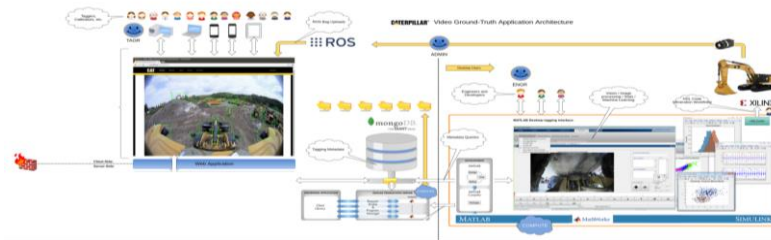


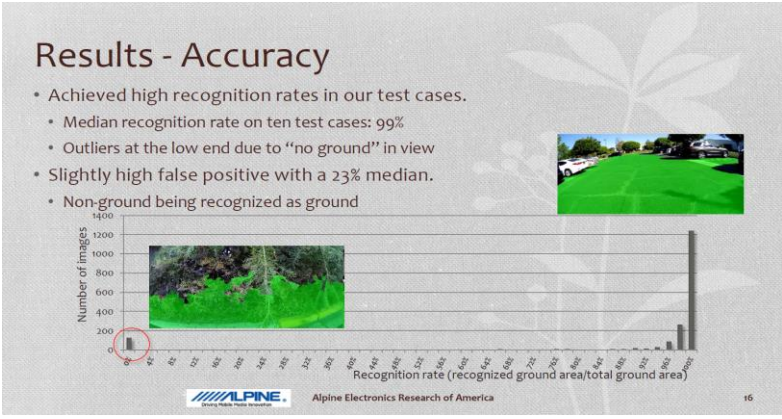
# Applying Artificial Intelligence to Product Development

Sebastian Bomberg, Application Engineering

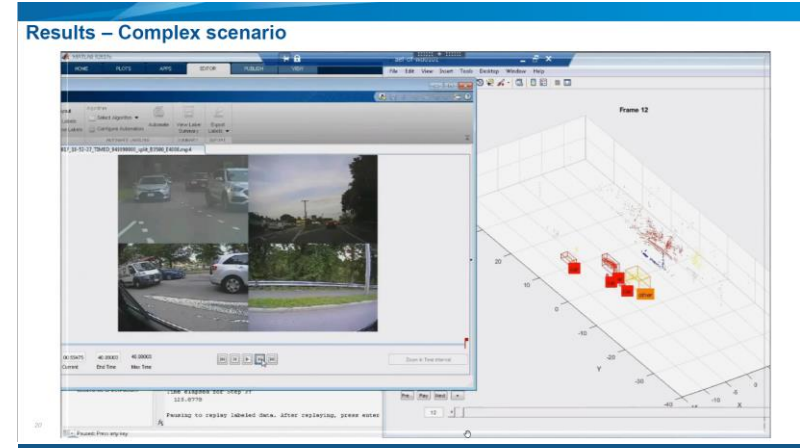
# Diverse Set of Automotive Customers use MATLAB for AI



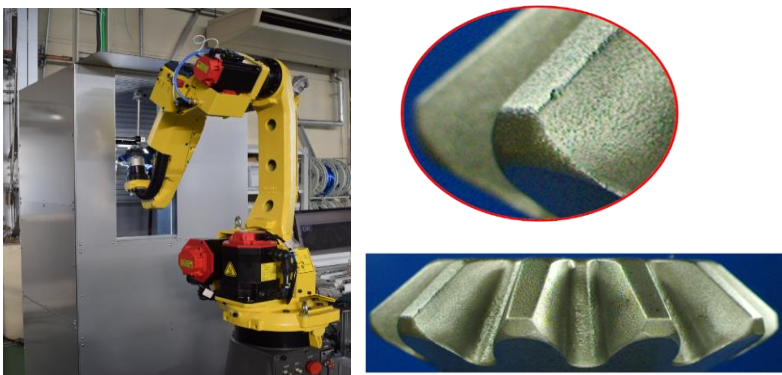
**Caterpillar**  
Cloud Based Data Labeling



**Alpine**  
Ground Detection

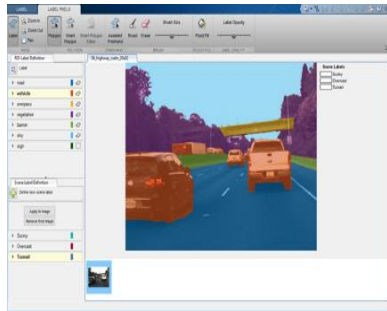


**Veoneer**  
Radar Sensor Verification

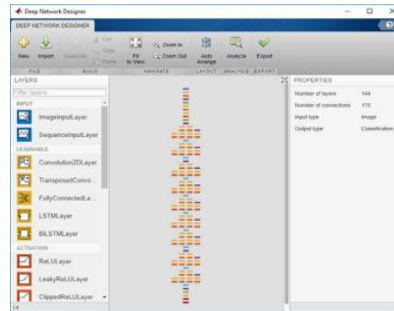


**Musashi Seimitsu**  
Automotive Part Defect Detection

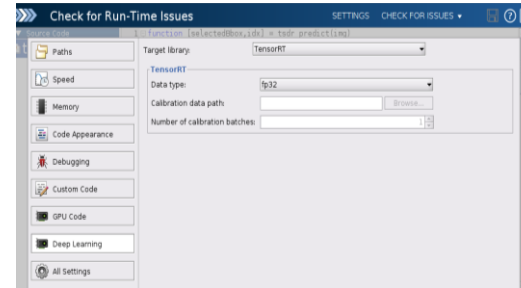
# Outline



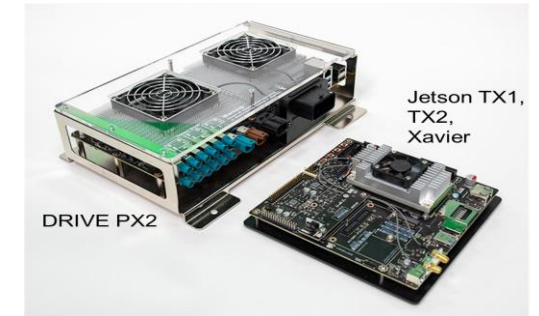
Ground Truth Labeling



Network Design and Training



CUDA and TensorRT Code Generation



Jetson Xavier and DRIVE Xavier Targeting

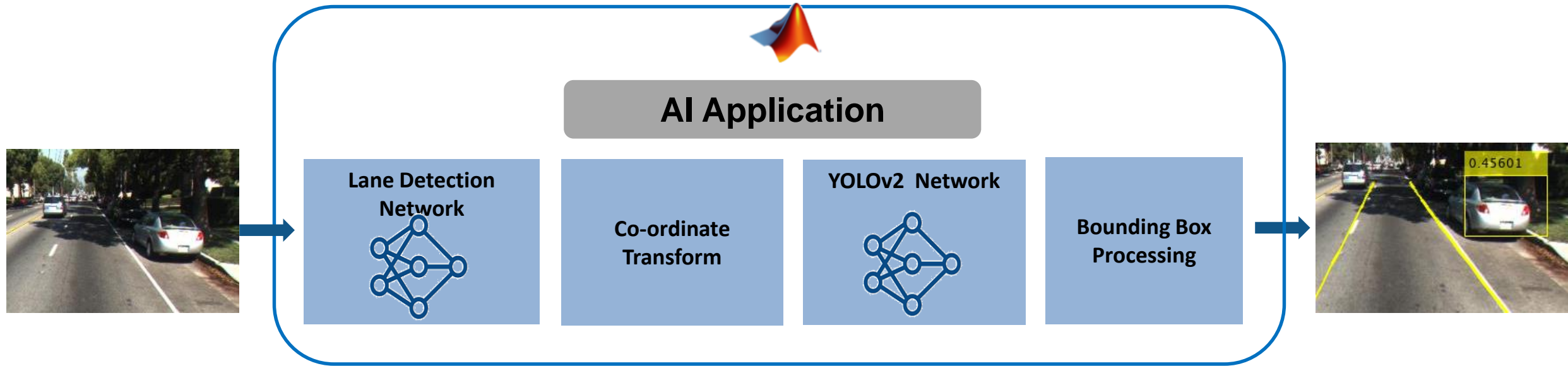
## Key Takeaways

**Platform Productivity:** Workflow automation, ease of use  
**Framework Interoperability:** ONNX, Keras-TensorFlow, Caffe

## Key Takeaways

**Optimized CUDA and TensorRT code generation**  
**Jetson Xavier and DRIVE Xavier targeting**  
**Processor-in-loop(PIL) testing and system integration**

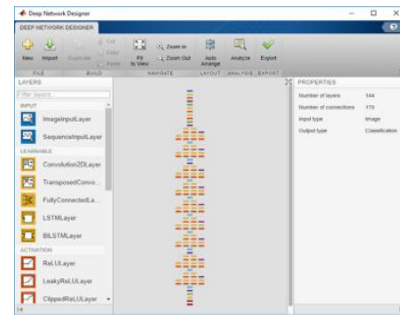
# Example Used in Today's Talk



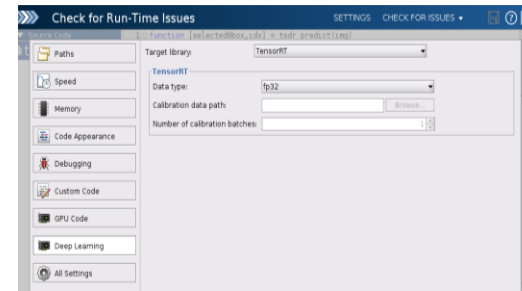
# Outline



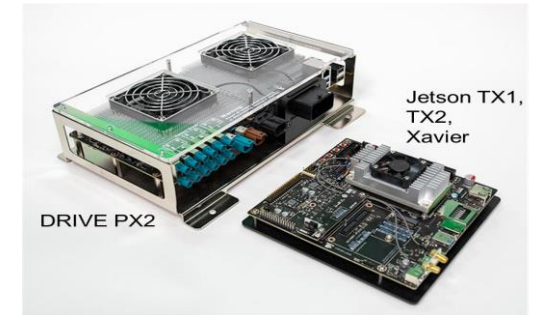
Ground Truth Labeling



Network Design and Training



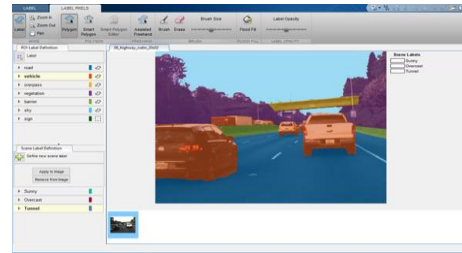
CUDA and TensorRT Code Generation



Jetson Xavier and DRIVE Xavier Targeting



Unlabeled Training Data



Ground Truth Labeling



```
>> gTruth
gTruth =
    groundTruth with properties:
        DataSource: [1x1 groundTruthDataSource]
        LabelDefinitions: [4x3 table]
        LabelData: [250x4 timetable]
>> gTruth.LabelData
ans =
    250x4 timetable
```

