ACCELERATING END TO END DEEP LEARNING WORKFLOW.

Deepshikha Kumari

Data Scientist II- Deep learning

AGENDA

1. AI Use cases for Industry
2. End To End Deep Learning Workflow

   Training Pipeline
   - a. NGC
   - b. Transfer Learning
   - c. Automatic Mixed Precision
   - d. Code walkthrough

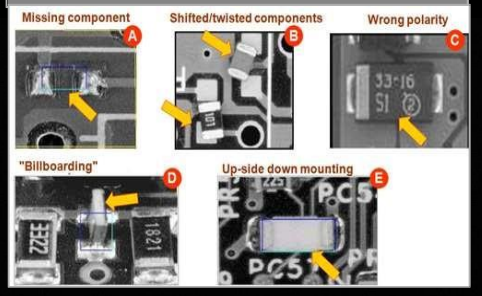
   Inference Pipeline
   - a. TensorRT (Float 16)
   - b. TensorRT (INT8)
   - c. Custom plugin support
   - d. Deepstream

INTELLIGENT VIDEO ANALYTICS (IVA) FOR EFFICIENCY AND SAFETY

Access Control

Public Transit

Industrial Inspection



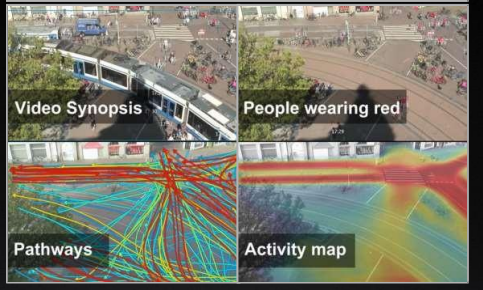
Traffic Engineering

Retail Analytics

Logistics

Critical Infrastructure

Public Safety



NVIDIA

DEEP LEARNING IN PRODUCTION

Speech Recognition

Recommender Systems

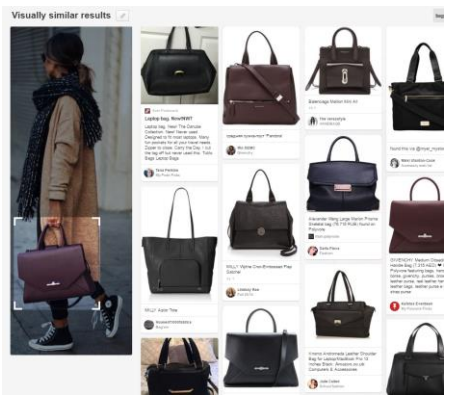
Autonomous Driving

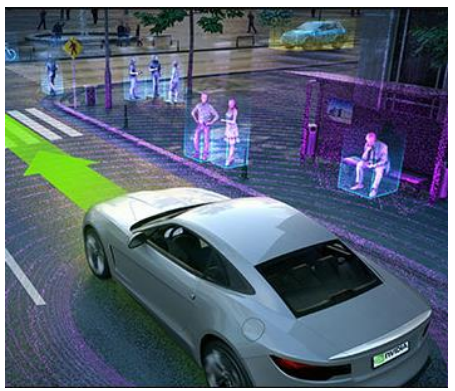
Real-time Object Recognition

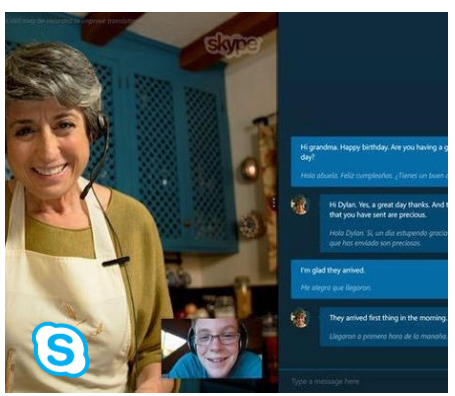
Robotics

Real-time Language Translation

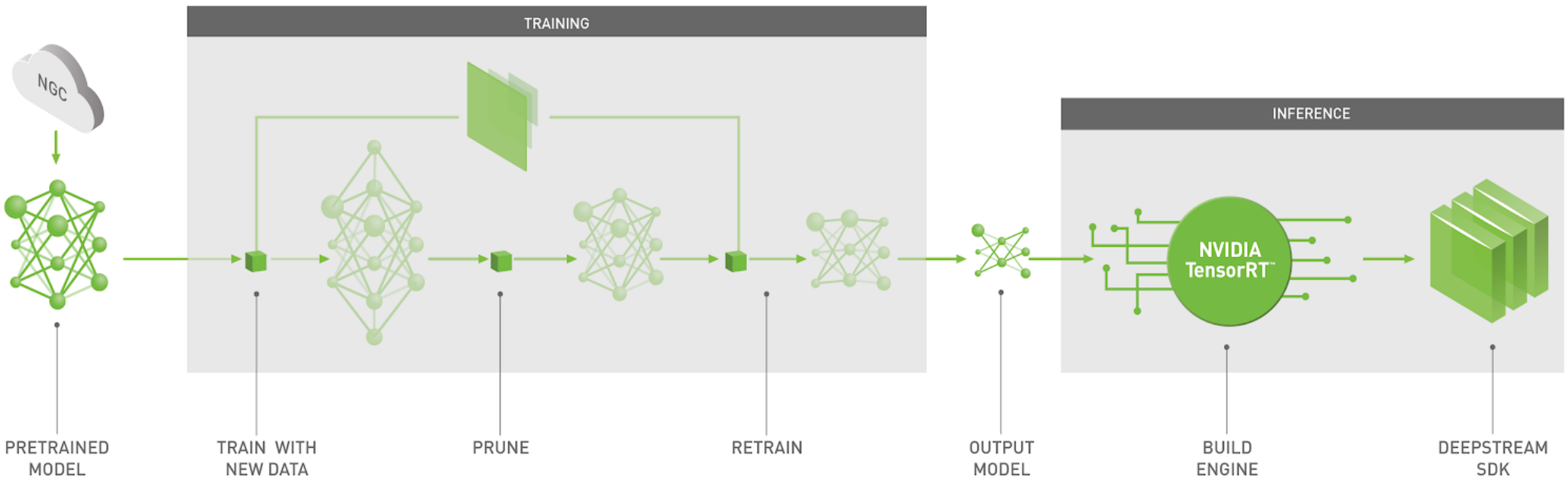
Many More…



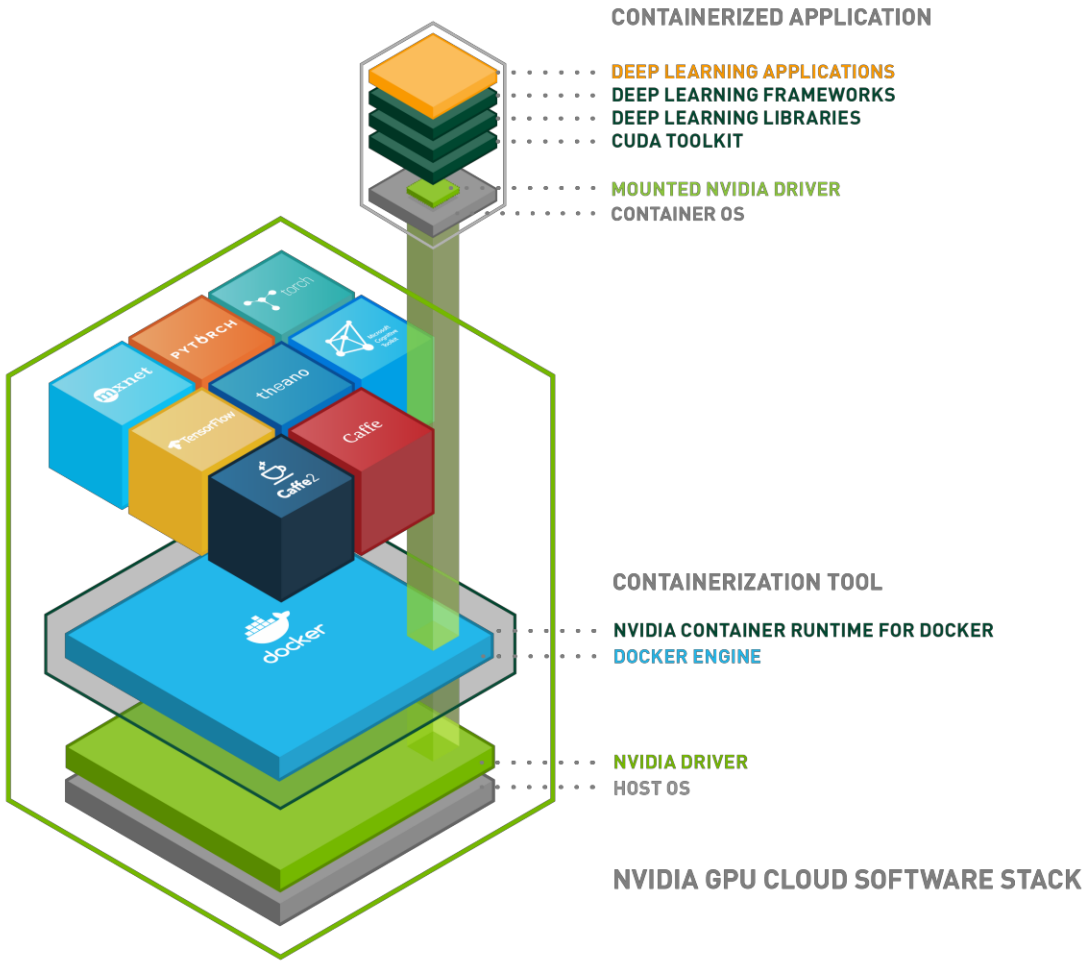




END TO END NVIDIA DEEP LEARNING WORKFLOW



NGC



WHY CONTAINERS?

Benefits of Containers:

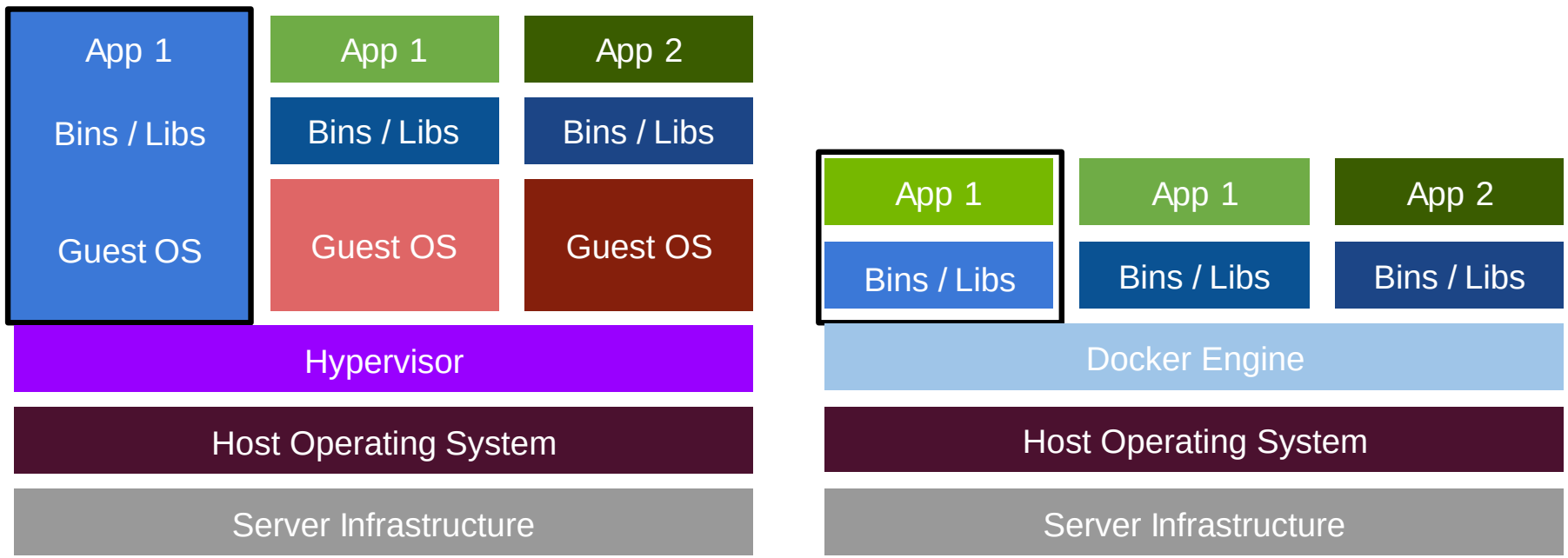
Simplify deployment of GPU-accelerated software, eliminating time-consuming software integration work

Isolate individual deep learning frameworks and applications

Share, collaborate, and test applications across different environments

Virtual Machine vs. Container

Not so similar



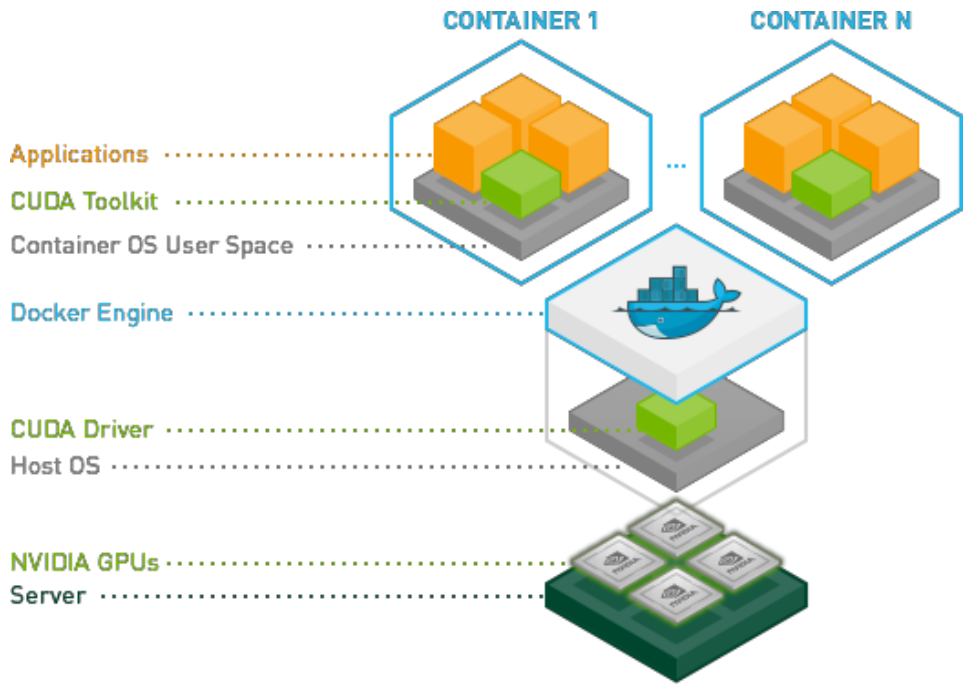
Virtual Machines

Containers

NVIDIA container runtime

https://github.com/NVIDIA/nvidia-docker

- Colloqually called “nvidia-docker”
- Docker containers are hardware-agnostic and platform-agnostic
- NVIDIA GPUs are specialized hardware that require the NVIDIA driver
- Docker does not natively support NVIDIA GPUs with containers
- NVIDIA Container Runtime makes the images agnostic of the NVIDIA driver



