request/Response Handling
Model Management
Dog
Cat
Inference Request
Inference Response
NVIDIA TensorRT Inference Server
Per-model Scheduler Queues
Framework Backends
Scheduler
TensorFlow
Caffe2
STATUS/HEALTH METRICS EXPORT
HTTP
GPU
GPU
GPU
GPU
CPU

Models supported
- TensorFlow GraphDef/SavedModel
- TensorFlow and TensorRT GraphDef
- TensorRT Plans
- Caffe2 NetDef (ONNX import)
- ONNX graph
- PyTorch JIT (.pb)

Multi-GPU support

Concurrent model execution

Server HTTP REST API/gRPC

Python/C++ client libraries

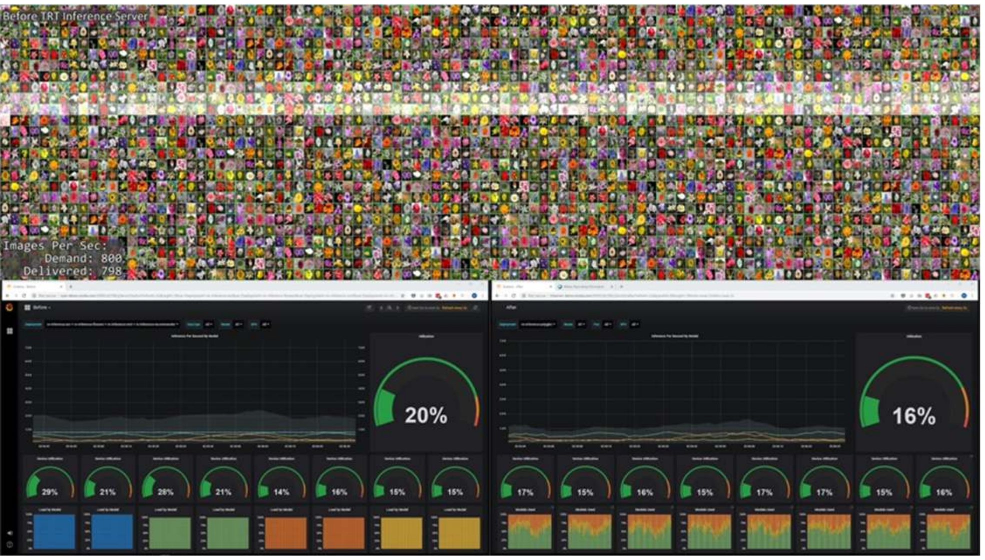
TENSORRT INFERENCE SERVER OVERVIEW

A typical TensorRT Inference Server pipeline can be broken down into the following 8 steps:

1. Client serializes the inference request into a message and sends it to the server (Client Send)
2. Message travels over the network from the client to the server (Network)
3. Message arrives at server, and is deserialized (Server Receive)
4. Request is placed on the queue (Server Queue)
5. Request is removed from the queue and computed (Server Compute)
6. Completed request is serialized in a message and sent back to the client (Server Send)
7. Completed message travels over network from the server to the client (Network)
8. Completed message is deserialized by the client and processed as a completed inference request (Client Receive)

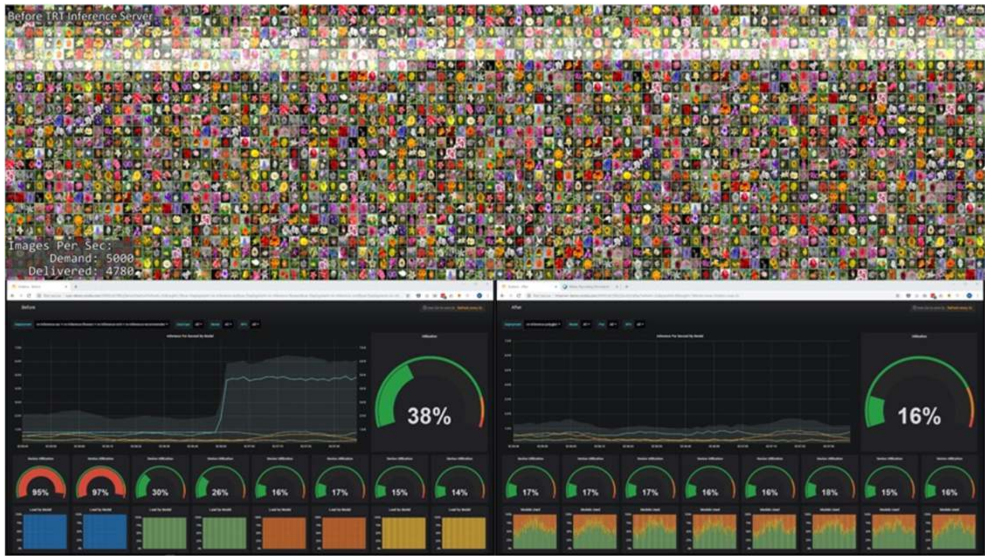
TENSORRT INFERENCE SERVER METRICS FOR AUTOSCALING

Before TensorRT Inference Server - 800 FPS



- One model per GPU
- Requests are steady across all models
- Utilization is low on all GPUs