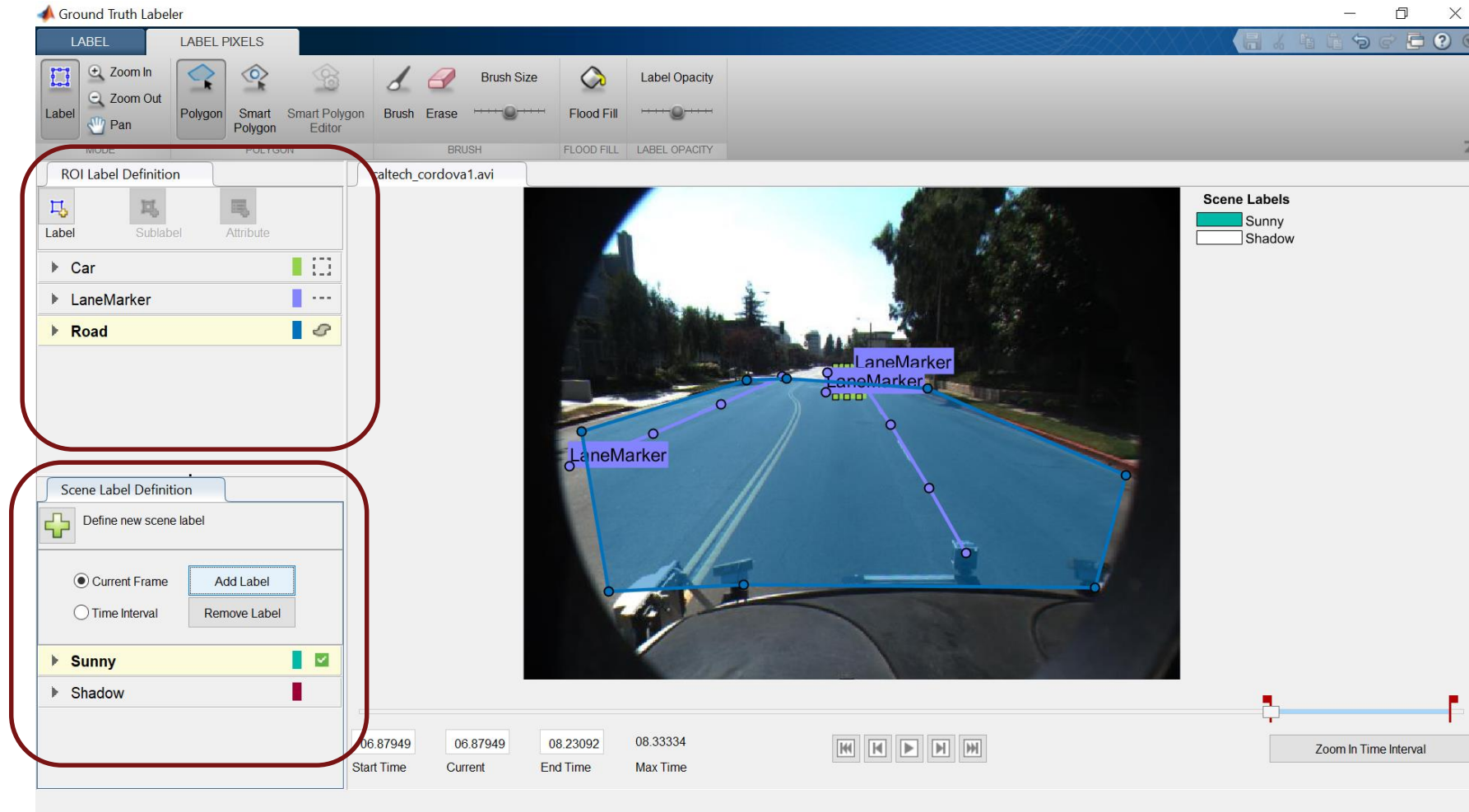
Time	Car	LaneMarker	Sunny	Shadow
0 sec	[2x4 double]	{2x1 cell}	true	false
0.033333 sec	[2x4 double]	{2x1 cell}	true	false
0.066667 sec	[]	[]	false	false

Labels for Training

# Interactive Tools for Ground Truth Labeling

- ROI Labels**
- Bound boxes
  - Pixel labels
  - Poly-lines

**Scene Labels**



# Automate Ground Truth Labeling

The screenshot displays the 'Ground Truth Labeler' application window. The interface is divided into several panels:

- Top Panel:** Contains file operations (Load, Save, Import Labels), a 'Label' tool, and zoom/pan controls.
- Left Panel (ROI Label Definition):** Lists 'Car' and 'LaneMarker' as ROI labels.
- Left Panel (Scene Label Definition):** Includes a 'Define new scene label' button and radio buttons for 'Current Frame' and 'Time Interval', with 'Add Label' and 'Remove Label' buttons.
- Bottom Left Panel:** Lists 'Sunny' and 'Shadow' as scene labels.
- Algorithm List (Center):** A dropdown menu showing 'Temporal Interpolator' selected. Below it are:
  - ACF Vehicle Detector:** Detect vehicles using Aggregate Channel Features (ACF).
  - ACF People Detector:** Detect people using Aggregate Channel Features (ACF).
  - Point Tracker:** Track one or more rectangle ROIs over short intervals using Kanade-Lucas-Tomasi (KLT) algorithm.
  - Temporal Interpolator:** Estimate ROIs in intermediate frames using interpolation of rectangle ROIs in key frames.
  - Buttons:** '+ Add Algorithm' and 'Refresh list'.
- Video View (Center):** Shows a first-person view from a vehicle dashboard. A white box labeled 'Pre-built Automation' is overlaid on the top part of the video, and another white box labeled 'User authored automation' is overlaid on the bottom part.
- Bottom Panel:** Displays time markers: Start Time (06.87949), Current (06.90794), End Time (08.23092), and Max Time (08.33334). It also includes playback controls and a 'Zoom In Time Interval' button.



# Automating Labeling of Lane Markers

The screenshot displays the Ground Truth Labeler software interface. A yellow arrow points to the 'Run automation' button in the top toolbar, with the text 'Run automation algorithm' overlaid on it. The interface is divided into several panels:

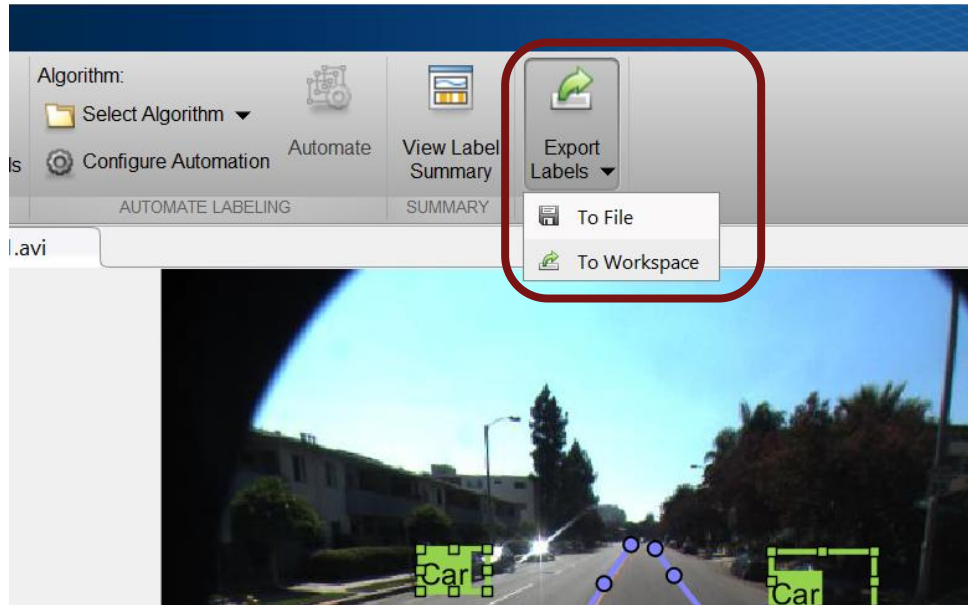
- AUTOMATE**: Contains navigation tools (Label, Zoom In, Zoom Out, Pan) and a 'Run automation' button.
- ROI Label Definition**: Lists labels such as 'laneMarker', 'Road', and 'Sky' with corresponding color swatches.
- Scene Label Definition**: Includes a 'Define new scene label' button and options for 'Current Frame' and 'Time Interval' with 'Add Label' and 'Remove Label' buttons. It also lists 'Sunny' and 'Cloudy' scene labels.
- Auto Lane Detection**: Provides instructions: 'Load a MonoCamera configuration object from the workspace using the settings panel', 'Specify additional parameters in the settings panel', 'Run the algorithm', and 'Manually inspect and modify results if needed'.
- Video Player**: Shows a video frame of a road with lane markers. Below the video is a timeline with 'Start Time' (01.30000), 'Current' (01.30000), 'End Time' (02.47726), and 'Max Time' (08.33334). A 'Zoom Out Time Interval' button is also present.

# Automate Labeling of Bounding Boxes for Vehicles

The screenshot displays the 'Ground Truth Labeler' software interface, specifically the 'AUTOMATE' workflow. The main window shows a video frame from 'caltech\_cordova1.avi' with a bounding box labeled 'Car' on a road scene. The interface is divided into several panels:

- Top Panel (AUTOMATE):** Contains a toolbar with icons for 'Label', 'Zoom In', 'Zoom Out', 'Pan', 'Default Layout', 'Settings', 'Run', 'Stop', 'Undo Run', 'Accept', and 'Cancel'. Below the icons are tabs for 'MODE', 'VIEW', 'SETTINGS', 'RUN', and 'CLOSE'.
- Left Panel (ROI Label Definition):** Includes 'Label', 'Sublabel', and 'Attribute' buttons. A list shows 'Car' (selected) and 'LaneMarker'. Below is the 'Scene Label Definition' section with a 'Define new scene label' button, radio buttons for 'Current Frame' and 'Time Interval', and 'Add Label' and 'Remove Label' buttons. A list shows 'Sunny' and 'Shadow' scene labels.
- Right Panel (Temporal Interpolator):** Contains instructions for 'ROI Selection', 'Run', 'Review and Modify', 'Undo Run', and 'Accept/Cancel'.
- Bottom Panel:** Features a timeline with 'Start Time' (06.87949), 'Current' (06.87949), 'End Time' (08.23092), and 'Max Time' (08.33334). It also includes playback controls and a 'Zoom Out Time Interval' button.

# Export Labeled Data for Training



```
>> gTruth
gTruth =
    groundTruth with properties:
        DataSource: [1x1 groundTruthDataSource]
        LabelDefinitions: [4x3 table]
        LabelData: [250x4 timetable]

>> gTruth.LabelData
ans =
    250x4 timetable
```

Time	Car	LaneMarker	Sunny	Shadow
0 sec	[2x4 double]	{2x1 cell}	true	false
0.033333 sec	[2x4 double]	{2x1 cell}	true	false
0.066667 sec	[]	[]	false	false