Docker Terms

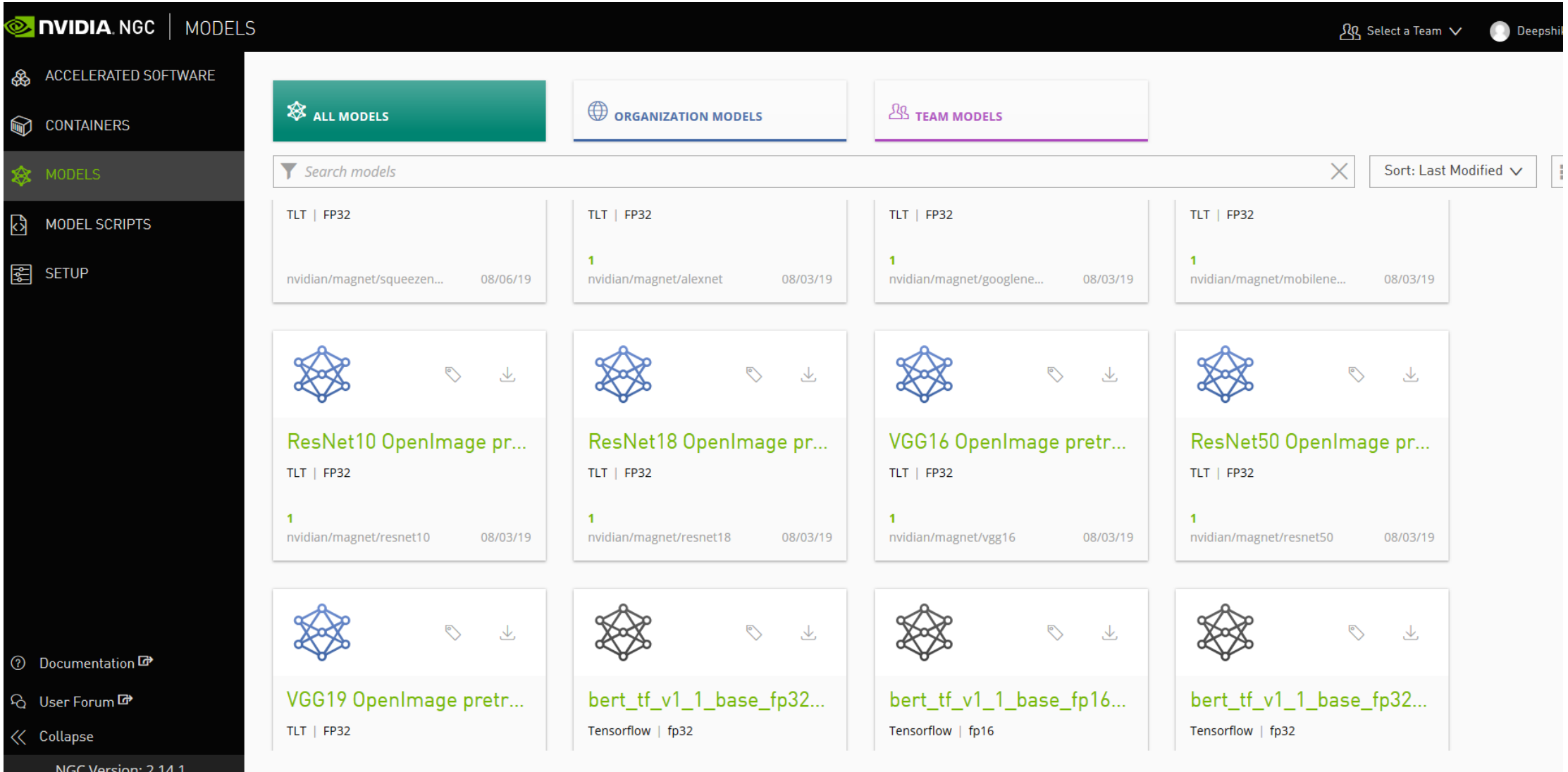
Definitions

Image
Docker images are the basis of containers. An Image is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. An image typically contains a union of layered filesystems stacked on top of each other. An image does not have state and it never changes.

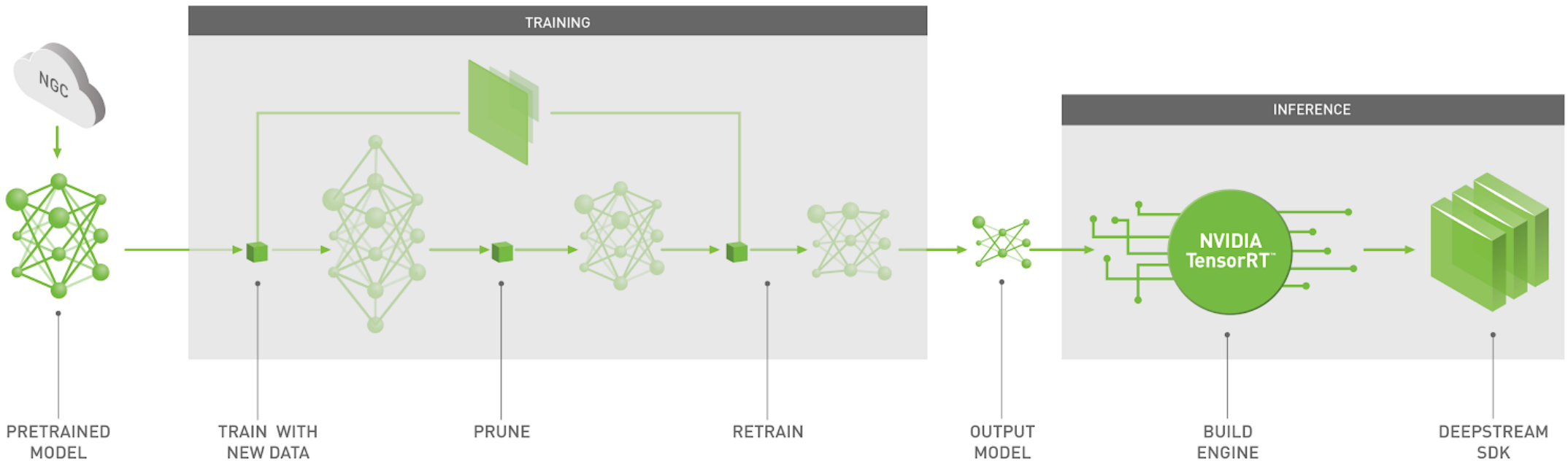
Container
A container is a runtime instance of a docker image.
A Docker container consists of

- A Docker image
- Execution environment
- A standard set of instructions

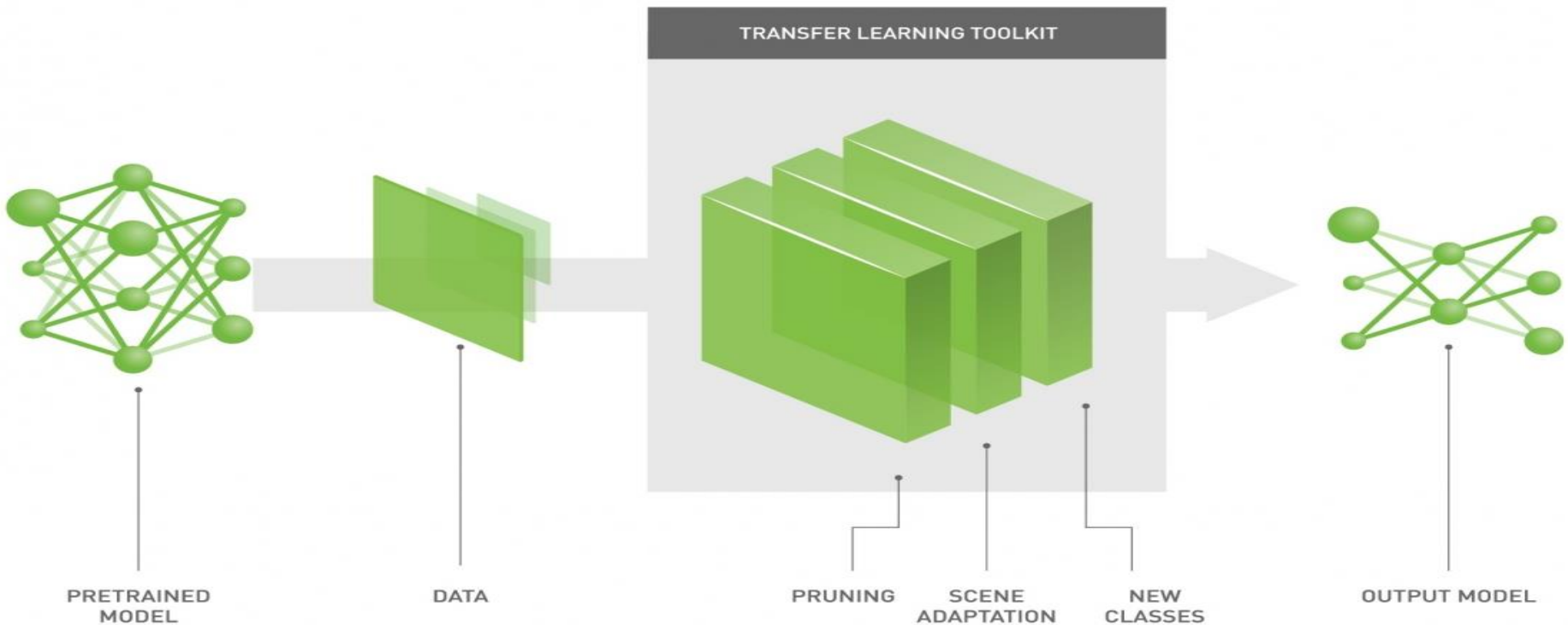
https://docs.docker.com/engine/reference/glossary/


NVIDIA NGC | MODELS
Select a Team
Deepshi
ACCELERATED SOFTWARE
CONTAINERS
MODELS
MODEL SCRIPTS
SETUP
Documentation
User Forum
Collapse
NGC Version: 2.14.1
ALL MODELS
ORGANIZATION MODELS
TEAM MODELS
Search models
Sort: Last Modified
TLT | FP32
nvidian/magnet/squeezen... 08/06/19
TLT | FP32
1
nvidian/magnet/alexnet 08/03/19
TLT | FP32
1
nvidian/magnet/googlene... 08/03/19
TLT | FP32
1
nvidian/magnet/mobilene... 08/03/19
ResNet10 OpenImage pr...
TLT | FP32
1
nvidian/magnet/resnet10 08/03/19
ResNet18 OpenImage pr...
TLT | FP32
1
nvidian/magnet/resnet18 08/03/19
VGG16 OpenImage pretr...
TLT | FP32
1
nvidian/magnet/vgg16 08/03/19
ResNet50 OpenImage pr...
TLT | FP32
1
nvidian/magnet/resnet50 08/03/19
VGG19 OpenImage pretr...
TLT | FP32
bert_tf_v1_1_base_fp32...
Tensorflow | fp32
bert_tf_v1_1_base_fp16...
Tensorflow | fp16
bert_tf_v1_1_base_fp32...
Tensorflow | fp32

END TO END NVIDIA DEEP LEARNING WORKFLOW

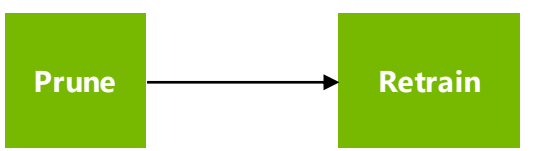


TRANSFER LEARNING TOOLKIT

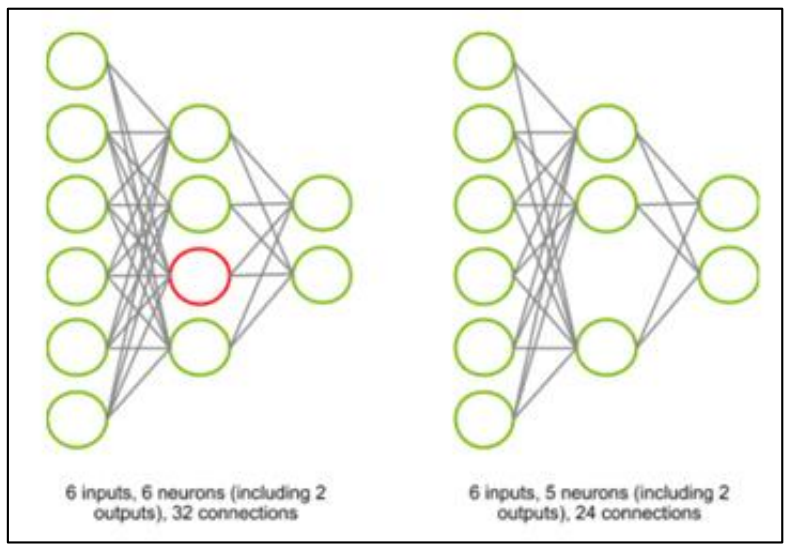


PRUNING

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



tlt-prune *tlt-train*



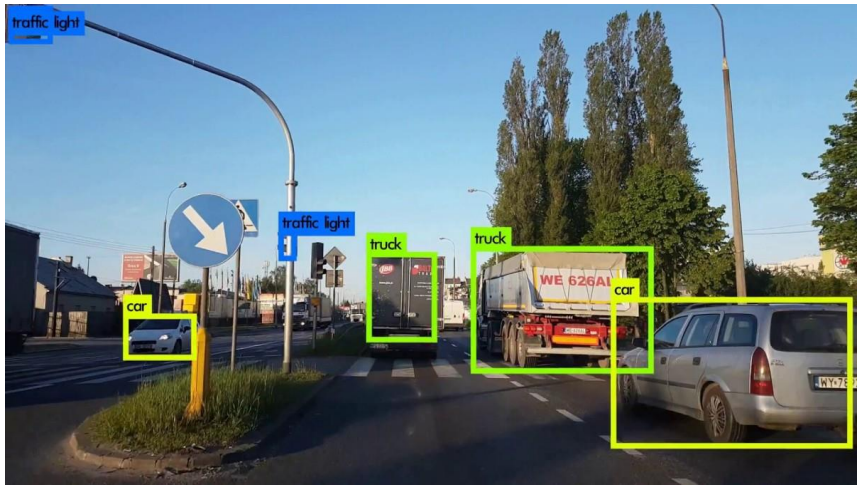
Selecting Unnecessary Neurons

- 1. DATA Driven operation
- 2. Non- Data Driven Operation.
- 3. Handling Element-Wise Operations of Multiple Inputs