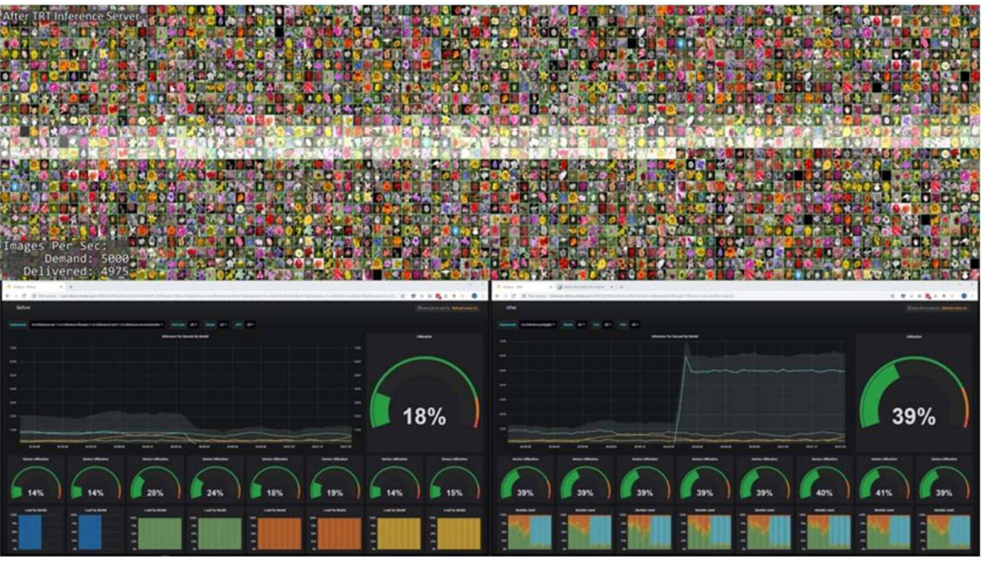
Before TensorRT Inference Server - 5,000 FPS



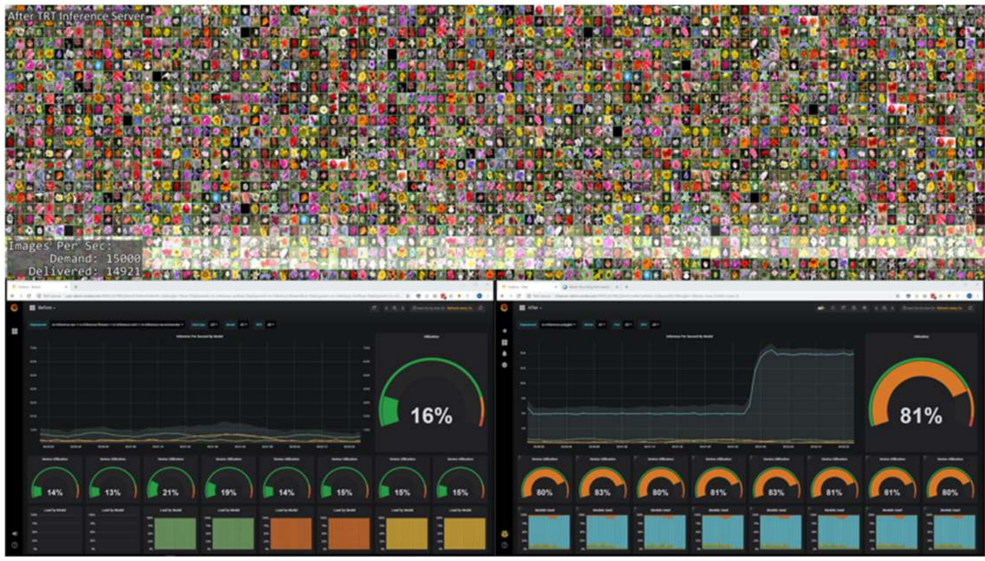
- Spike in requests for blue model
- GPUs running blue model are being fully utilized
- Other GPUs remain underutilized

TENSORRT INFERENCE SERVER METRICS FOR AUTOSCALING

After TensorRT Inference Server - 5,000 FPS



After TensorRT Inference Server - 15,000 FPS



- Load multiple models on every GPU
- Load is evenly distributed between all GPUs

- Spike in requests for blue model
- Each GPU can run the blue model concurrently
- Metrics to indicate time to scale up
  - GPU utilization
  - Power usage
  - Inference count
  - Queue time
  - Number of requests/sec