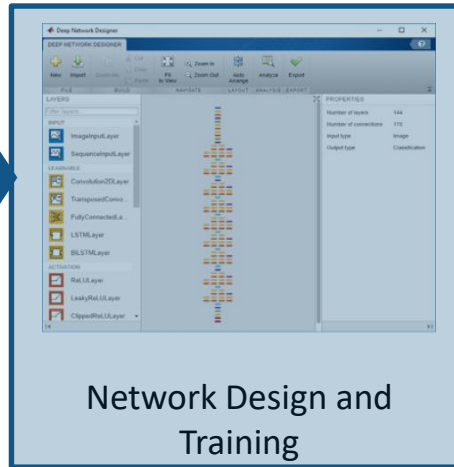
Bounding  
Boxes Labels

Polyline Labels

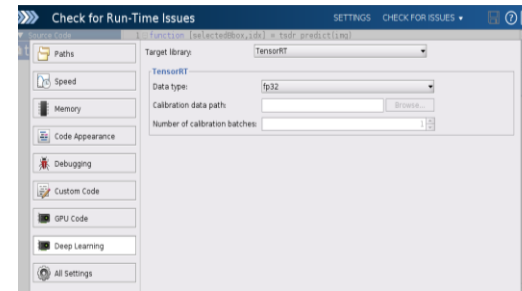
# Outline



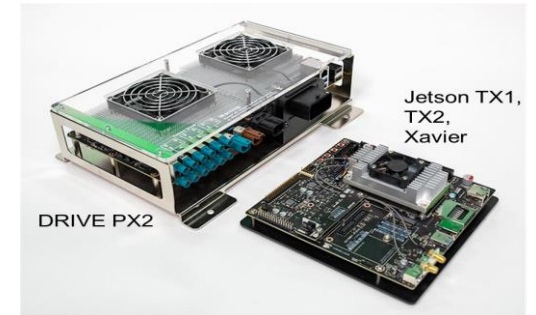
Ground Truth Labeling



Network Design and Training

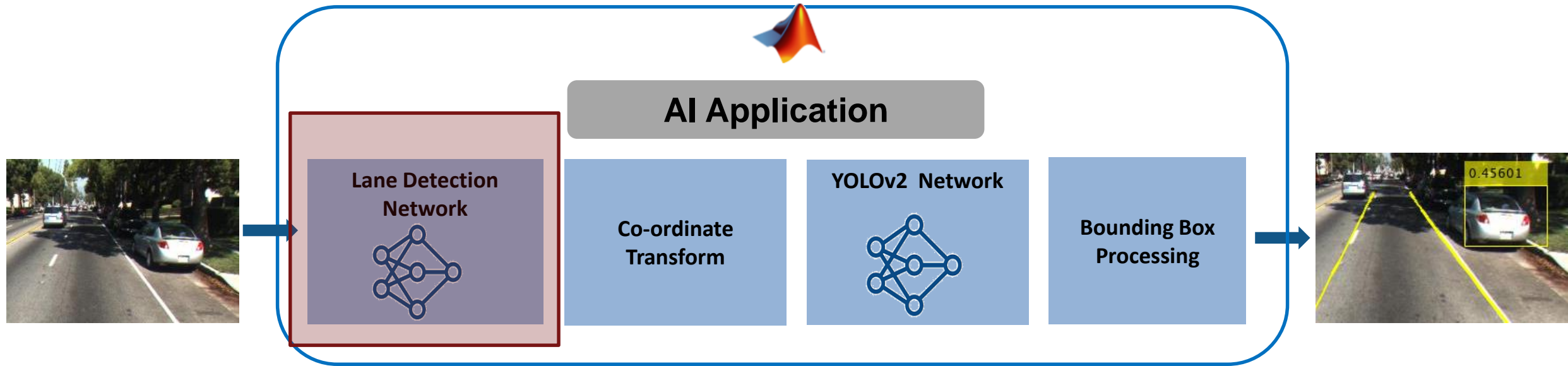


CUDA and TensorRT Code Generation



Jetson Xavier and DRIVE Xavier Targeting

# Example Used in Today's Talk



# Lane Detection Algorithm

Pretrained Network  
(E.g. AlexNet)



Modify Network for  
Lane Detection



Coefficients of parabola



Transform to  
Image Coordinates



```

regressionOutputs =
1225x6 table
    leftLane_a    leftLane_b    leftLane_c    rightLane_a    rightLane_b    rightLane_c
    _____    _____    _____    _____    _____    _____
    3.5482e-05     0.0060327     1.7599        -0.00015691    0.030256      -2.0559
    -3.9519e-05    0.014116      1.662         -0.00097636    0.02979       -2.0749
    -6.778e-07     -0.00063158    1.776         -7.0963e-05    0.0024721     -1.9428
    -0.00023646    0.0088324     1.8188        -0.00050391    -0.0015166    -1.973
    -0.00055867    0.012996      1.8074        -8.6643e-05    0.00098652    -1.935
    ...

```

# Lane Detection: Load Pretrained Network

Lane Boundaries in Image Coordinates



```
>> net = alexnet  
>> deepNetworkDesigner
```

## Lane Detection Network

- Regression CNN for lane parameters
- MATLAB code to transform to image co-ordinates

# View Network in Deep Network Designer App

The screenshot displays the Deep Network Designer interface. The main workspace shows a vertical sequence of layers: dropout6 (DropoutLayer), fc7 (FullyConnected...), relu7 (ReLU Layer), dropout7 (DropoutLayer), fc8 (FullyConnected...), prob (SoftmaxLayer), and output (ClassificationO...). The left sidebar contains a 'LAYERS' panel with categories: INPUT (ImageInputLayer, SequenceInputLayer), LEARNABLE (Convolution2DLayer, TransposedConvolution2DLayer, FullyConnectedLayer, LSTMLayer, BiLSTMLayer), ACTIVATION (ReLU Layer, LeakyReLU Layer, ClippedReLU Layer), and NORMALIZATION AND DROPOUT (BatchNormalizationLayer). The right sidebar shows the 'PROPERTIES' panel for the selected layer, listing: Number of layers: 25, Number of connections: 24, Input type: Image, and Output type: Classification.



# Remove Layers from AlexNet

Deep Network Designer

DEEP NETWORK DESIGNER

FILE BUILD NAVIGATE LAYOUT ANALYSIS EXPORT

Layers

Filter layers...

INPUT

- ImageInputLayer
- SequenceInputLayer

LEARNABLE

- Convolution2DLayer
- TransposedConvolution2DLayer
- FullyConnectedLayer
- LSTMLayer
- BiLSTMLayer

ACTIVATION

- ReLULayer
- LeakyReLULayer
- ClippedReLULayer

NORMALIZATION AND DROPOUT

- BatchNormalizationLayer

Properties

Number of layers	25
Number of connections	24
Input type	Image
Output type	Classification

# Add Regression Output for Lane Parameters

The screenshot shows the Deep Network Designer interface with a neural network architecture. The layers are:

- drop6 DropoutLayer
- fcLane1 FullyConnected...
- relu7 ReLULayer
- fcLane1Relu FullyConnected...
- relu ReLULayer
- fcLane2 FullyConnected...
- regressionout... RegressionOut...

The regression output layer is highlighted with a red box. A red text overlay reads: **Regression Output for Lane Coefficients**.

The Properties panel on the right shows the following details:

PROPERTIES	
Number of layers	25
Number of connections	24
Input type	Image
Output type	Regression

# Transparently Scale Compute for Training

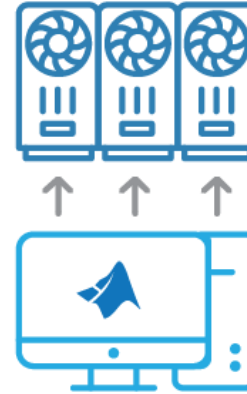
## Specify Training on:



'CPU'



'gpu'



'multi-gpu'

Works on Windows  
(no additional setup)

Quickly change training hardware

```
opts = trainingOptions('sgdm', ...  
    'BatchSize', 100, ...  
    'MiniBatchSize', 250, ...  
    'InitialLearnRate', 0.00005, ...  
    'ExecutionEnvironment', 'auto');
```

# NVIDIA NGC & DGX Supports MATLAB for Deep Learning

- GPU-accelerated MATLAB Docker container for deep learning
  - Leverage multiple GPUs on NVIDIA DGX Systems and in the Cloud
    - Cloud providers include: AWS, Azure, Google, Oracle, and Alibaba
- NVIDIA DGX System / Station
  - Interconnects 4/8/16 Volta GPUs in one box
- Containers available for R2018a and R2018b
  - New Docker container with every major release (a/b)
- Download MATLAB container from NGC Registry
  - <https://ngc.nvidia.com/registry/partners-matlab>



# Evaluate Lane Boundary Detections vs. Ground Truth

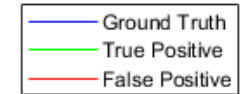
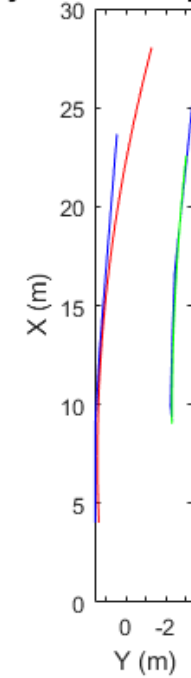
Sample Ground Truth Data for Left Lane Boundary