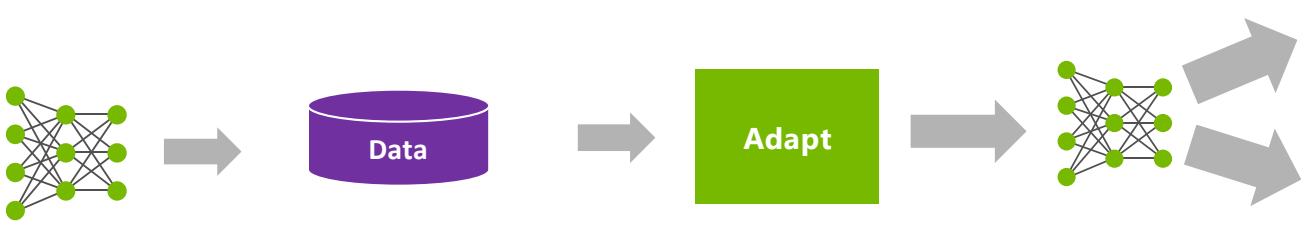
```
pruned_model = TLT.prune(model, t)
```

SCENE ADAPTATION

Camera location vantage point

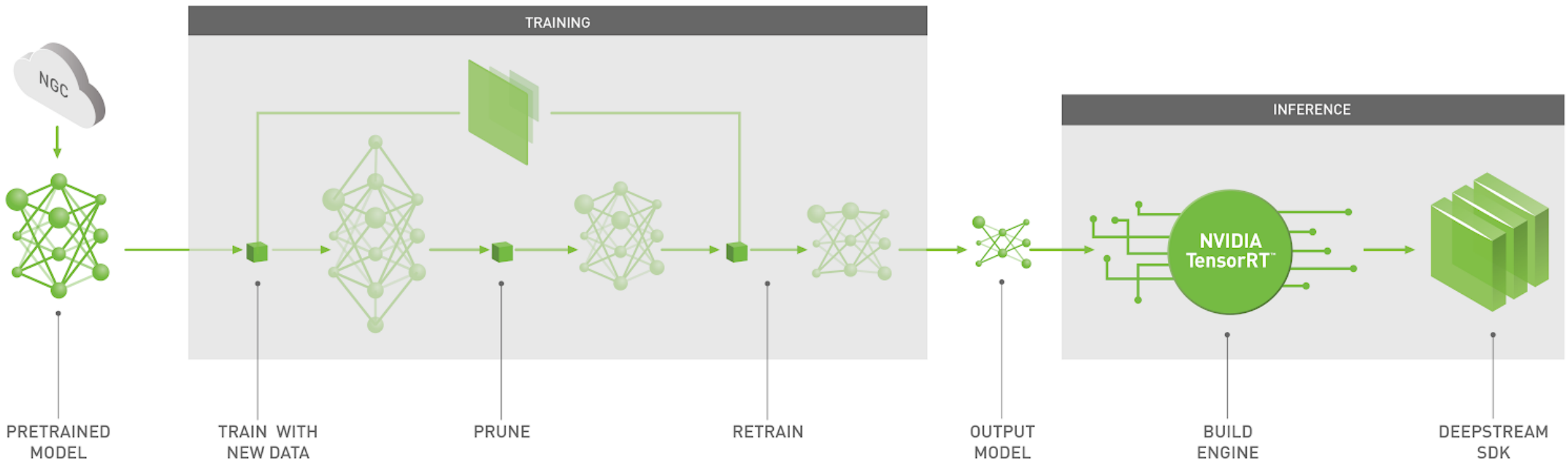


Person with blue shirt

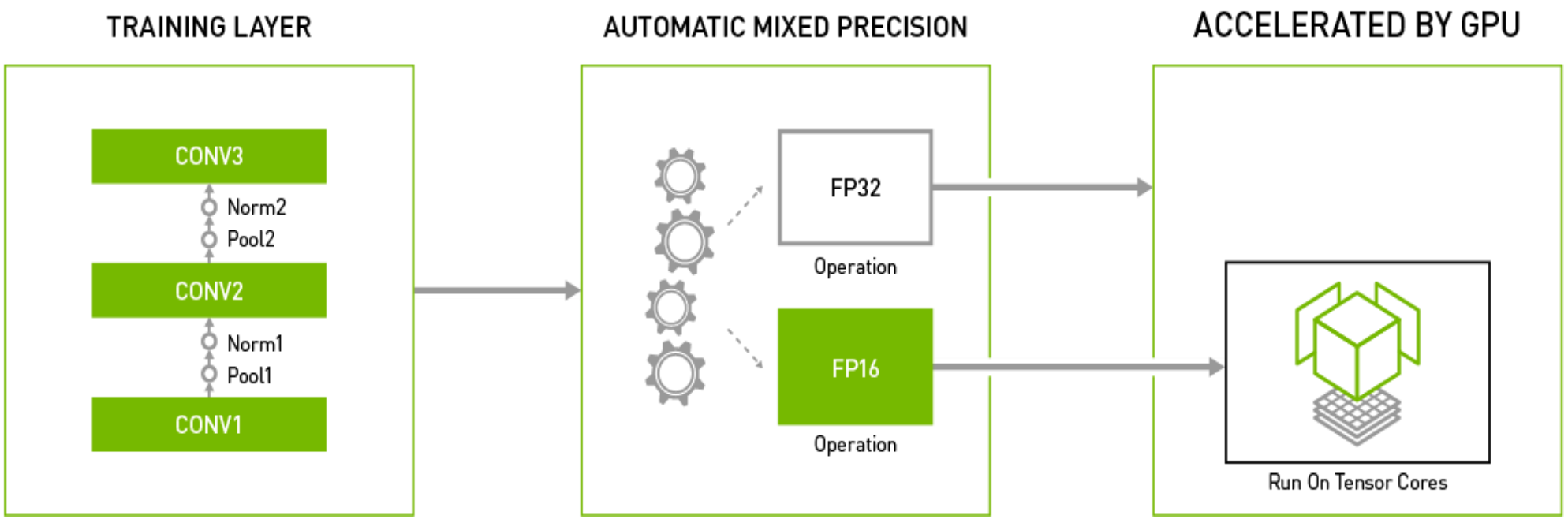


Train with new data from another vantage point, camera location, or added attribute

END TO END NVIDIA DEEP LEARNING WORKFLOW



TLT


TRAINING LAYER
CONV3
Norm2
Pool2
CONV2
Norm1
Pool1
CONV1
AUTOMATIC MIXED PRECISION
FP32
Operation
FP16
Operation
ACCELERATED BY GPU
Run On Tensor Cores

TENSORFLOW

Automatic Mixed Precision feature is available both in native TensorFlow and inside the TensorFlow container on

NVIDIA NGC container registry:

```
export TF_ENABLE_AUTO_MIXED_PRECISION=1
```

As an alternative, the environment variable can be set inside the TensorFlow Python script:

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

PYTORCH

Automatic Mixed Precision feature is available in the Apex repository on GitHub. To enable, add these two lines of code into your existing training script:

```
model, optimizer = amp.initialize(model, optimizer, opt_level="O1")

with amp.scale_loss(loss, optimizer) as scaled_loss:
    scaled_loss.backward()
```

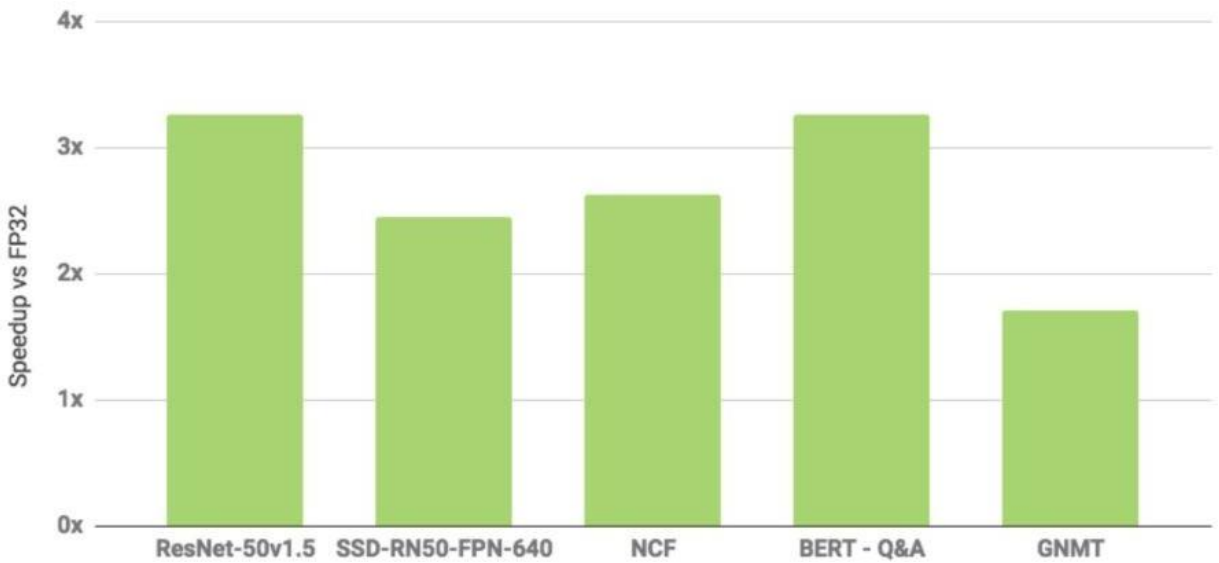
MXNET

Automatic Mixed Precision feature is available both in native MXNet (1.5 or later) and inside the MXNet container (19.04 or later) on NVIDIA NGC container registry. To enable the feature, add the following lines of code to your existing training script:

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

AUTOMATIC MIXED PRECISION IN TENSORFI

Upto 3X Speedup



TensorFlow Medium Post: Automatic Mixed Precision in TensorFlow for Faster AI Training on NVIDIA GPUs

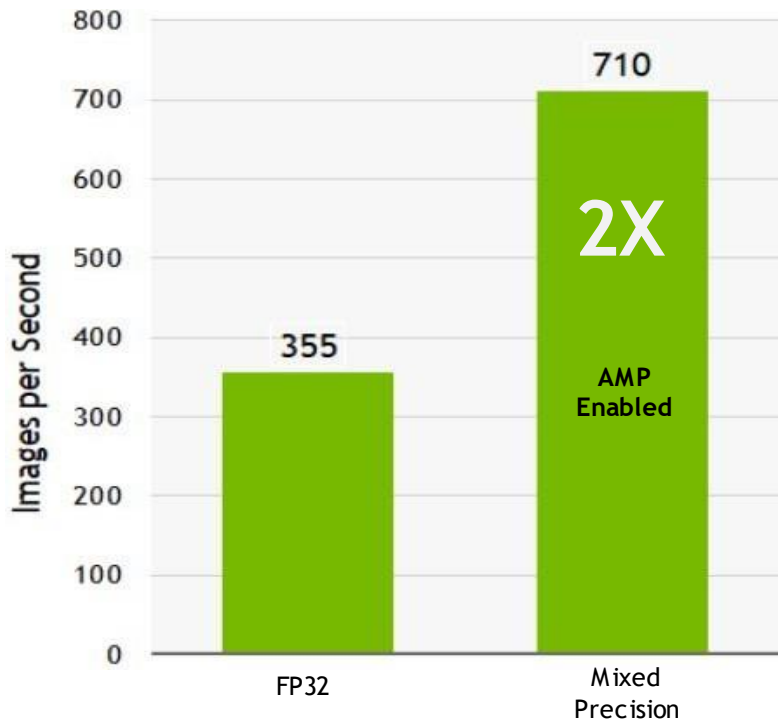
All models can be found at:
https://github.com/NVIDIA/DeepLearningExamples/tree/master/TensorFlow, except for ssd-rn50-fpn-640, which is here: https://github.com/tensorflow/models/tree/master/research/object_detection All performance collected on 1xV100-16GB, except bert-squadqa on 1xV100-32GB.
Speedup is the ratio of time to train for a fixed number of epochs in single-precision and Automatic Mixed Precision. Number of epochs for each model was matching the literature or common practice (it was also confirmed that both training sessions achieved the same model accuracy).
Batch sizes:. rn50 (v1.5): 128 for FP32, 256 for AMP+XLA; ssd-rn50-fpn-640: 8 for FP32, 16 for AMP+XLA; NCF: 1M for FP32 and AMP+XLA; bert-squadqa: 4 for FP32, 10 for AMP+XLA; GNMT: 128 for FP32, 192 for AMP.

AUTOMATIC MIXED PRECISION IN PYTORCH

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

- Plot shows ResNet-50 result with/without automatic mixed precision(AMP)
- More AMP enabled model scripts coming soon:

  Mask-R CNN, GNMT, NCF, etc.



Source: https://github.com/NVIDIA/apex/tree/master/examples/imagenet

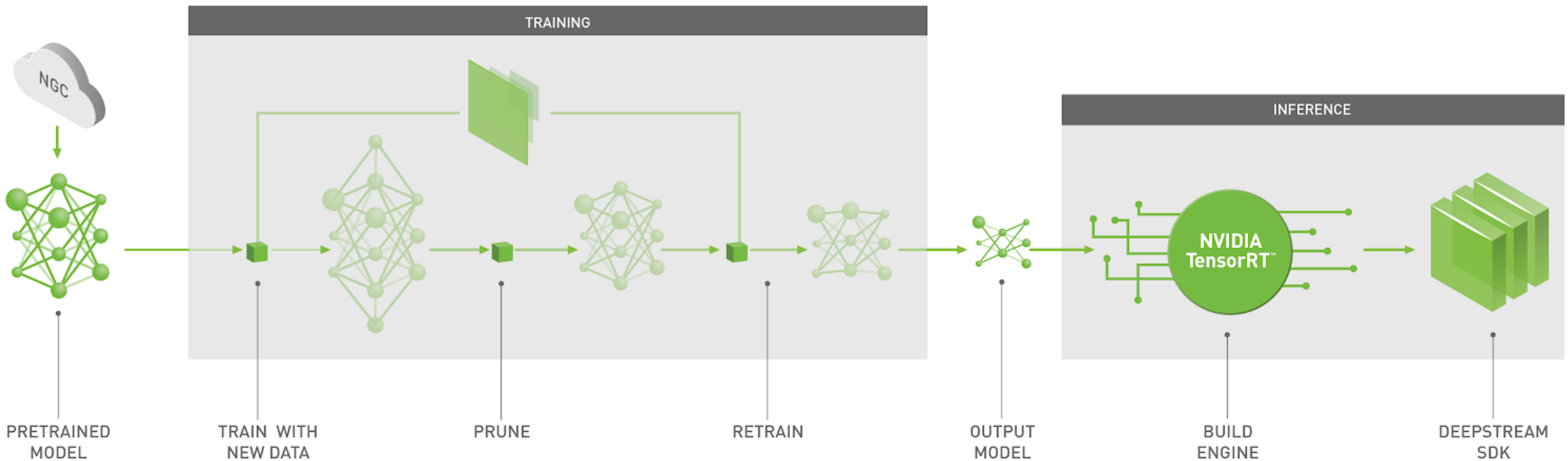
AUTOMATIC MIXED PRECISION IN MXNET

AMP speedup ~1.5X to 2X in comparison with FP32

(*) based on ResNet50 v1.5

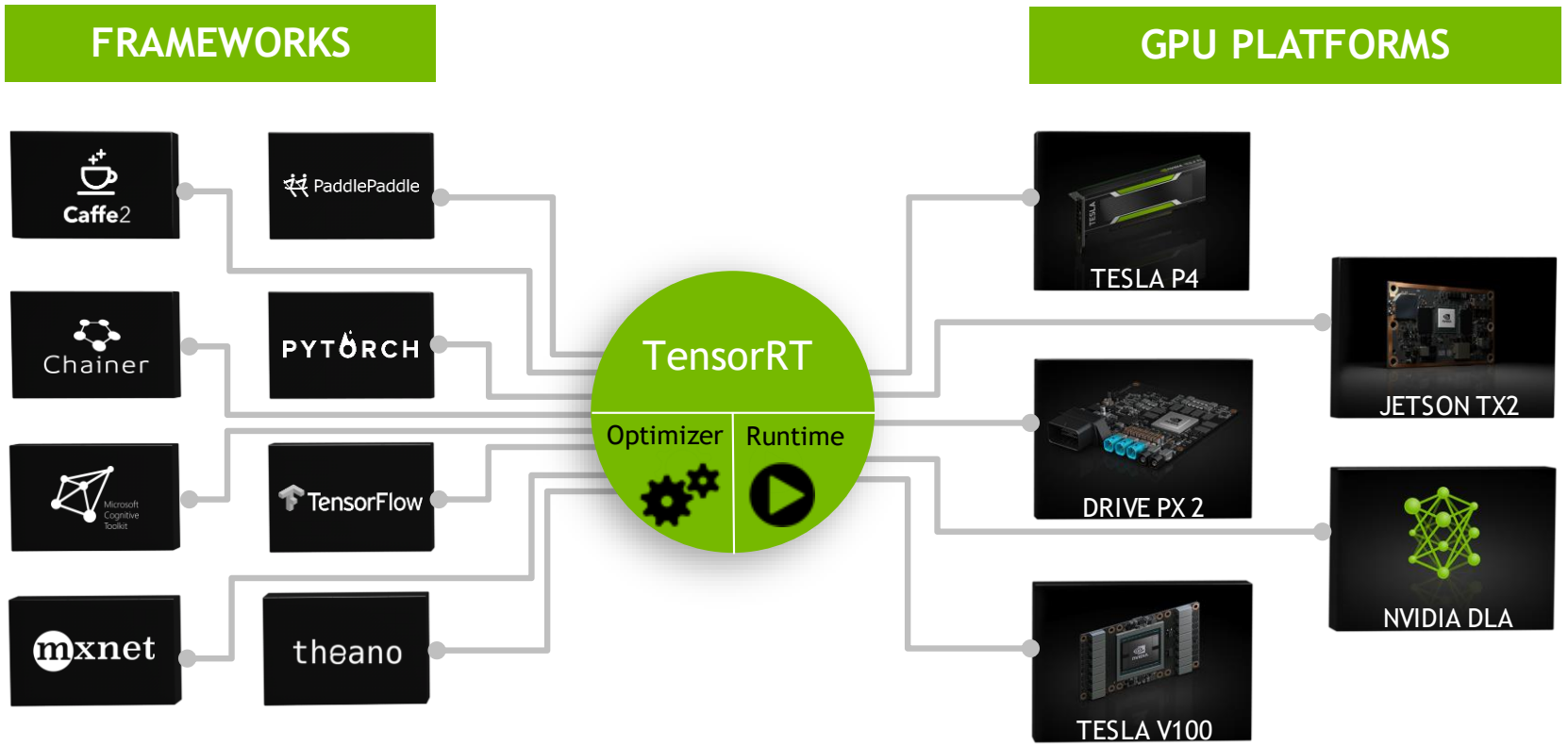
https://github.com/apache/incubator-mxnet/pull/14173