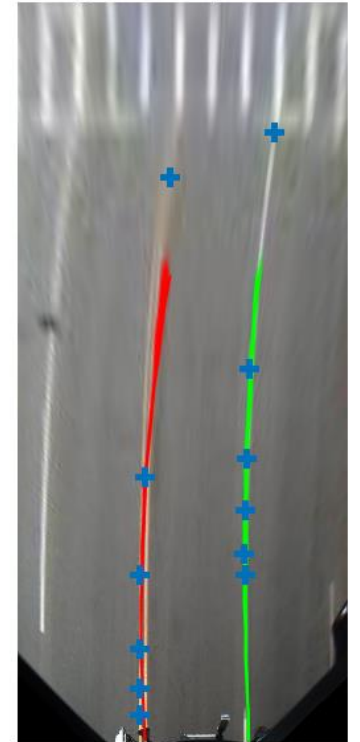
`evaluateLaneBoundaries`



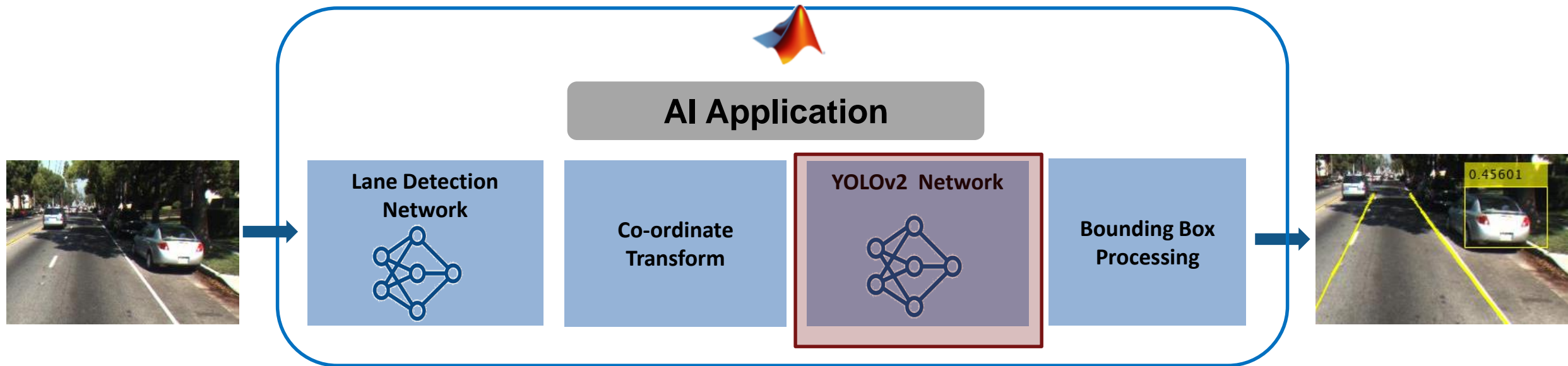
Bird's-Eye Plot of Comparison Results



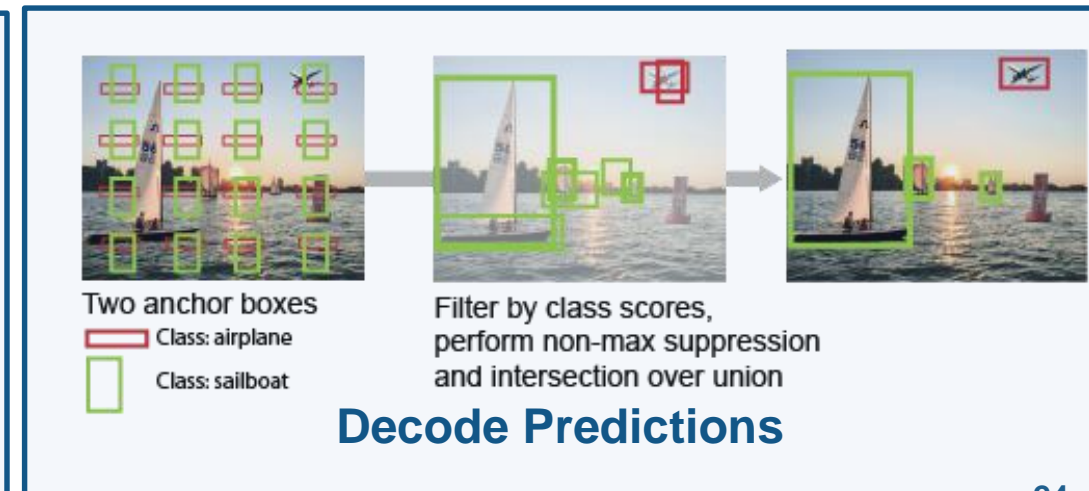
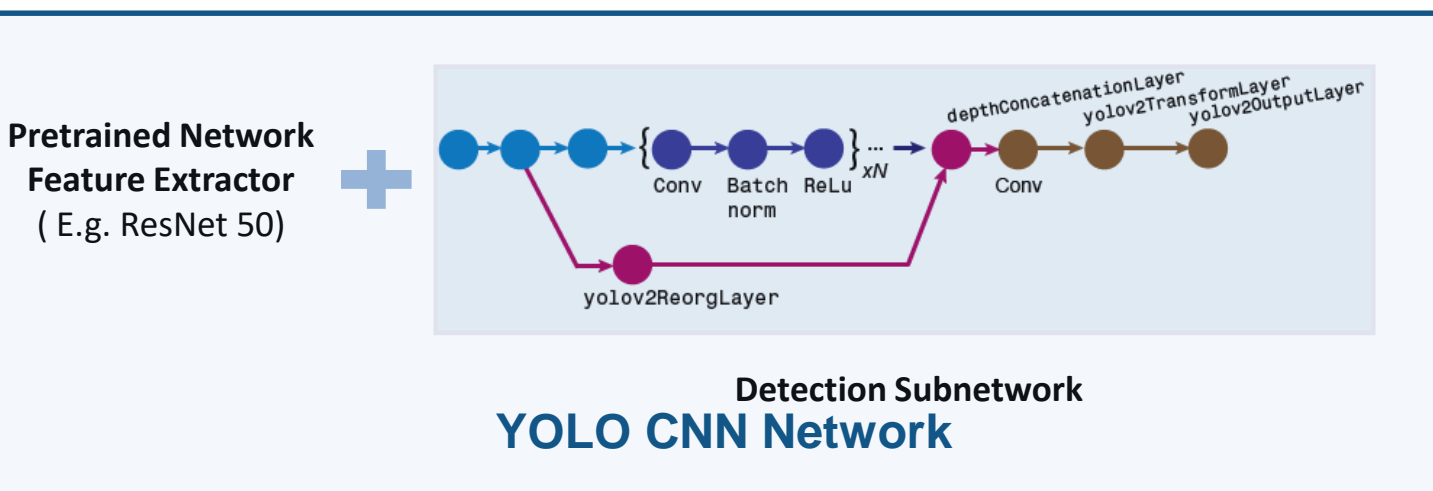
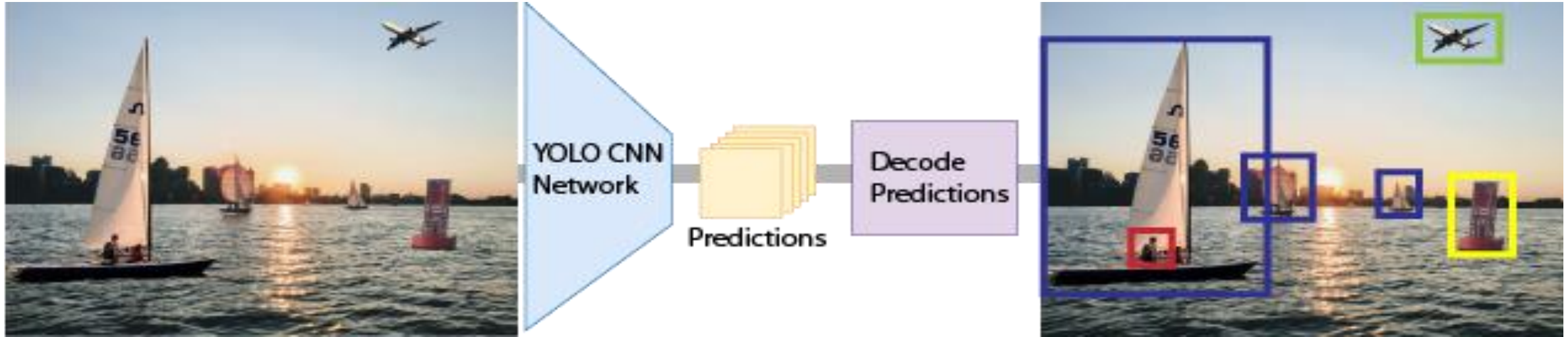
Bird's-Eye View of Comparison Results



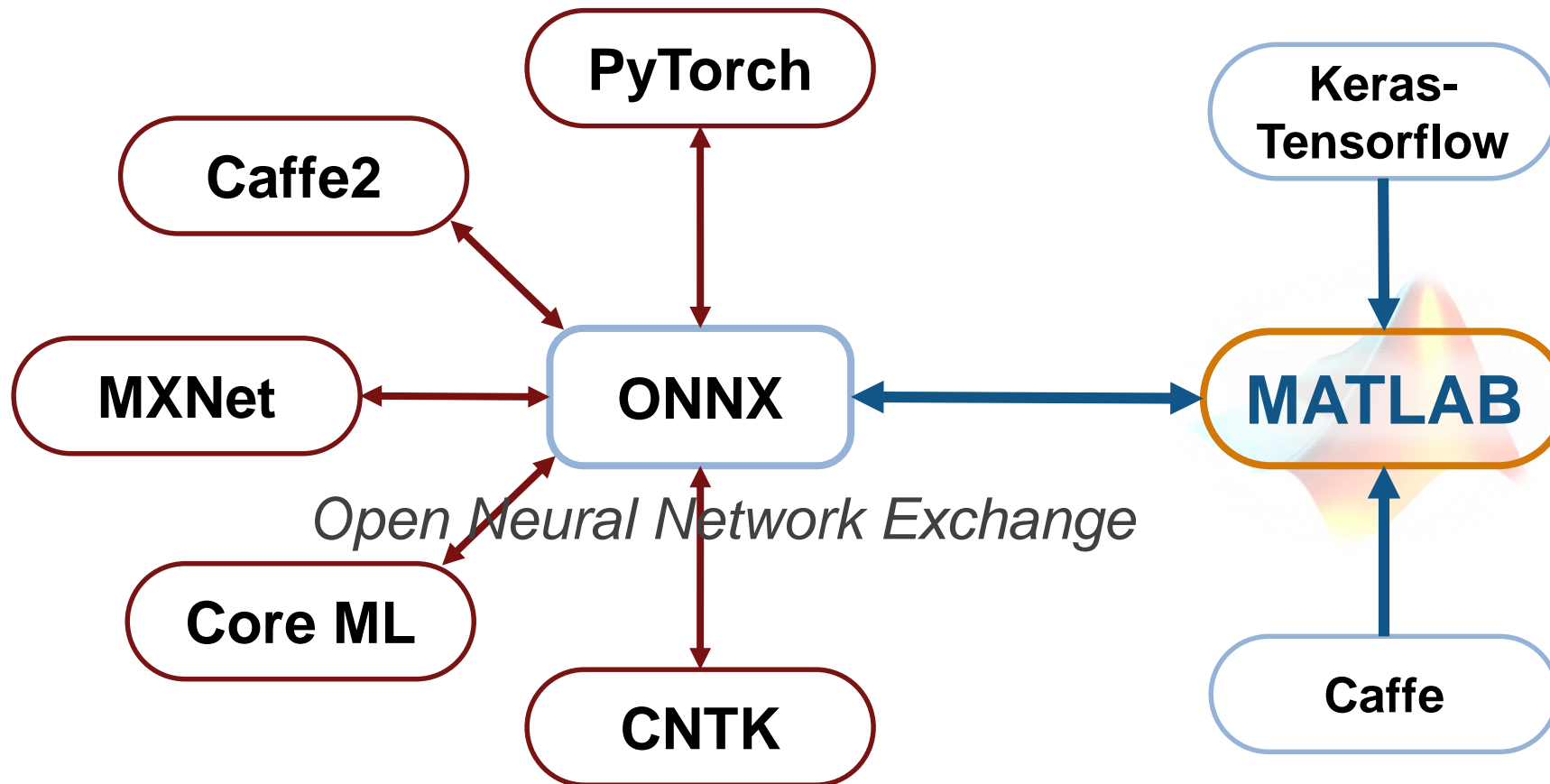
# Example Used in Today's Talk



# YOLO v2 Object Detection



# Model Exchange with MATLAB





# Import Pretrained Network in ONNX Format

```
load resnetClassNames.mat
net = importONNXNetwork('resnet50.onnx', ...
    'OutputLayerType', 'classification', ...
    'ClassNames', classnames);
analyzeNetwork(net)
```

# Import Pretrained Network in ONNX Format

resnet50

Analysis date: 09-Jan-2019 09:39:08

177 layers

0 warnings

0 errors

ANALYSIS RESULT				
	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	input_1 224x224x3 images with 'zerocenter' normalization	Image Input	224x224x3	-
2	conv1 64 7x7x3 convolutions with stride [2 2] and padding [3 3 3 3]	Convolution	112x112x64	Weights 7x7x3x64 Bias 1x1x64
3	bn_conv1 Batch normalization with 64 channels	Batch Normalization	112x112x64	Offset 1x1x64 Scale 1x1x64
4	activation_1_relu ReLU	ReLU	112x112x64	-
5	max_pooling2d_1 3x3 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	55x55x64	-
6	res2a_branch2a 64 1x1x64 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	55x55x64	Weights 1x1x64x64 Bias 1x1x64
7	bn2a_branch2a Batch normalization with 64 channels	Batch Normalization	55x55x64	Offset 1x1x64 Scale 1x1x64
8	activation_2_relu ReLU	ReLU	55x55x64	-
9	res2a_branch2b 64 3x3x64 convolutions with stride [1 1] and padding 'same'	Convolution	55x55x64	Weights 3x3x64x64 Bias 1x1x64
10	bn2a_branch2b Batch normalization with 64 channels	Batch Normalization	55x55x64	Offset 1x1x64 Scale 1x1x64
11	activation_3_relu ReLU	ReLU	55x55x64	-
12	res2a_branch2c 256 1x1x64 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	55x55x256	Weights 1x1x64x256 Bias 1x1x256
13	res2a_branch1 256 1x1x64 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	55x55x256	Weights 1x1x64x256 Bias 1x1x256
14	bn2a_branch2c	Batch Normalization	55x55x256	Offset 1x1x256

# Modify Network

```
lgraph = layerGraph(net);  
lgraph = removeLayers(lgraph, 'Input_input_1');  
lgraph = removeLayers(lgraph, 'fc1000_Flatten1');  
lgraph = connectLayers(lgraph, 'avg_pool', 'fc1000');
```

Removing the 2  
ResNet-50 layers

```
avgImgBias = -1*(lgraph.Layers(1).Bias);
```

```
%Create new input layer and incorporate average image bias
```

```
larray = imageInputLayer([224 224 3],...  
    'Name','input',...  
    'AverageImage',avgImgBias);
```

`imageInputLayer` replaces the  
input and subtraction layer

```
lgraph = replaceLayer(lgraph, 'input_1_Sub', larray);
```

```
netModified = assembleNetwork(lgraph);
```

```
save('resnet50_model.mat', 'netModified');
```

Save MAT file for code gen

# YOLOv2 Detection Network

- **yoloV2Layers**: Create network architecture

```
>> lgraph = yoloV2Layers(imageSize, numClasses, anchorBoxes, network, featureLayer)
```