END TO END NVIDIA DEEP LEARNING WORKFLOW



NVIDIA TENSORRT

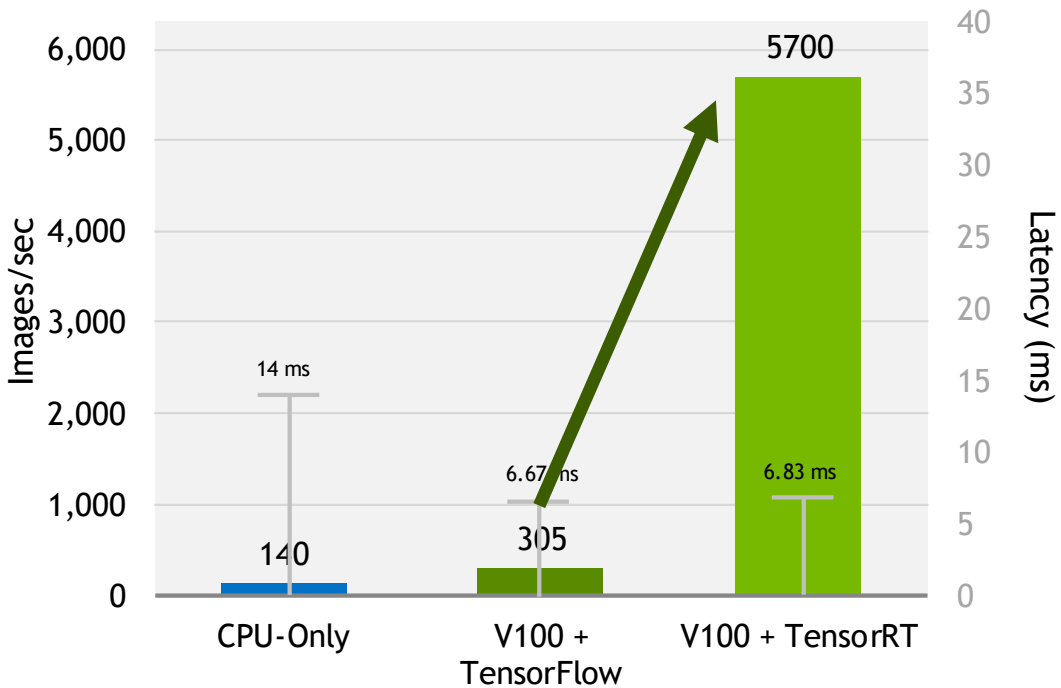
Programmable Inference Accelerator



developer.nvidia.com/tensorrt

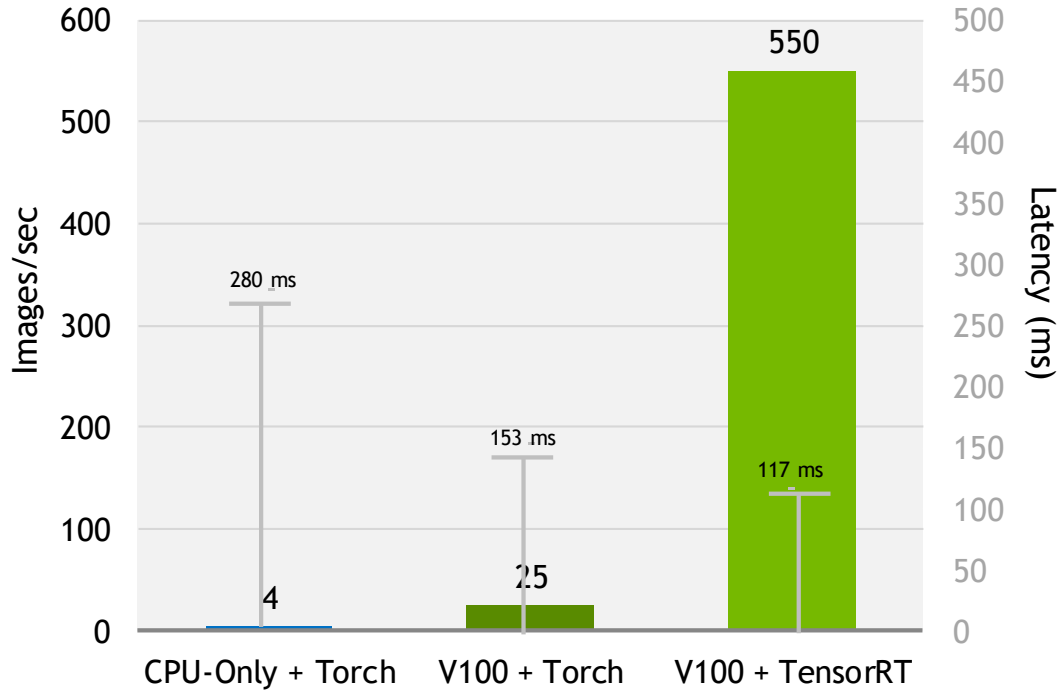
TENSORRT PERFORMANCE

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

developer.nvidia.com/tensorrt

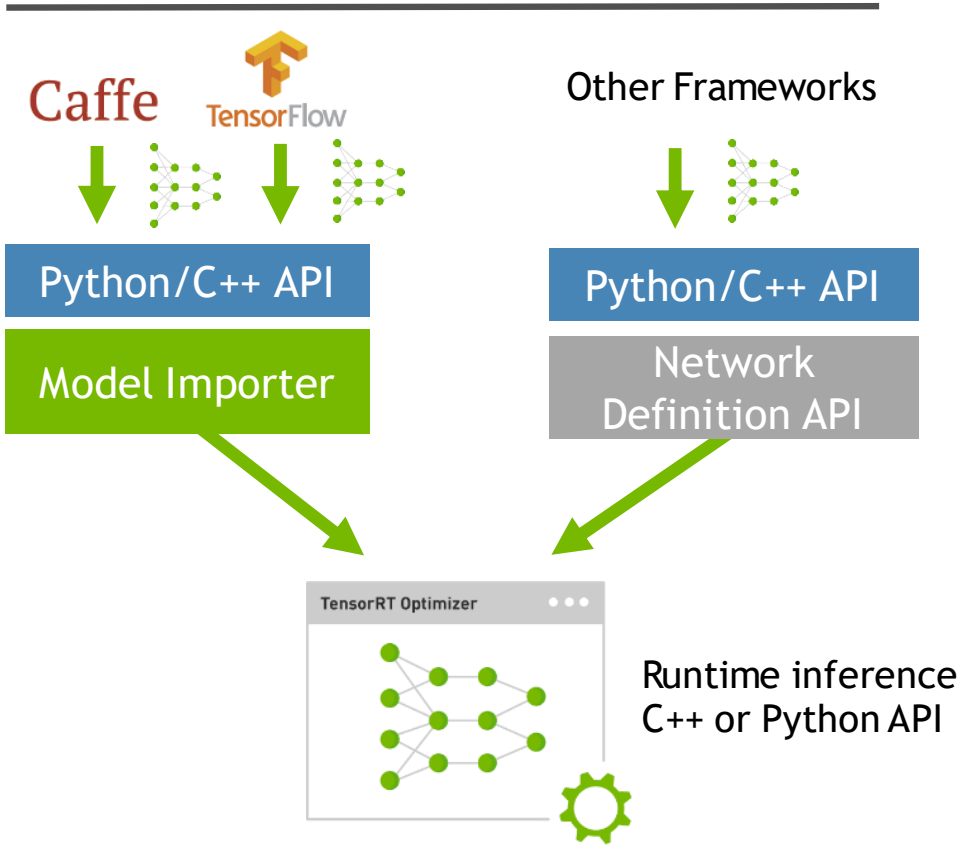
TENSORRT DEPLOYMENT WORKFLOW

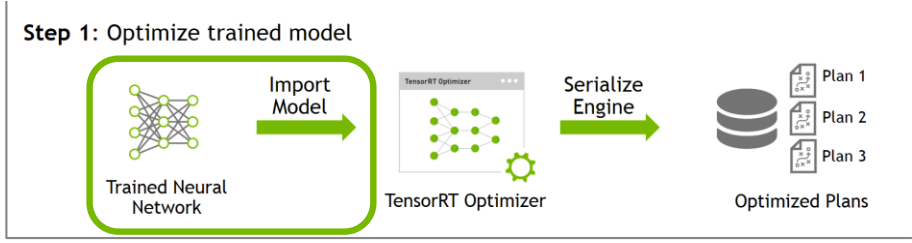
Step 1: Optimize trained model

Step 2: Deploy optimized plans with runtime

MODEL IMPORTING

- AI Researchers
- Data Scientists

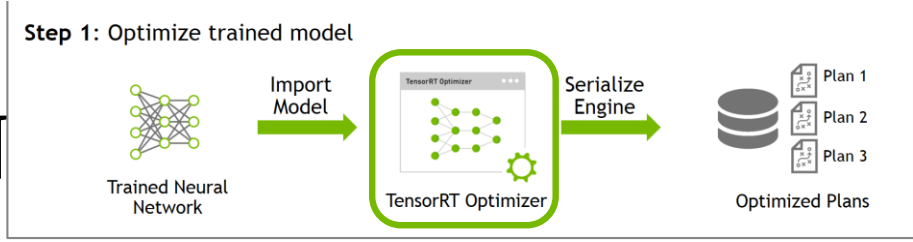


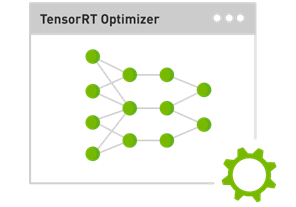


Example: Importing a TensorFlow model

```
import tensorrt as trt
import uff
from tensorrt.parsers import uffparser

G_LOGGER = trt.infer.ConsoleLogger(trt.infer.LogSeverity.INFO)

uff_model = uff.from_tensorflow_frozen_model("frozen_model.pb",
                                             "dense_2/Softmax")

parser = uffparser.create_uff_parser()
parser.register_input("input_1", (3,224,224),0)
parser.register_output("dense_2/Softmax")

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

runtime = trt.infer.create_infer_runtime(G_LOGGER)
context = engine.create_execution_context()
```

TENSORRT OPTIMIZATION





TensorRT Optimizer

Layer & Tensor Fusion

Weights & Activation Precision Calibration

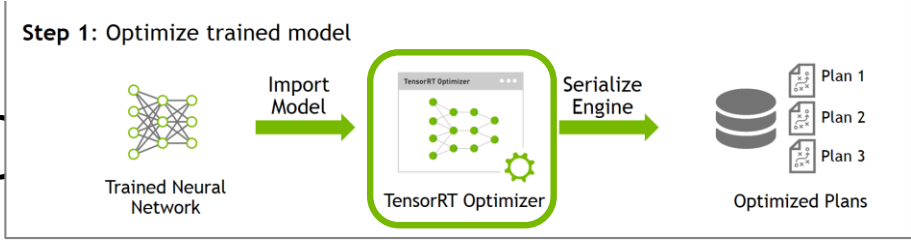
Kernel Auto-Tuning

Dynamic Tensor Memory

- 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)
```

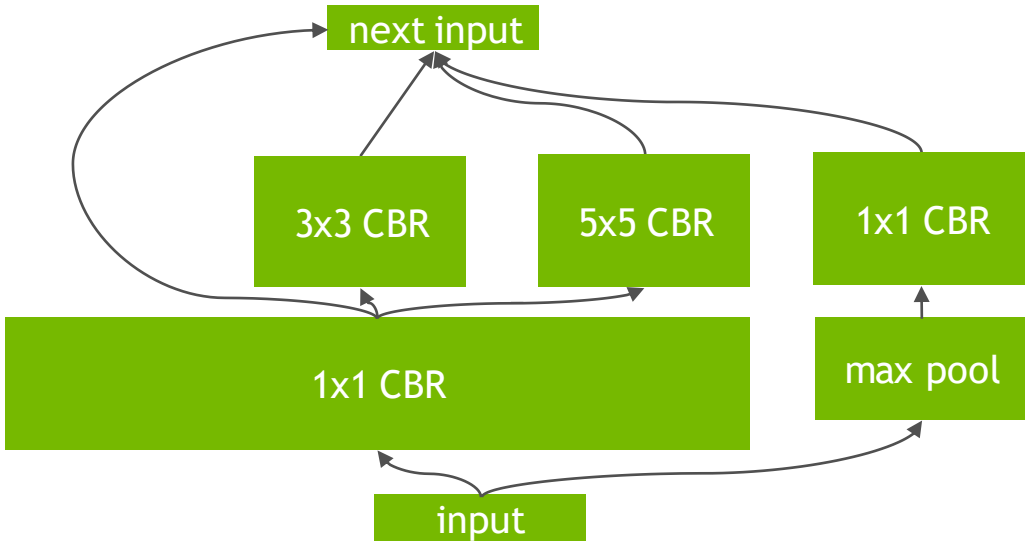
LAYER & TENSOR FUSIO

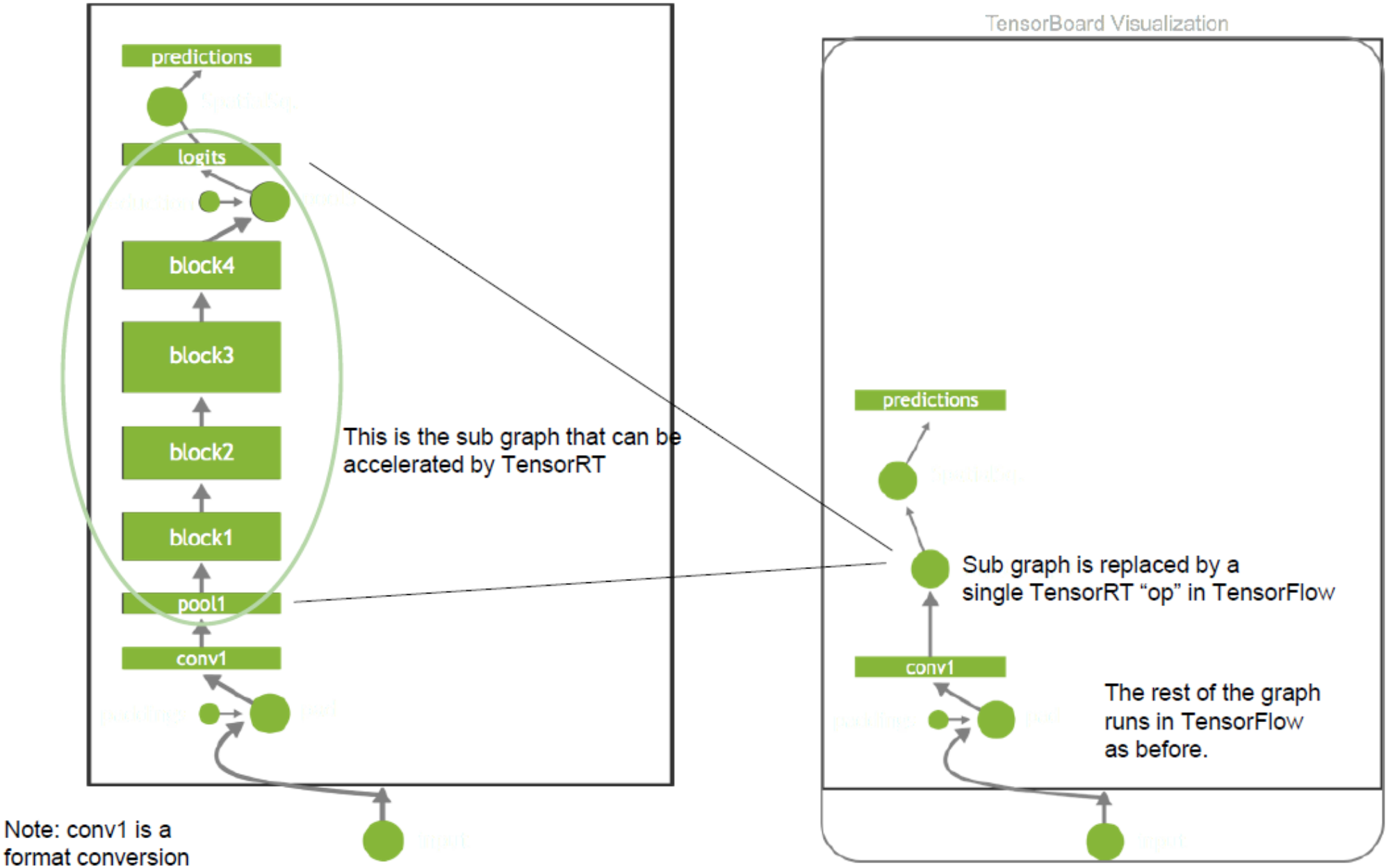


- Vertical Fusion
- Horizonal Fusion
- Layer Elimination

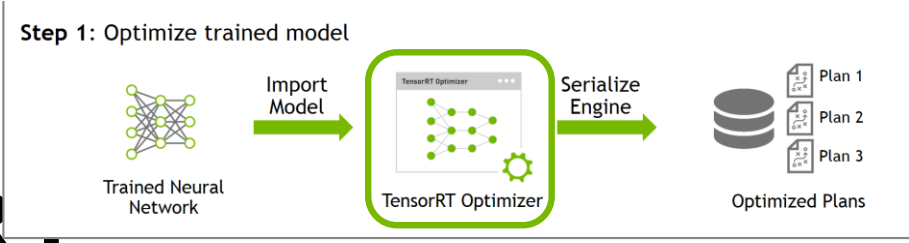
Network	Layers before	Layers after
VGG19	43	27
Inception V3	309	113
ResNet-152	670	159

TensorRT Optimized Network




predictions
logits
block4
block3
block2
block1
pool1
conv1
input
This is the sub graph that can be accelerated by TensorRT
TensorBoard Visualization
predictions
Sub graph is replaced by a single TensorRT “op” in TensorFlow
conv1
The rest of the graph runs in TensorFlow as before.
input
Note: conv1 is a format conversion

KERNEL AUTO-TUNING
DYNAMIC TENSOR MEMORY



Kernel Auto-Tuning

100s for specialized kernels
Optimized for every GPU platform


Tesla V100


Jetson TX2


Drive PX2

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

...

Dynamic Tensor Memory

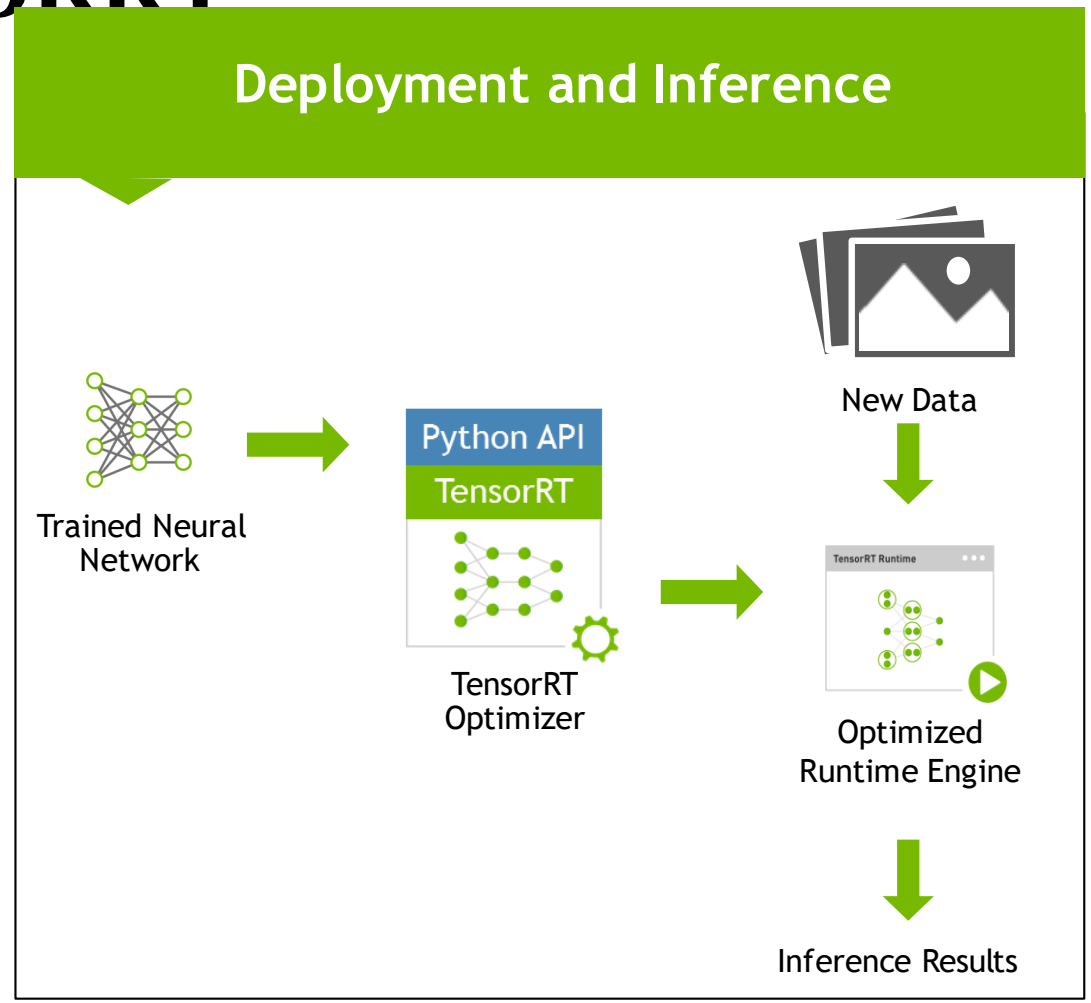
- Reduces memory footprint and improves memory re-use
- Manages memory allocation for each tensor only for the duration of its usage

EXAMPLE: DEPLOYING TENSORFLOW MODELS WITH TENSORRT

Import, optimize and deploy TensorFlow models using TensorRT python API

Steps:

- Start with a frozen TensorFlow model
- Create a model parser
- Optimize model and create a runtime engine
- Perform inference using the optimized runtime engine



7 STEPS TO DEPLOYMENT WITH TENSORRT