↑  
Number of  
Classes



Two anchor boxes

— Class: airplane

□ Class: sailboat

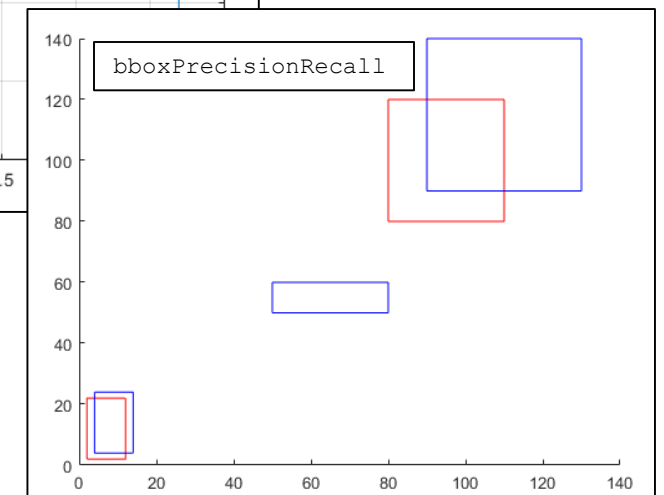
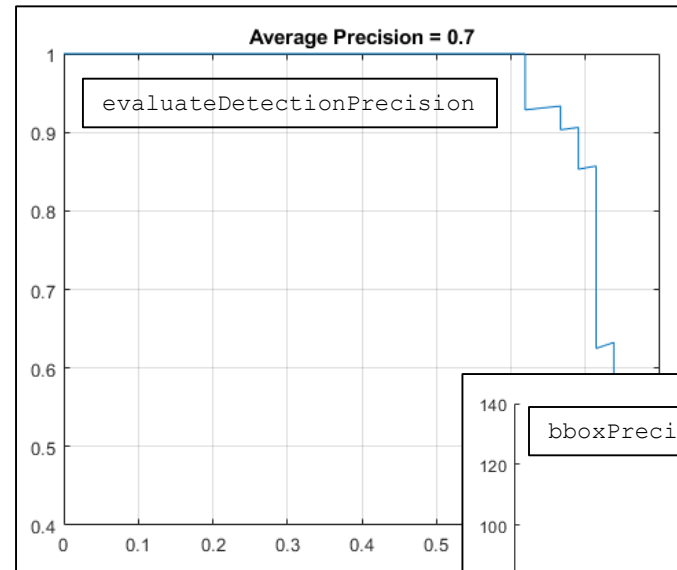
↑  
Pretrained  
Feature Extractor

```
>> detector = trainYOLOv2ObjectDetector(trainingData, lgraph, options)
```

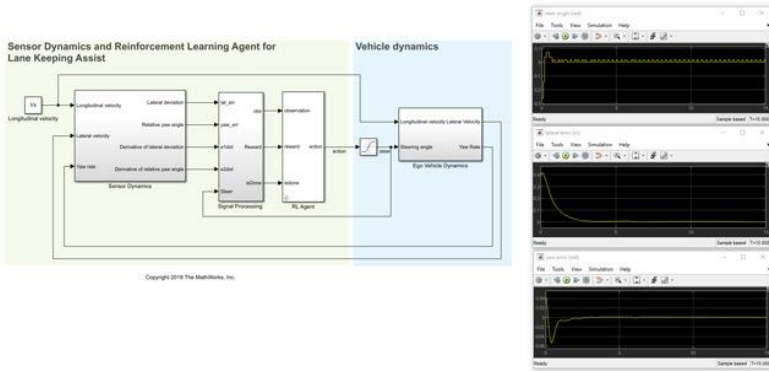
# Evaluate Performance of Trained Network

- **Set of functions** to evaluate trained network performance
  - evaluateDetectionMissRate
  - **evaluateDetectionPrecision**
  - bboxPrecisionRecall
  - bboxOverlapRatio

```
>> [ap, recall, precision] =  
evaluateDetectionPrecision(results, vehicles(:, 2));
```



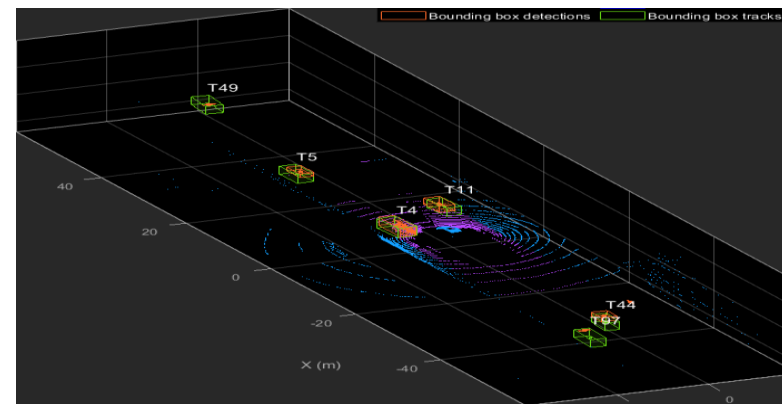
# Example Applications using MATLAB for AI Development



**Lane Keeping Assist using Reinforcement Learning**

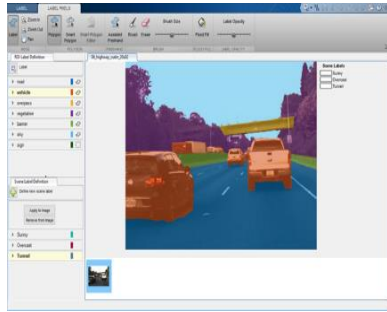


**Occupancy Grid Creation using Deep Learning**

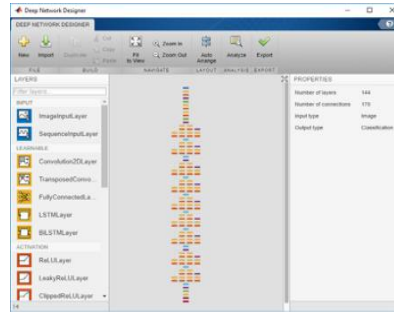


**Lidar Segmentation with Deep Learning**

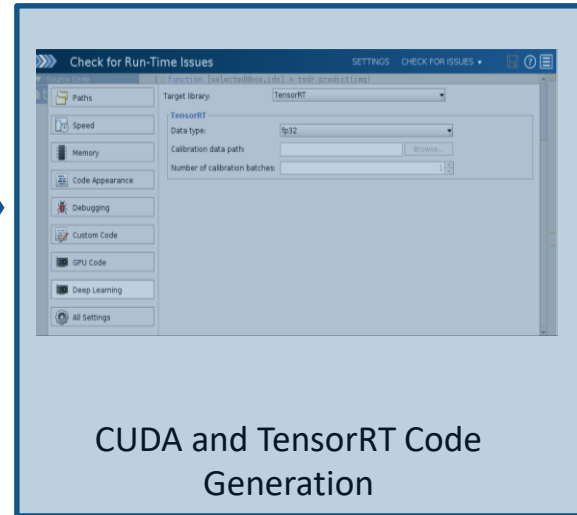
# Outline



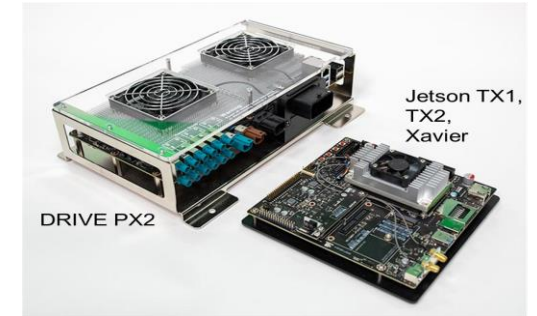
Ground Truth Labeling



Network Design and Training



CUDA and TensorRT Code Generation



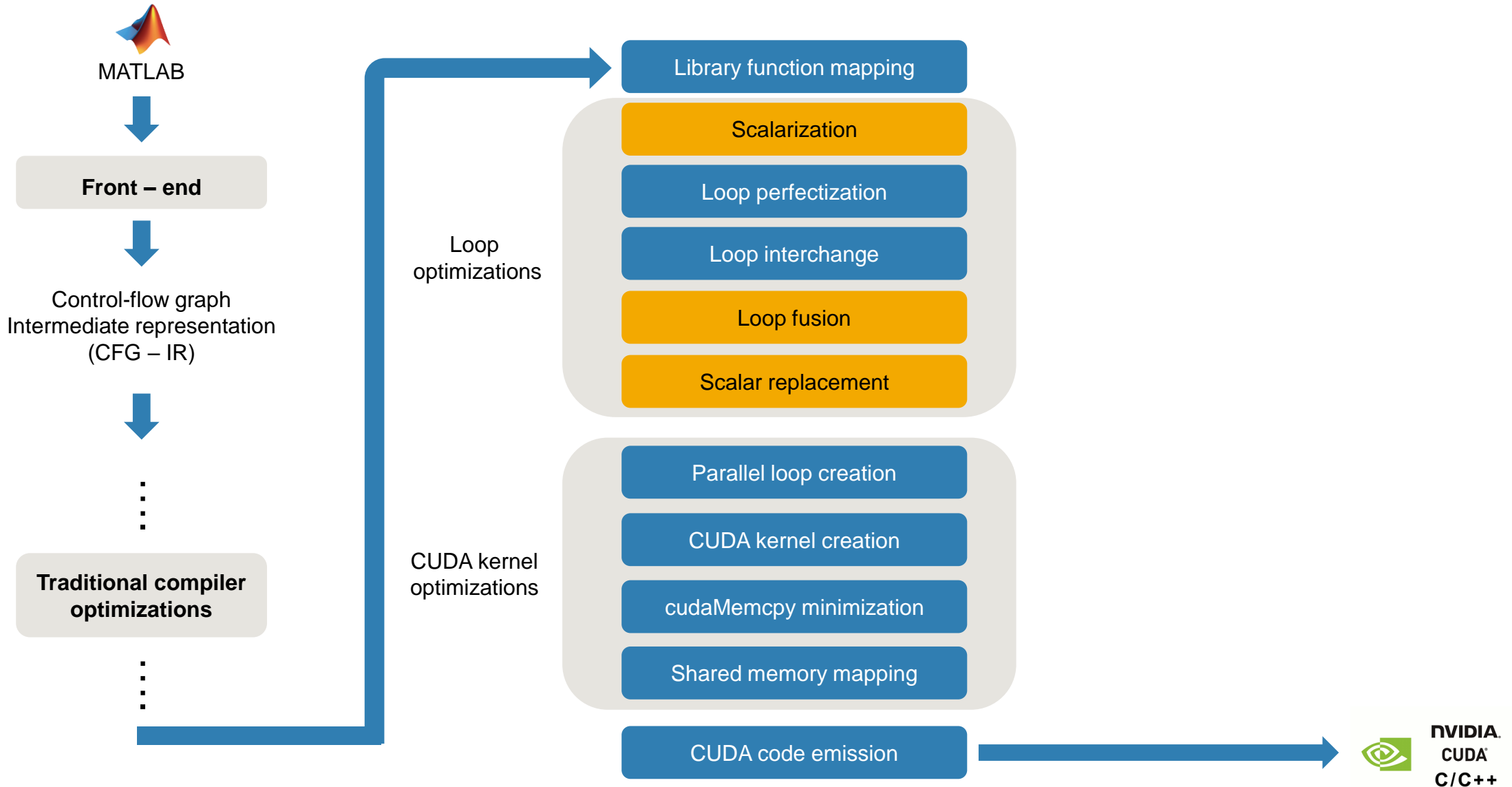
Jetson Xavier and DRIVE Xavier Targeting