```
uff_model = uff.from_tensorflow_frozen_model("frozen_model_file.pb",
                                             OUTPUT_LAYERS)

parser = uffparser.create_uff_parser()


parser.register_input(INPUT_LAYERS[0], (INPUT_C,INPUT_H,INPUT_W),0)
parser.register_output(OUTPUT_LAYERS[0])


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


trt.utils.write_engine_to_file(save_path, engine.serialize())




engine = Engine(PLAN=plan,
                postprocessors={"output_layer_name":post_processing_function})

result = engine_single.infer(image)
```

← **Step 1:** Convert trained model into TensorRT format

← **Step 2:** Create a model parser

← **Step 3:** Register inputs and outputs

← **Step 4:** Optimize model and create a runtime engine

← **Step 5:** Serialize optimized engine

← **Step 6:** De-serialize engine

← **Step 7:** Perform inference

developer.nvidia.com/tensorrt

TensorRT Inference with TensorFlow

NVIDIA

TensorFlow

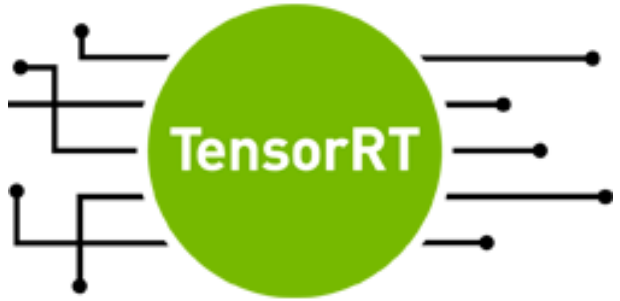
An end-to-end open source machine learning platform

- Powerful platform for research and experimentation
- Versatile, easy model building
- Robust ML production anywhere
- Most popular ML project on Github

41m Downloads

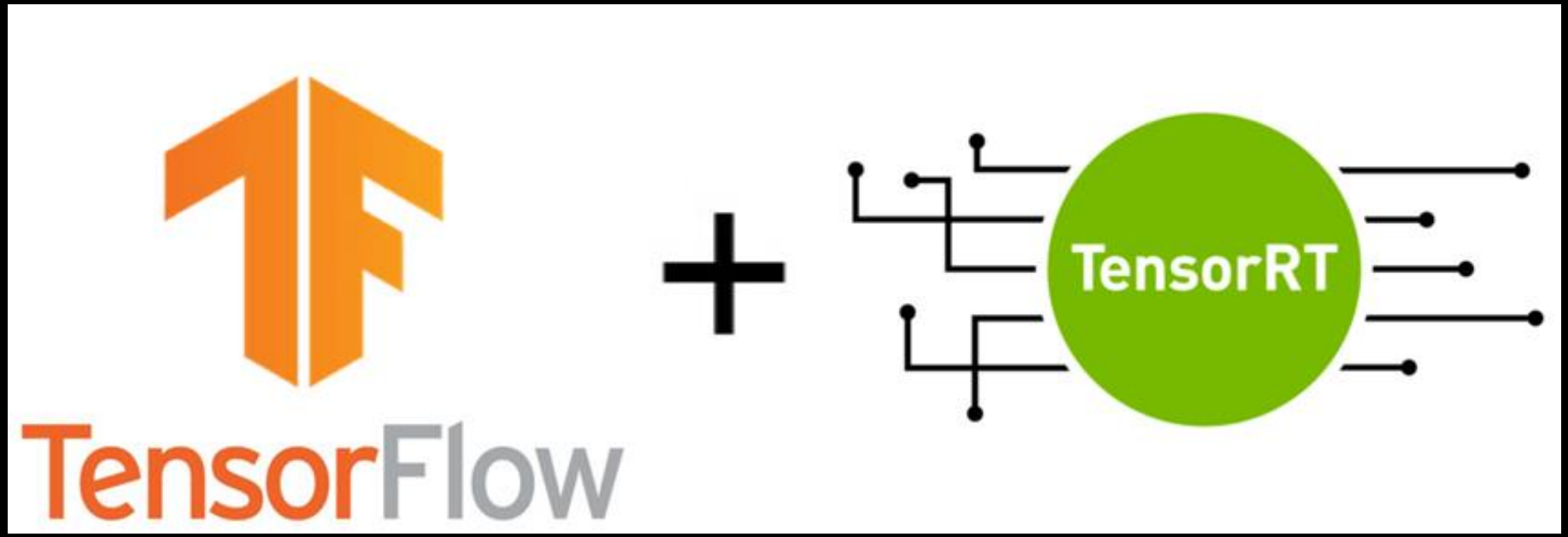
NVIDIA TensorRT

Platform for High-Performance Deep Learning Inference



- Optimize and Deploy neural networks in production environments
- Maximize throughput for latency-critical apps with optimizer and runtime
- Deploy responsive and memory efficient apps with INT8 & FP16

300k Downloads in 2018



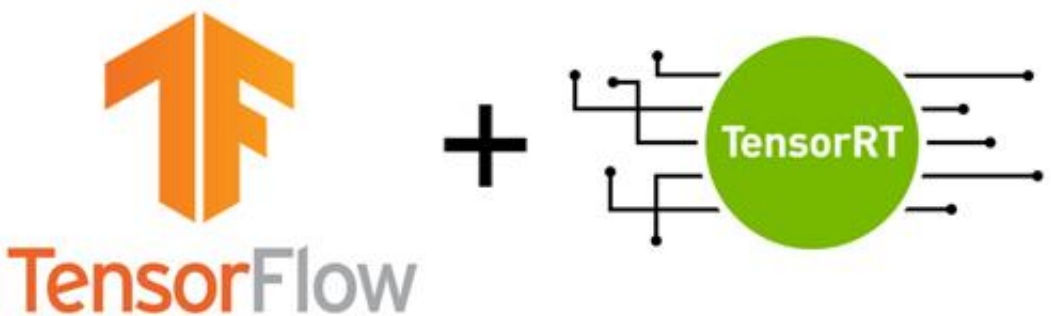
TF-TRT = TF + TRT

TensorRT Inference with TensorFlow

- **Benefits to using TF-TRT**
- How to use
- Customer experience: Clarifai
- How TF-TRT works
- Additional Resources

Benefits to using TF-TRT

- Optimize TF inference while still using the TF ecosystem
- Simple API: up to 8x performance gain with little effort
- Fallback to native TensorFlow where TensorRT does not support



Over 10 optimized models with published examples

Models	TF FP32 (imgs/s)	TF-TRT INT8 (imgs/s)	Speedup
ResNet-50	399	3053	**7.7x**
Inception V4	158	1128	**7.1x**
Mobilenet V1	1203	4975	**4.1x**
NASNet large	43	162	**3.8x**
VGG16	245	1568	**6.4x**
SSD Mobilenet V2	102	411	**4.0x**
SSD Inception V2	82	327	**4.0x**

- Performance optimizations soon: More NLP and Object Detection Models
- For non-optimized layers, fallback support is provided by TensorFlow

TensorFlow FP32 vs TensorFlow-TensorRT INT8 on T4, largest possible batch size, no I/O.
NGC Tensorflow 19.07 with scripts: https://github.com/tensorflow/tensorrt/blob/master/tftrt/examples/image-classification/image_classification.py

FP16 accuracy

Models	TF FP32	TF-TRT FP16
Mobilenet V2	74.08	74.07
NASNet Mobile	73.97	73.87
ResNet 50 V1.5	76.51	76.48
ResNet 50 V2	76.43	76.40
VGG 16	70.89	70.91
Inception V3	77.99	77.97
SSD Mobilenet v1	23.06	23.07

FP16 accuracy is within 0.1% of FP32 accuracy

Top-1 metric (%) for classification models. mAP for SSD detection models.
Complete data: https://docs.nvidia.com/deeplearning/dgx/integrate-tf-trt/index.html#verified-models

INT8 accuracy

Models	TF FP32	TF-TRT INT8
Mobilenet V2	74.08	73.90
NASNet Mobile	73.97	73.55
ResNet 50 V1.5	76.51	76.23
ResNet 50 V2	76.43	76.30
VGG 16	70.89	70.78
Inception V3	77.99	77.85