## Key Takeaways

**Platform Productivity:** Workflow automation, ease of use

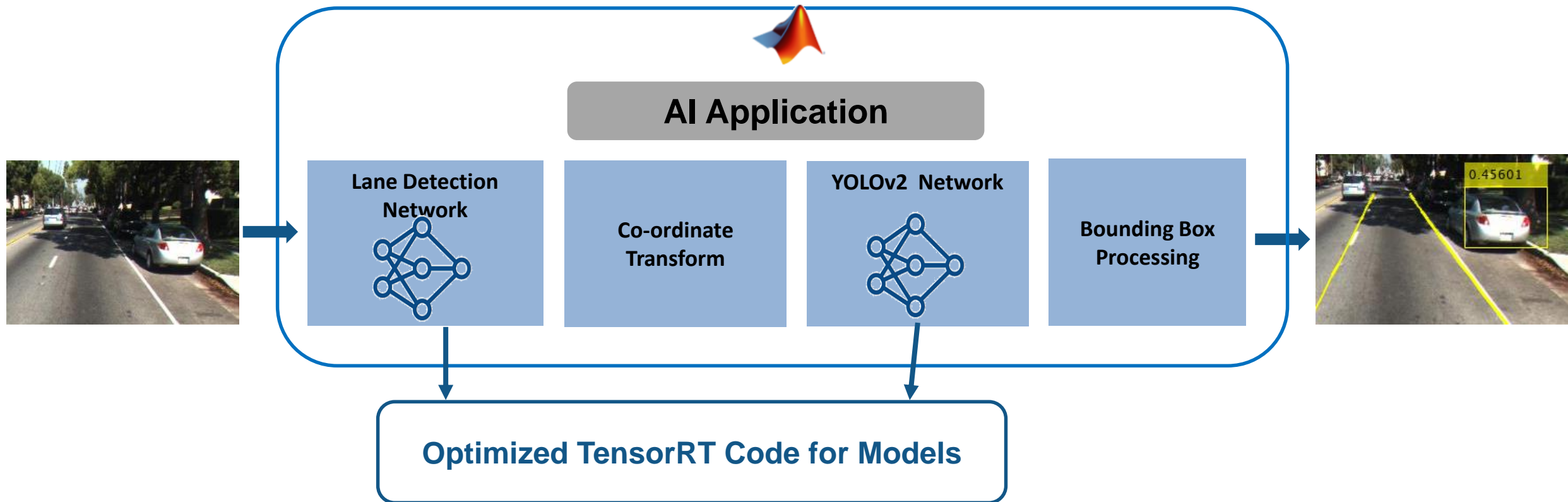
**Framework Interoperability:** ONNX, Keras-TensorFlow, Caffe

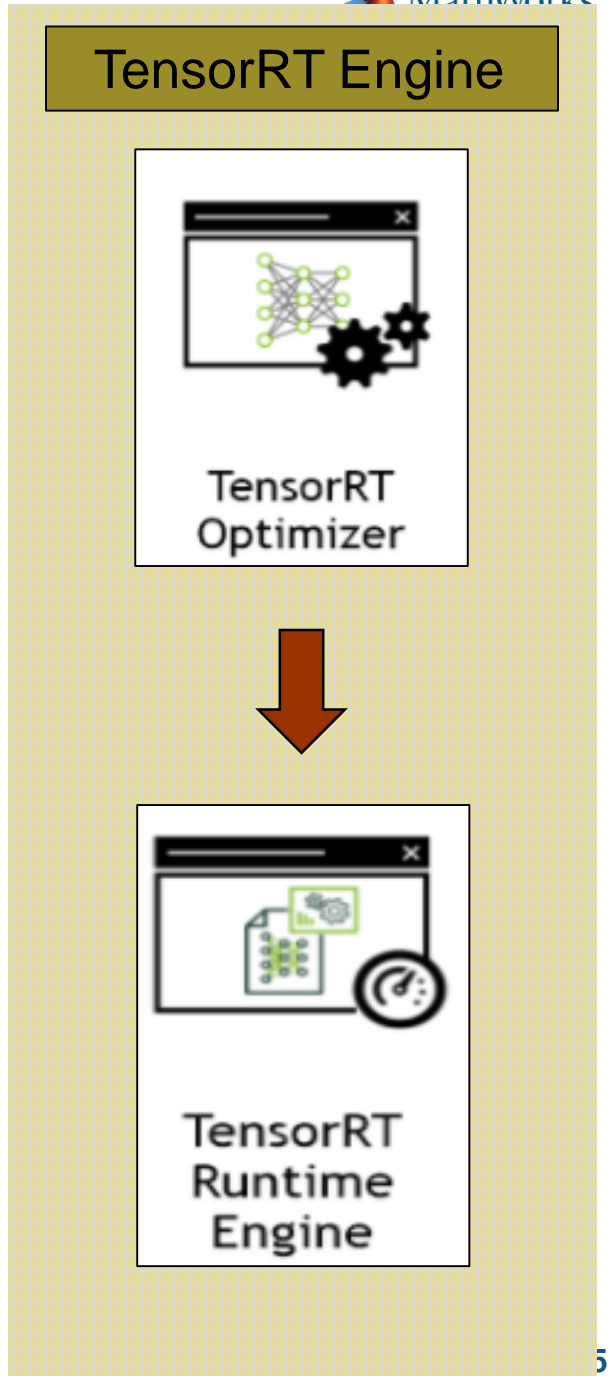
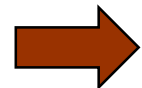
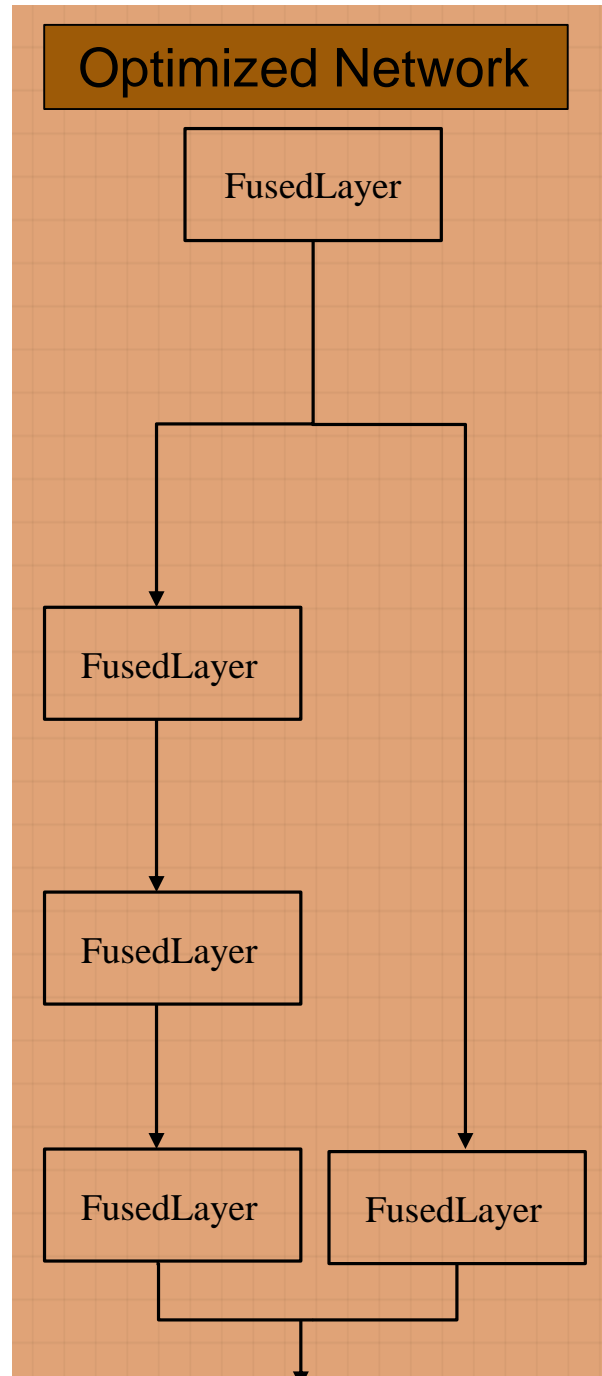
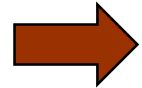
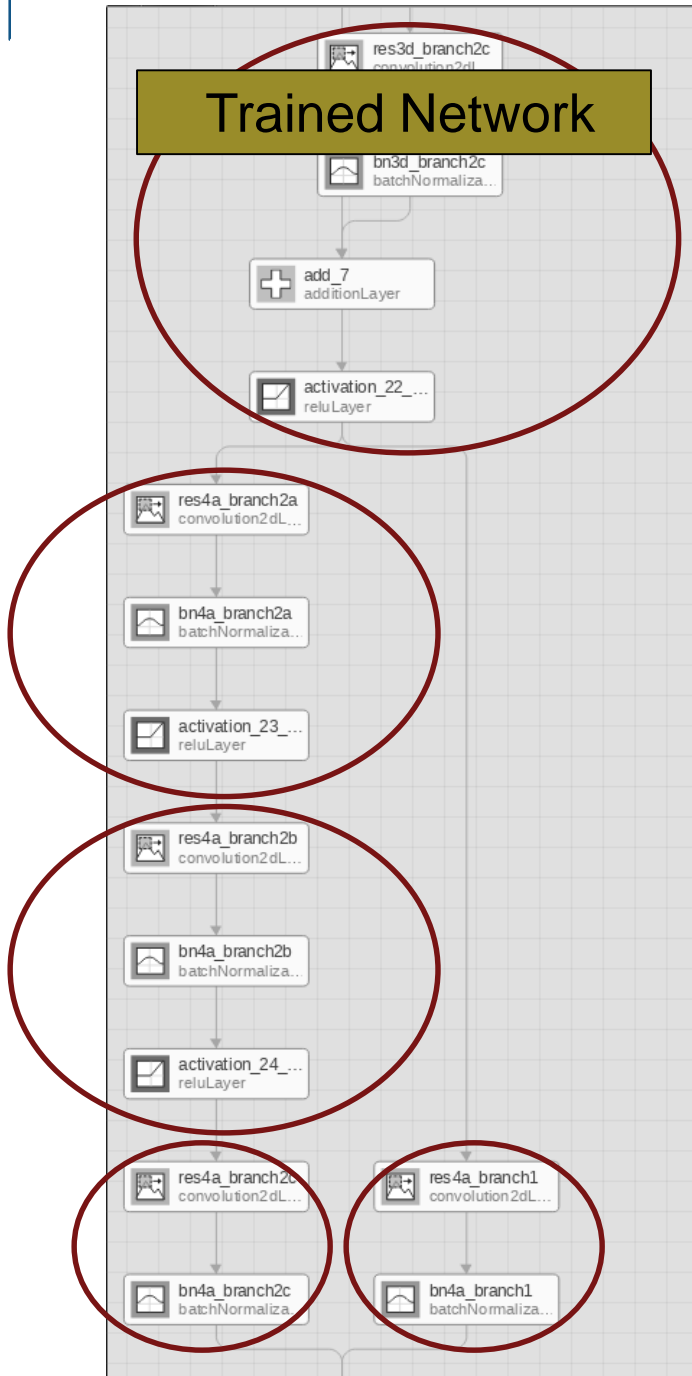
# GPU Coder runs a host of compiler transforms to generate CUDA





# Example Used in Today's Talk





HOME PLOTS APPS EDITOR PUBLISH VIEW

New Open Save Find Files Compare Go To Find Insert Comment Indent Breakpoints Run Run and Advance Run Section Advance Run and Time

FILE NAVIGATE EDIT BREAKPOINTS RUN

/ > mathworks > devel > sandbox > jshankar > GTC2019 > demofolder > demo\_files >

Current Folder

Name

- codegen
- caltech\_washington1.avi
- lane\_and\_vehicleDetection.m
- lane\_yolo.m
- LaneDetectionNet.mat
- VehicleDetectorNet.mat

Editor - /mathworks/devel/sandbox/jshankar/GTC2019/demofolder/demo\_files/lane\_yolo.m

```

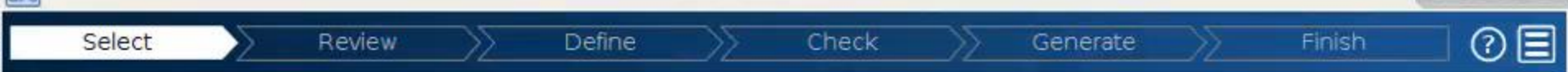
1 function Out = lane_yolo(In)
2 % The regression network is trained to detect parameters of lane parabola
3 % The outputs are unnormalized and converted to left and right lane points
4 % in image coordinates.
5 % The camera coordinates are described by the caltech mono camera model.
6
7 %#codegen
8
9 frame = imresize(In, [227,227]);
10
11 persistent lanenet;
12 if isempty(lanenet)
13     lanenet = coder.loadDeepLearningNetwork('LaneDetectionNet.mat', 'Lanenet');
14 end
15
16 lanecoefsNetworkOutput = lanenet.predict(frame);
17
18 % Recover original coeffs by reversing the normalization steps
19 laneCoeffMeans = [-.0002, .0002, 1.4740, -.0002, .0045, -1.3787];
20 laneCoeffStds = [.0030, .0766, .6313, .0026, .0736, .9946];
21 params = lanecoefsNetworkOutput .* laneCoeffStds + laneCoeffMeans;
22
23 % should be more than 0.5 for it to be a lane
24 isRightLaneFound = abs(params(6)) > 0.5;
25 isLeftLaneFound = abs(params(3)) > 0.5;
26
27 vehicleXPoints = 3:30; %meters, ahead of the sensor
28 ltPts = coder.nullcopy(zeros(28,2,'single'));
29 rtPts = coder.nullcopy(zeros(28,2,'single'));
30
31 % map vehicle to image coordinates
32 if isRightLaneFound && isLeftLaneFound
33

```

Command Window

New to MATLAB? See resources for [Getting Started](#).