INT8 accuracy is within 0.2% of FP32 accuracy except for NASNet Mobile within 0.5%.

Top-1 metric (%) for classification models.

Complete data: https://docs.nvidia.com/deeplearning/dgx/integrate-tf-trt/index.html#verified-models

TensorRT ONNX PARSER

Parser to import ONNX-models into TensorRT

Optimize and deploy models from ONNX-supported frameworks in production

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

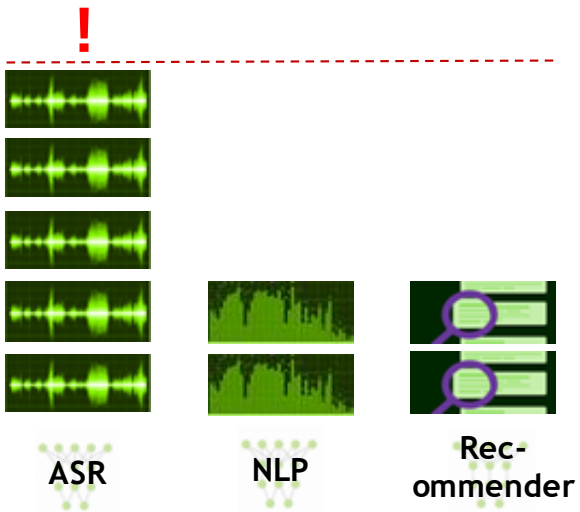
C++ and Python APIs to import ONNX models

New samples demonstrating step-by-step process to get started

INEFFICIENCY LIMITS INNOVATION

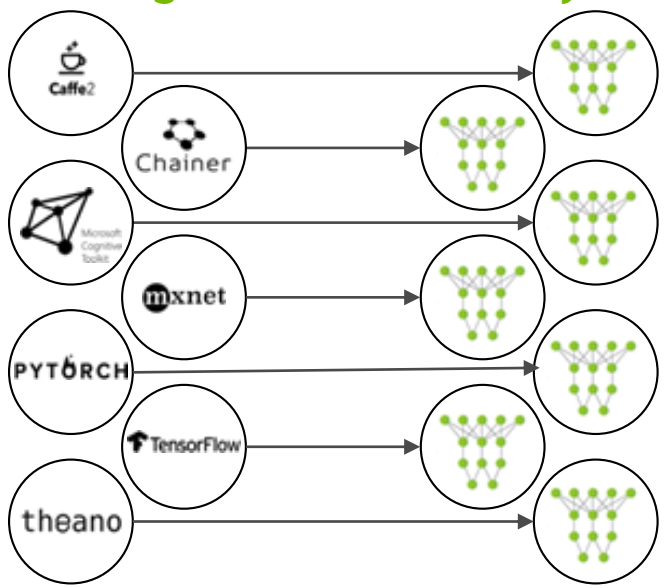
Difficulties with Deploying Data Center Inference

Single Model Only



Some systems are overused while others are underutilized

Single Framework Only



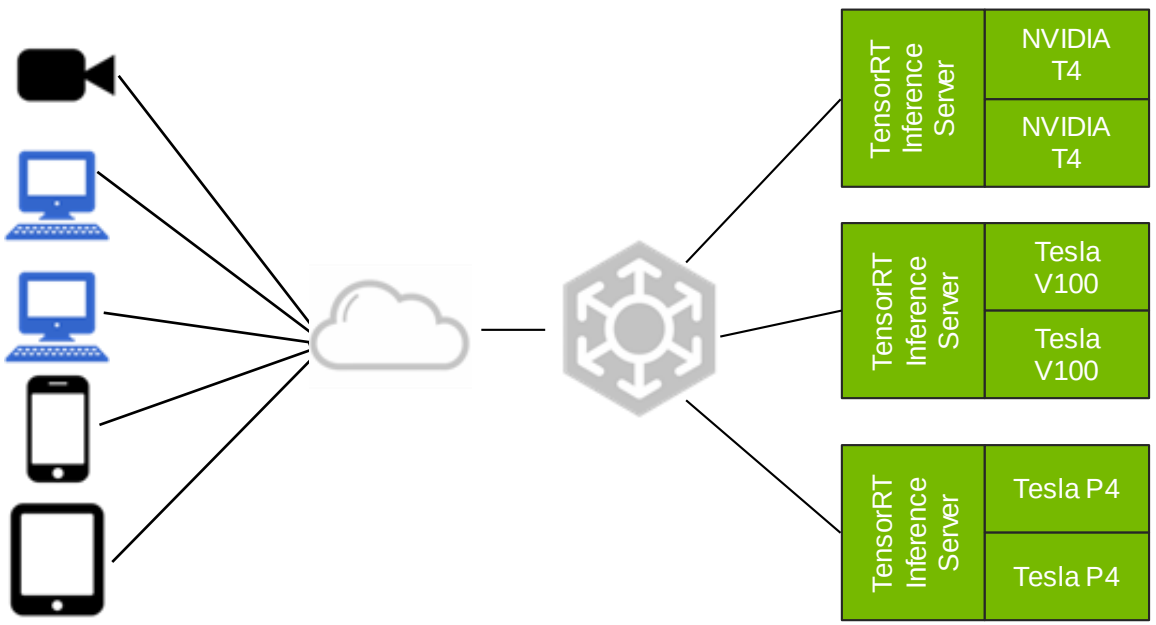
Solutions can only support models from one framework

Custom Development

Developers need to reinvent the plumbing for every application

NVIDIA TENSORRT INFERENCE SERVER

Production Data Center Inference Server



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

FEATURES

Concurrent Model Execution
Multiple models (or multiple instances of same model) may execute on GPU simultaneously

CPU Model Inference Execution
Framework native models can execute inference requests on the CPU

Metrics
Utilization, count, memory, and latency

Custom Backend
Custom backend allows the user more flexibility by providing their own implementation of an execution engine through the use of a shared library

Model Ensemble
Pipeline of one or more models and the connection of input and output tensors between those models (can be used with custom backend)

Dynamic Batching
Inference requests can be batched up by the inference server to 1) the model-allowed maximum or 2) the user-defined latency SLA

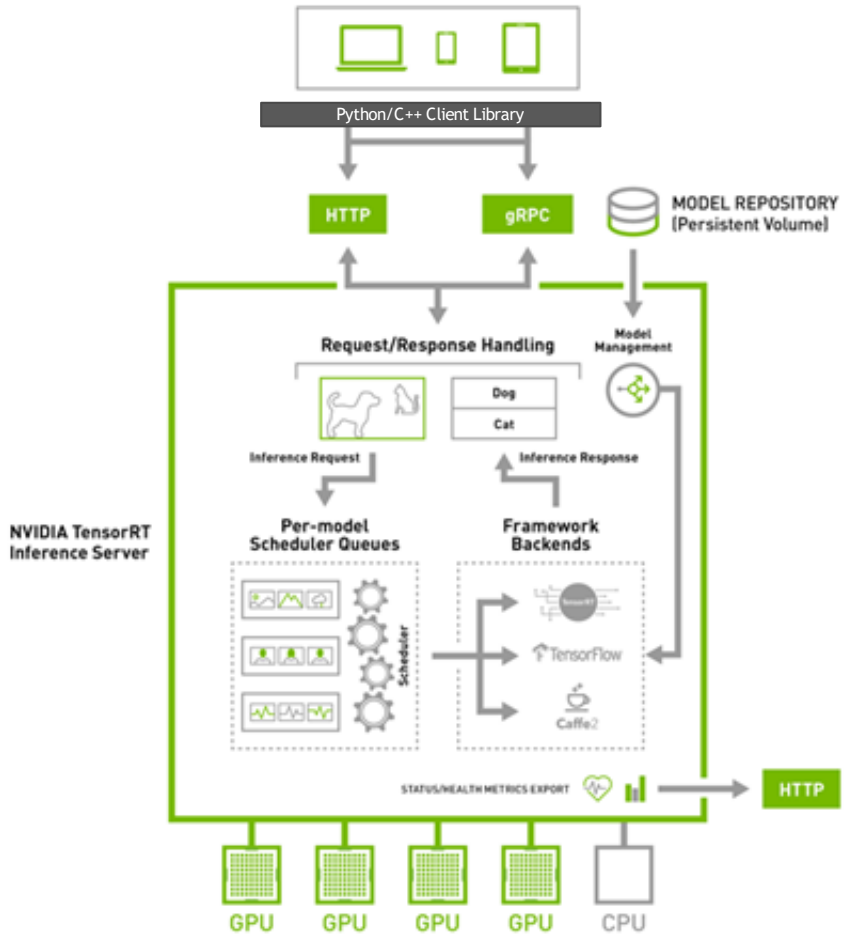
Multiple Model Format Support
PyTorch JIT (.pt)
TensorFlow GraphDef/SavedModel
TensorFlow and TensorRT GraphDef
ONNX graph (ONNX Runtime)
TensorRT Plans
Caffe2 NetDef (ONNX import path)

CMake build
Build the inference server from source making it more portable to multiple OSes and removing the build dependency on Docker

Streaming API
Built-in support for audio streaming input e.g. for speech recognition

INFERENCE SERVER ARCHITECTURE

Available with Monthly Updates



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

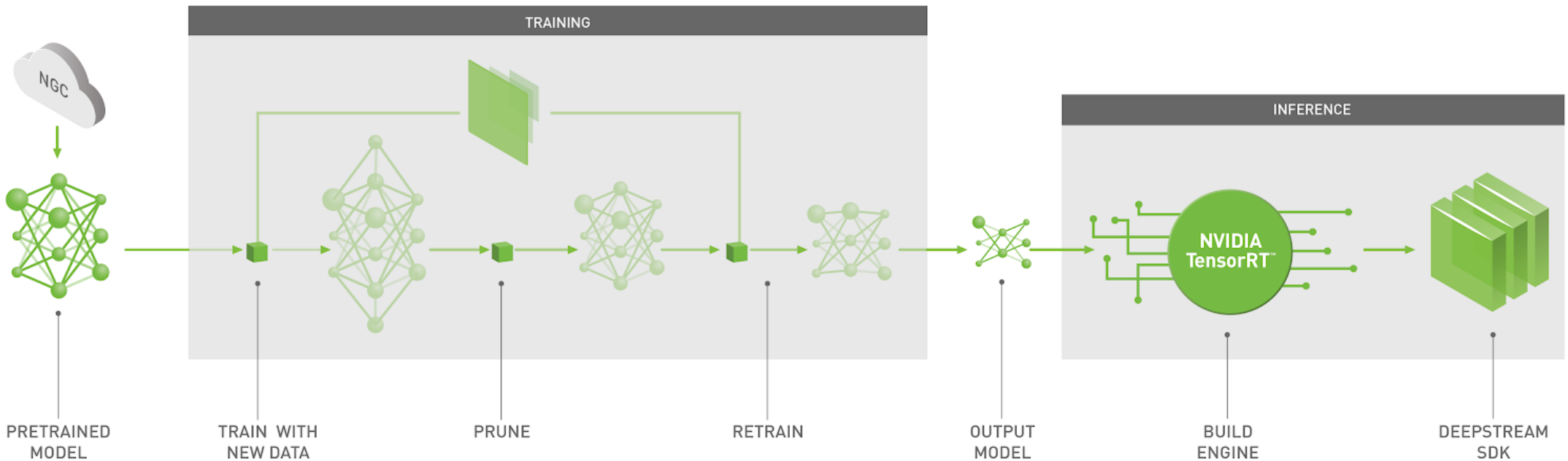
Server HTTP REST API/gRPC

Python/C++ client libraries

Additional resources

- **GTC Technical presentation:** https://developer.nvidia.com/gtc/2019/video/S9431/video
- **TF-TRT user guide:** https://docs.nvidia.com/deeplearning/dgx/tf-trt-user-guide/index.html
- **NVIDIA DLI course on TF-TRT:** https://www.nvidia.com/en-us/deep-learning-ai/education/
- **Monthly release notes:** https://docs.nvidia.com/deeplearning/dgx/tf-trt-release-notes/index.html
- **Google Blog on TF-TRT inference:** https://cloud.google.com/blog/products/ai-machine-learning/running-tensorflow-inference-workloads-at-scale-with-tensorrt-5-and-nvidia-t4-gpus
- **Nvidia Developer Blog:** https://devblogs.nvidia.com/tensorrt-integration-speeds-tensorflow-inference/

END TO END NVIDIA DEEP LEARNING WORKFLOW



AGENDA

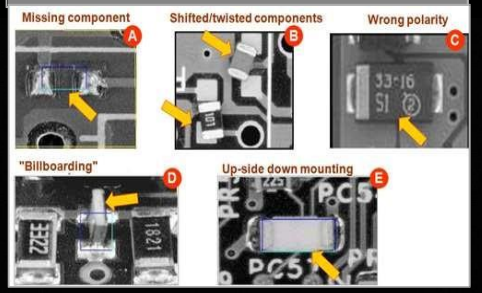