>>



- 1
- 2
- 3 -
- 4 -
- 5 -
- 6 -
- 7
- 8 -
- 9
- 10
- 11 -
- 12 -
- 13 -
- 14
- 15
- 16 -
- 17 -
- 18
- 19
- 20 -
- 21 -
- 22 -
- 23 -
- 24
- 25 -

# GPU Coder

The GPU Coder workflow generates CUDA code. **To begin, select your entry-point function(s).**

Generate code for function:  ...

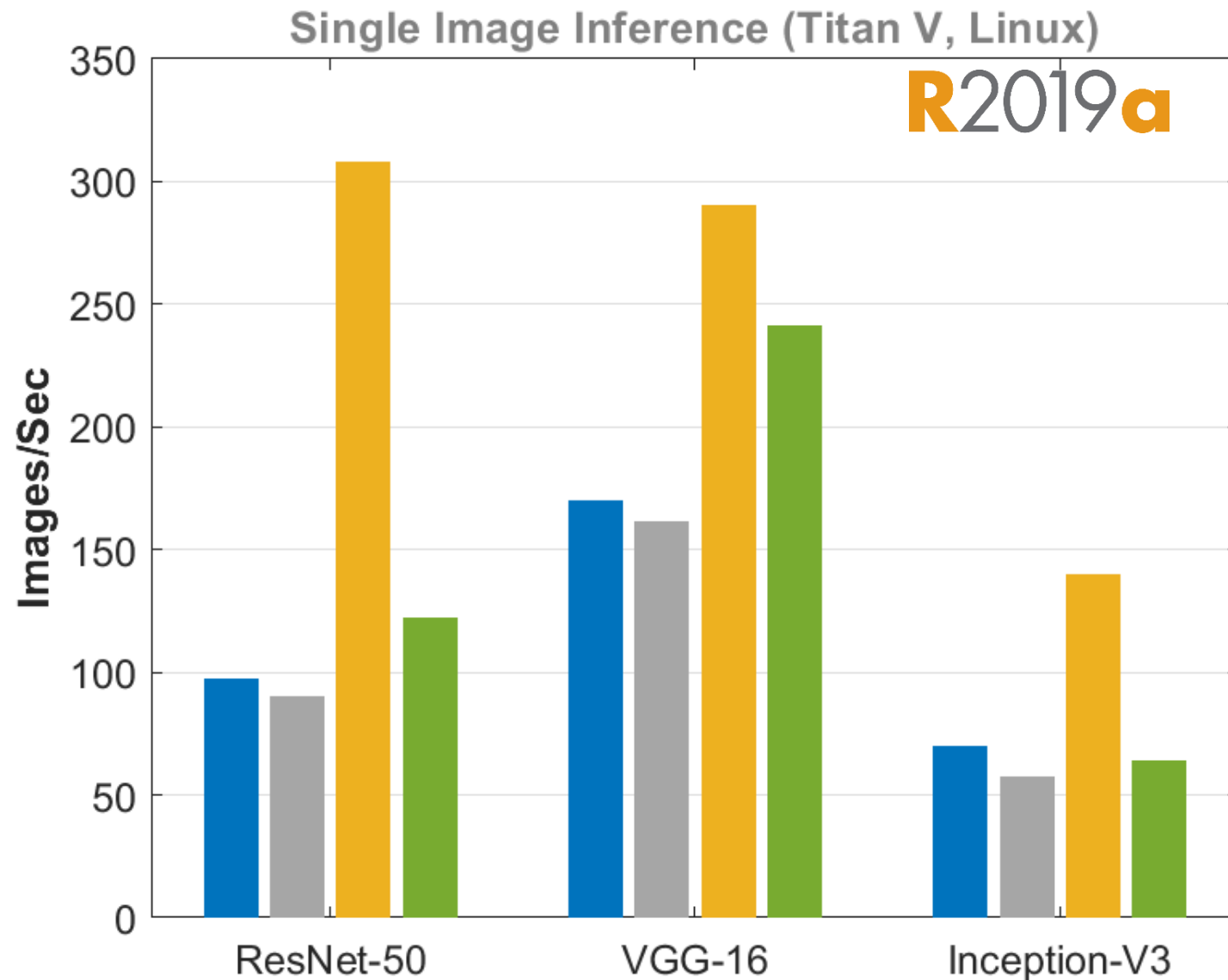
Comr

New f

>>

f<sub>x</sub> >>

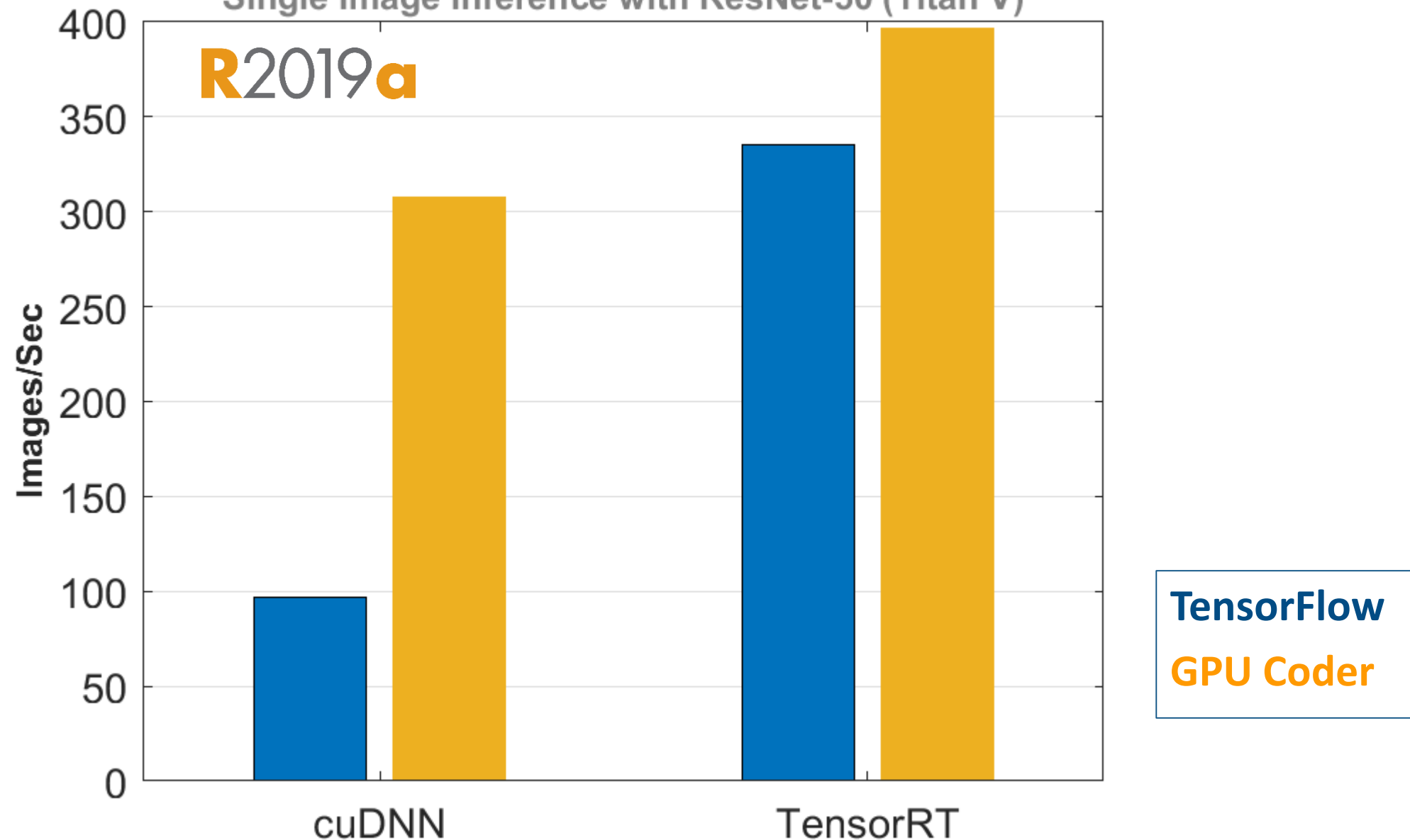
## With GPU Coder, MATLAB is fast



**Faster than TensorFlow,  
MXNet, and PyTorch**

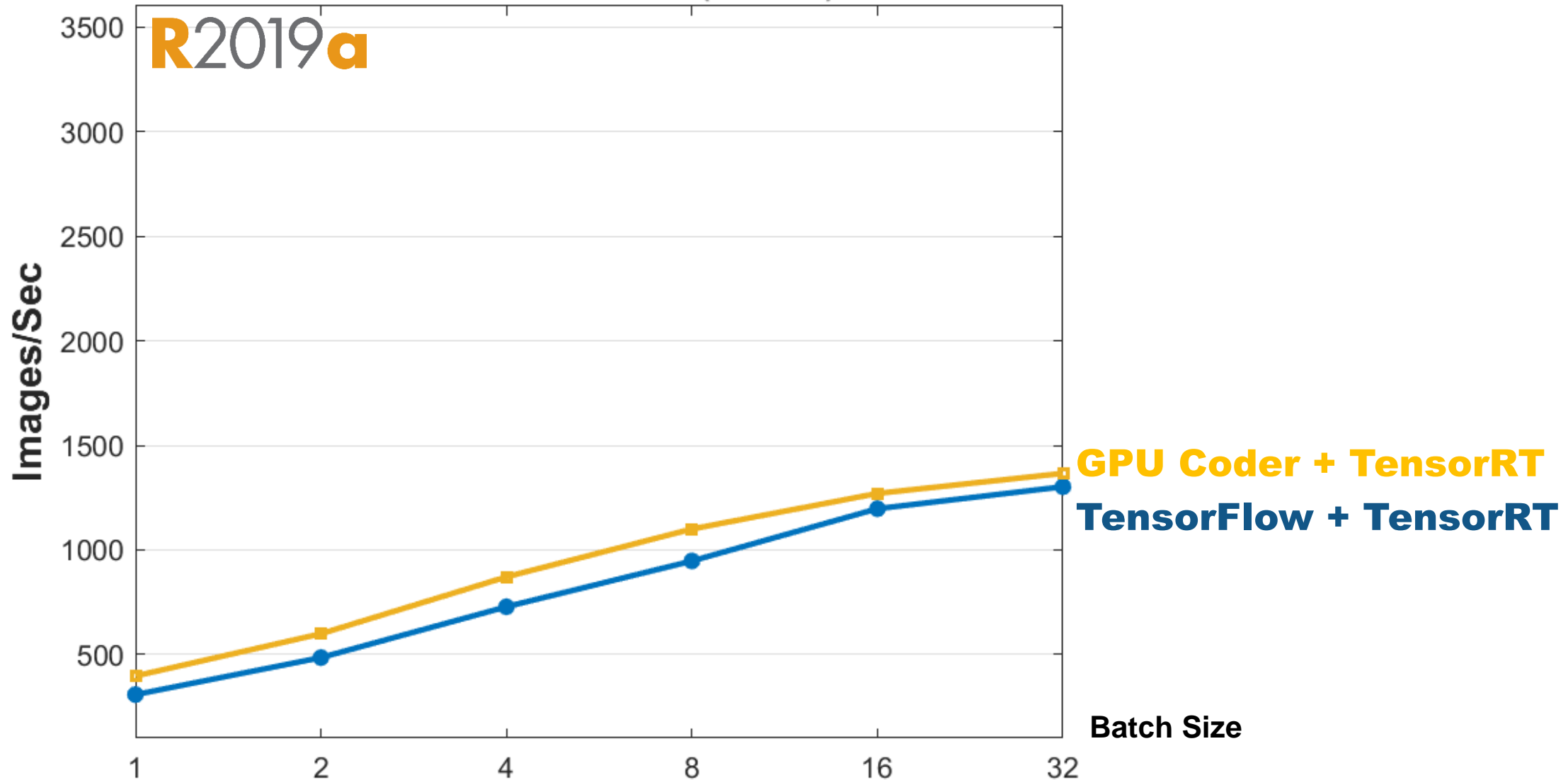
# TensorRT speeds up inference for TensorFlow and GPU Coder