1. Intelligent Video Analytics
2. Deepstream SDK
   a. What is Deepstream SDK?
   a. Why Deepstream SDK?
   b. What's new with DS4.0?
   c. Deepstream Building Blocks
3. Getting started with Deepstream SDK
   a. Where to start?
   b. Directory hierarchy
   c. Configurable file and pipeline details
   d. Running application
4. Building with Deepstream SDK
   a. Real world use cases with demo

INTELLIGENT VIDEO ANALYTICS (IVA) FOR EFFICIENCY AND SAFETY

Access Control

Public Transit

Industrial Inspection



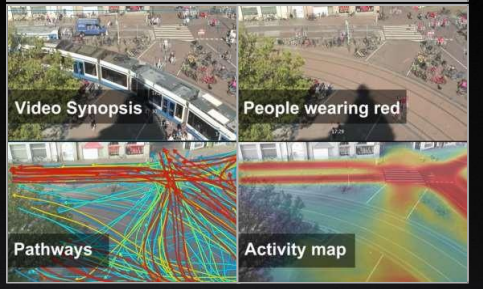
Traffic Engineering

Retail Analytics

Logistics

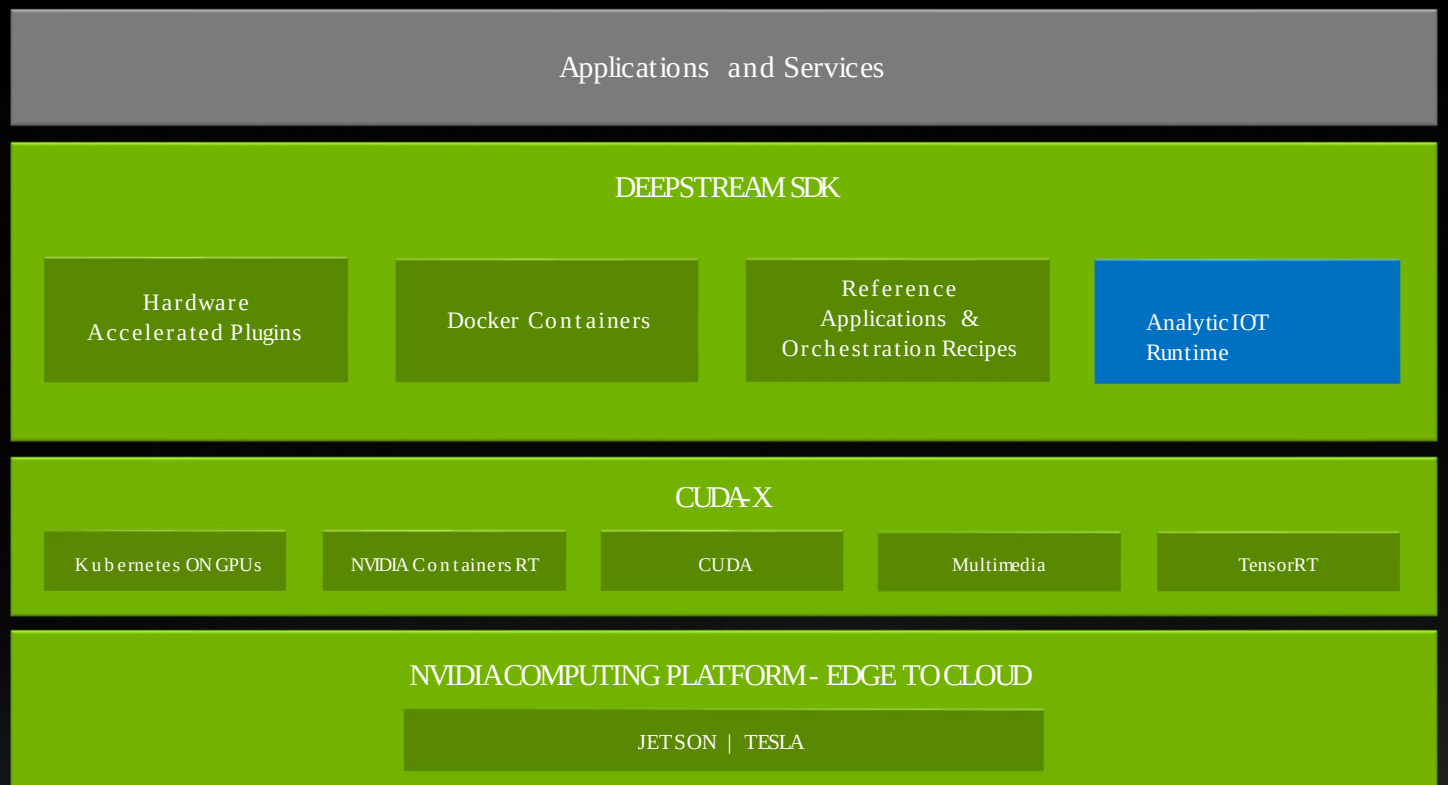
Critical Infrastructure

Public Safety



NVIDIA.

WHAT IS DEEPSTREAM?



WHY DEEPSTREAM?

Broader Use Cases and Industries

Build your own application for smart cities, retail analytics, industrial inspection, logistics, and more

Faster Time to Progress

Provides ready to use building blocks and IP simplify building your innovative product.

Faster Time to Market

Provides ready to use building blocks and IP simplify building your innovative product.

Performance Driven

Low latency and exceptional performance optimized for NVIDIA GPUs for real-time edge analytics

Cloud Integration

Pushbutton IoT solution integration to build applications and services with Cloud Service Providers.

Faster Time to Progress

Iterate and integrate by quick plug and play of popular plug-ins that are pre-packaged or build your own.

DEEPSTREAM SDK

Plugins (build with open source, 3rd party, NV

DNN inference/TensorRT plugins

Communications plugins

Video/image capture and processing plugins

3rd party library plugins … …

Analytics - multi-camera, multi-sensor framework

DeepStream in containers, Multi-GPU

Tracking & analytics across large scale/ multi-camera

Streaming and Batch Analytics

Event fabric

Development Tools

End to end reference applications

App building/configuration tools

End-end orchestration recipes & adaptation guides

Plugin templates, custom IP integration

DeepStream SDK

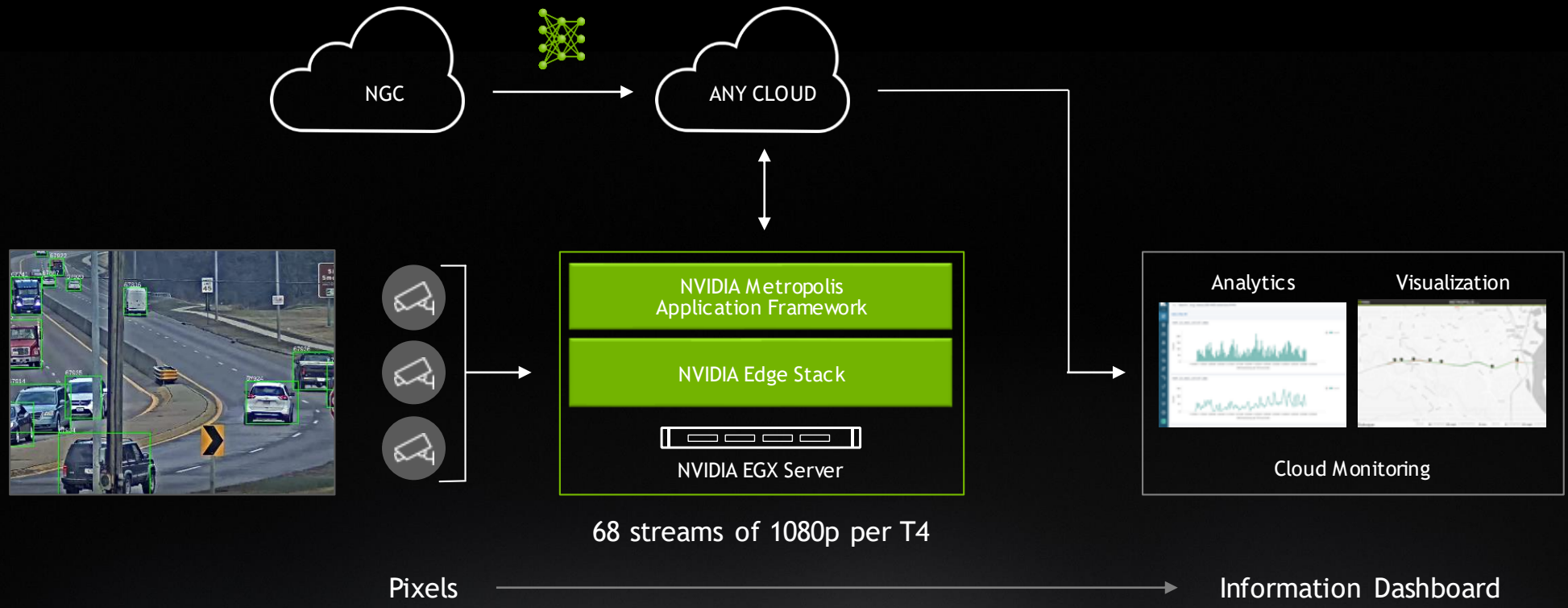
TensorRT | Multimedia APIs/ Video Codec SDK | Imaging & Dewarping library | Metadata & messaging | NV containers | Message bus clients | Multi-camera tracking lib

Linux, CUDA

Perception infra - Jetson, Tesla server (Edge and cloud)

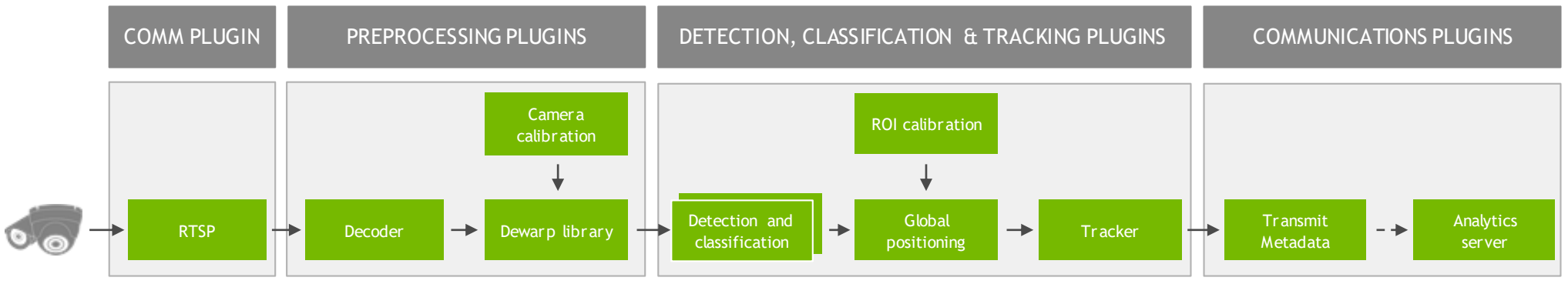
Analytics infra - Edge server, NGC, AWS, Azure

REAL TIME INSIGHTS, HIGHEST STREAM DENSITY

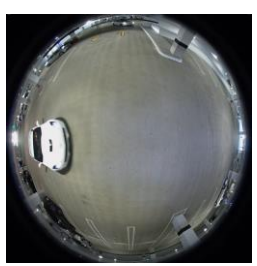


Smart Parking

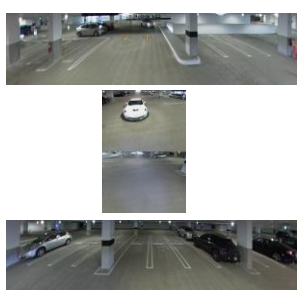
PERCEPTION GRAPH



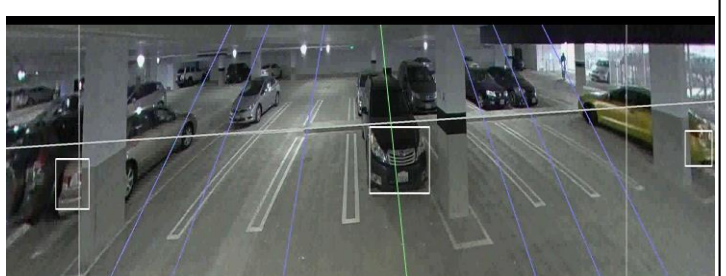
360d feeds



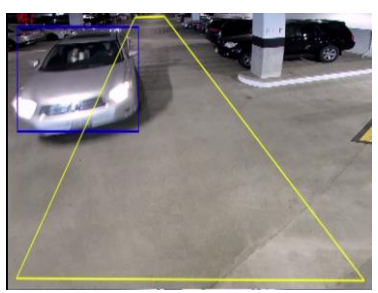
Dewarping



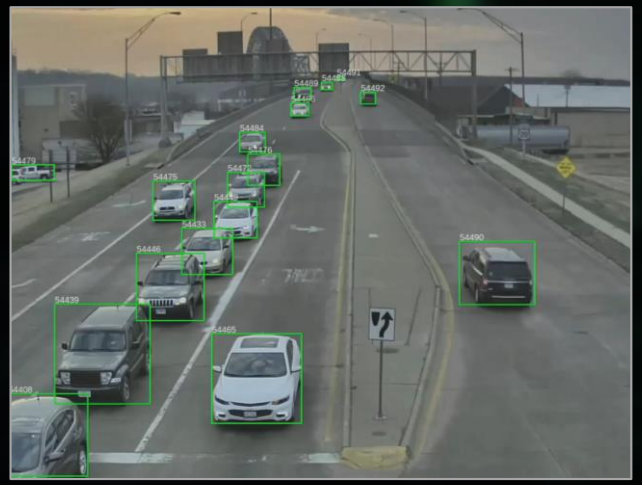
ROI: Lines



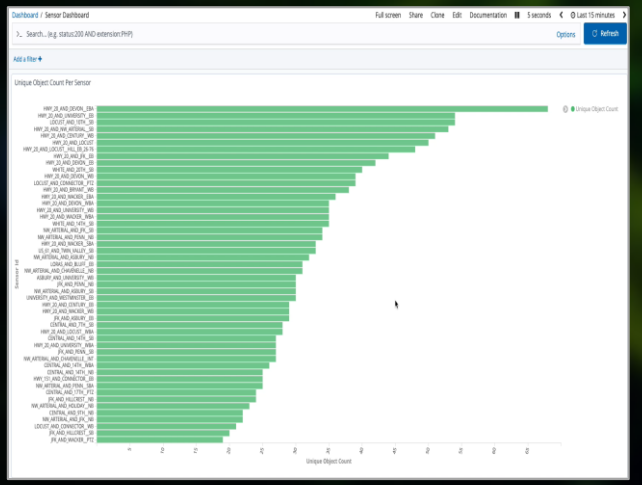
ROI: Polygon



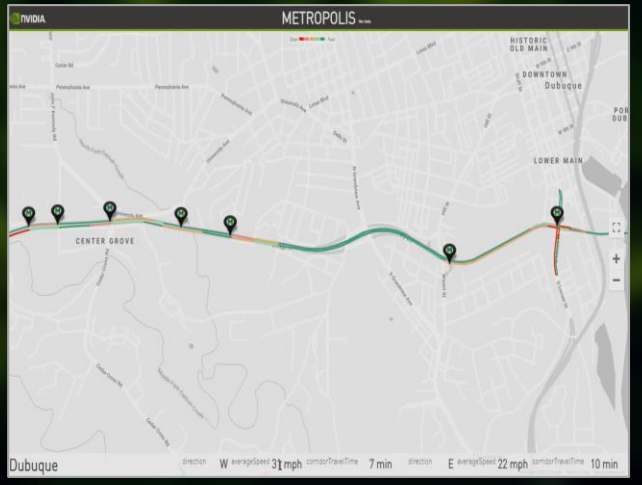
Perception



Analytics



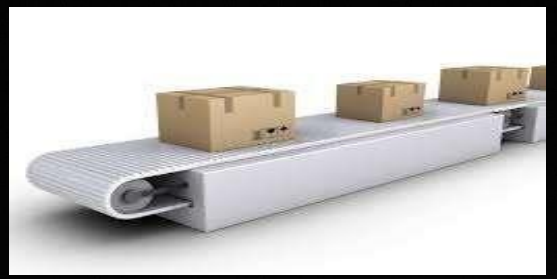
Visualization



VIDEO : INTELLIGENT TRAFFIC SYSTEM

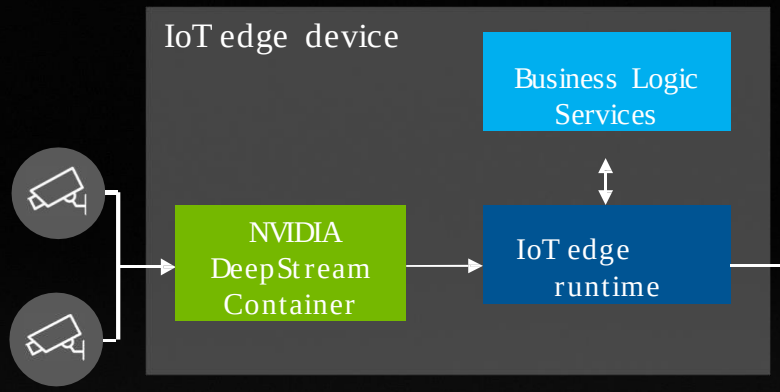
WAREHOUSE LOGISTICS: INVENTORY SORTING

USE CASE



Detect and flag packages on a conveyor belt

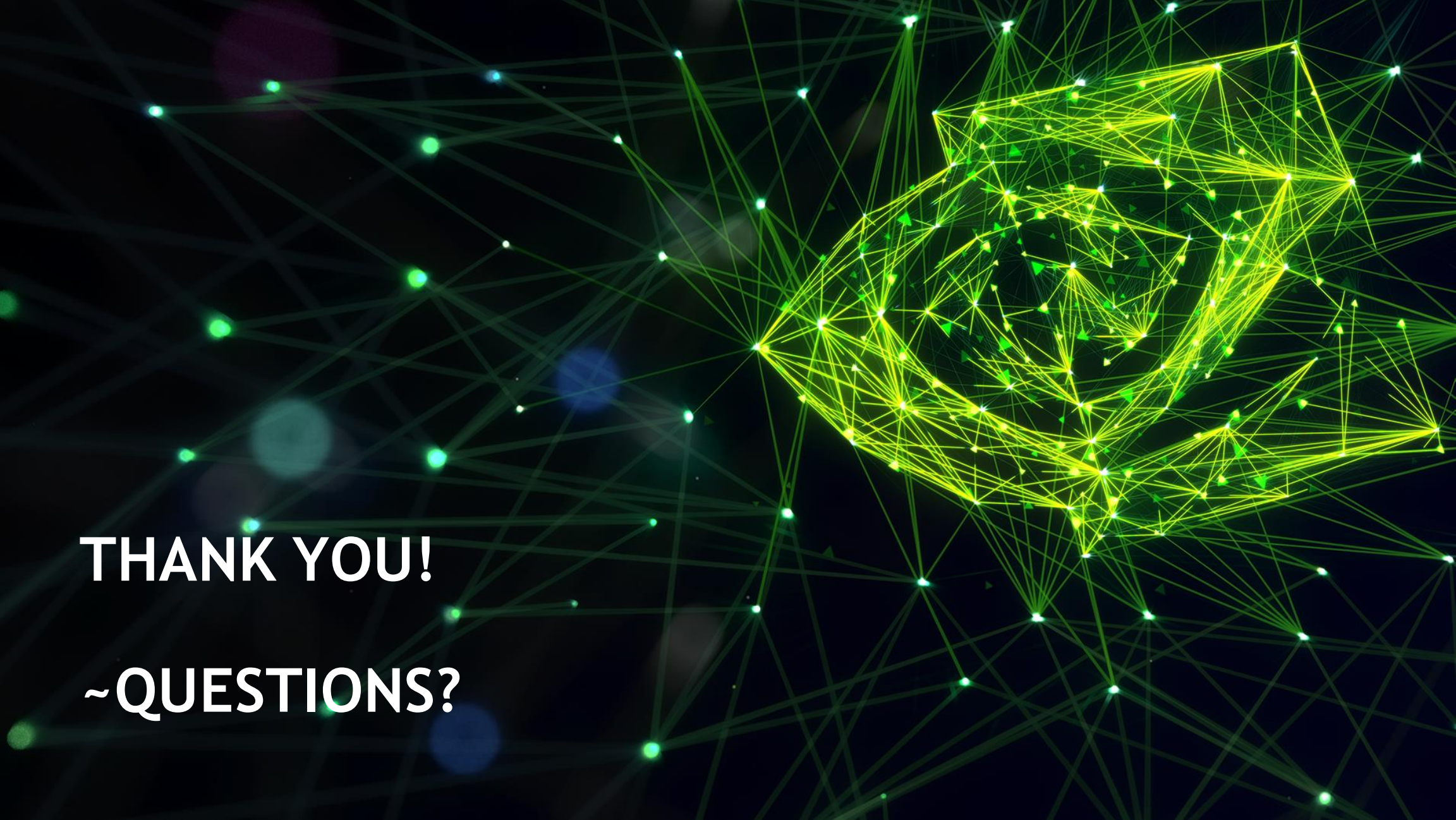
SOLUTION



DeepStream container can connect to Azure IoT central through Azure IoT edge runtime

Microsoft Azure Lenovo

NVIDIA


THANK YOU!
~QUESTIONS?