Single Image Inference with ResNet-50 (Titan V)



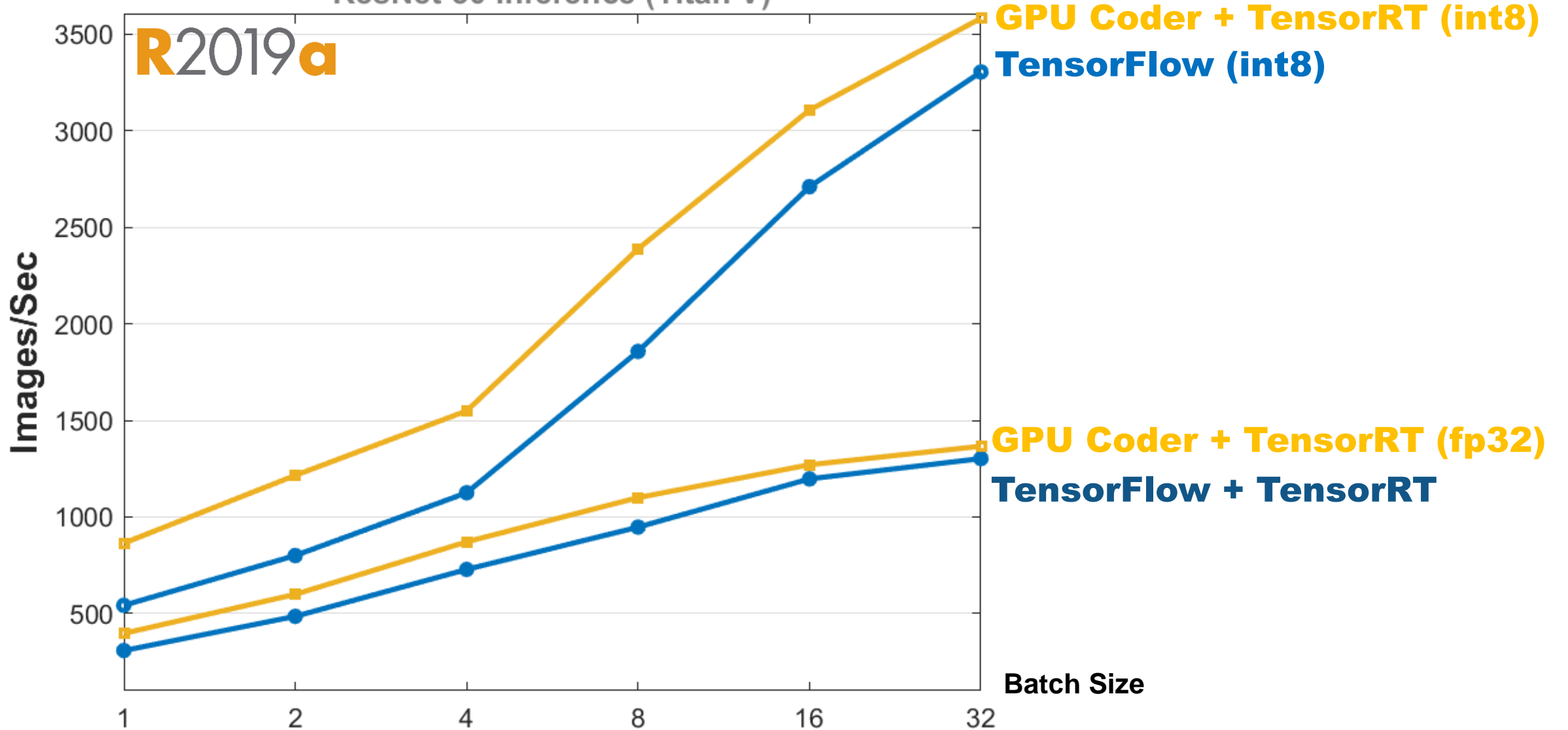
# GPU Coder with TensorRT faster across various Batch Sizes

ResNet-50 Inference (Titan V)



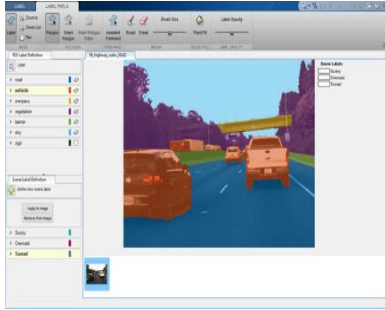
# Even higher Speeds with Integer Arithmetic (int8)

ResNet-50 Inference (Titan V)

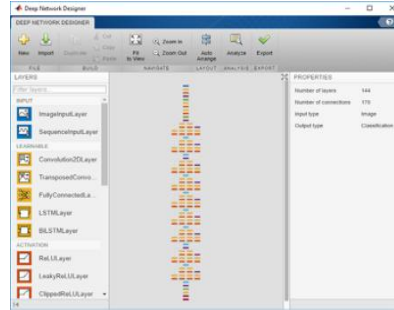




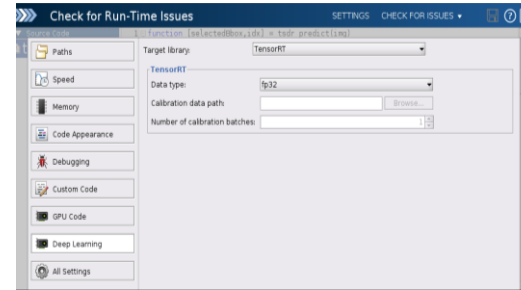
# Outline



Ground Truth Labeling



Network Design and Training



CUDA and TensorRT Code Generation



Jetson Xavier and DRIVE Xavier Targeting

**Key Takeaways**  
**Optimized CUDA and TensorRT code generation**

# Deploy to Jetson and Drive

```

%_main.m: Primary function. This
% function generates the GPU code
% to be used on the target.
%
% The main code is in the
% function main.m.
%
% Copyright 2014 MathWorks, Inc.
% All rights reserved.
%
% 2014-01, P. Rodriguez
  
```

MATLAB algorithm  
(functional reference)

GPU Coder

Build type

Call compiled  
application from  
MATLAB directly

.mex

Call compiled  
application  
from hand-coded  
main()

.lib

Deploy to target  
and run with  
hardware-in-loop

Deploy to target

Desktop  
GPU

Desktop  
GPU

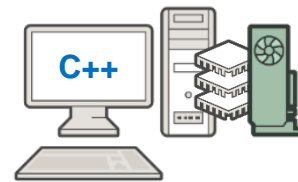
Embedded GPU

1 Functional test

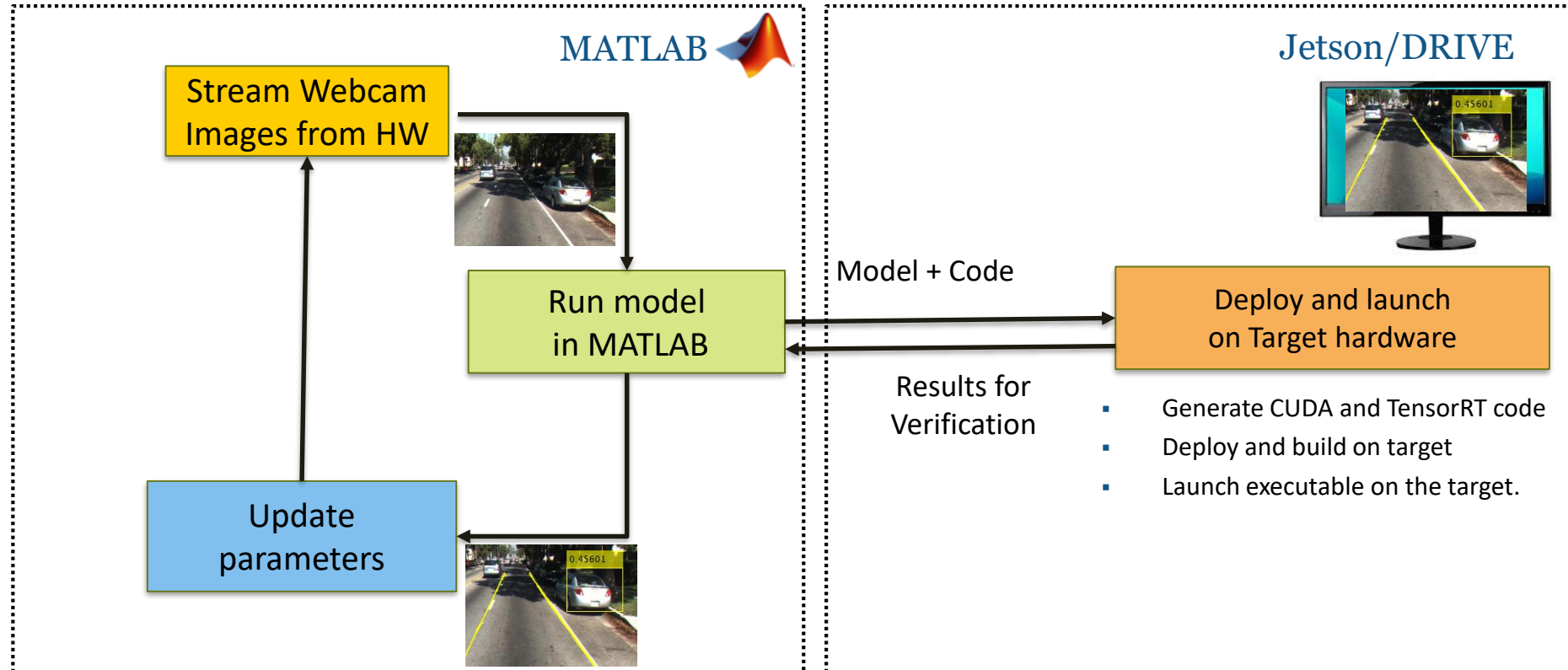
2 Deployment  
unit-test

3 Deployment  
integration-test

4 Real-time test




# Hardware in the loop workflow with Jetson/DRIVE device





GPU Coder - lane\_yolo.prj

Generate Code GENERATE VERIFY CODE

Build type:  

Output file name:

Language  C  C++

 More Settings  Generate

```

25 /* Type Definitions */
26 typedef struct {
27     b_VehicleDetectorNet_0 *net;
28 } coder_YOLOv2Network;
29
30 struct emxArray_int32_T_4
31 {
32     int32_T data[4];
33     int32_T size[1];
34 };
35
    
```

Target Build Log Variables

Variable	Type	Size
----------	------	------

```
lane_yolo.m x lane_and_vehicleDetection.m x +
1 function lane_and_vehicleDetection
2
3     videoFileReader = VideoReader('caltech_washington1.avi');
4     depVideoPlayer = vision.DeployableVideoPlayer('Name', 'simulation');
5     fps = 0;
6     while hasFrame(videoFileReader)
7         % grab frame from video
8         I = readFrame(videoFileReader);
9
10        % Run the detector on the input test image
11        tic;
12        sim_frame = lane_yolo_mex(I);
13        mltime = toc;
14
15        % Calculate fps
```

## Command Window

New to MATLAB? See resources for [Getting Started](#).

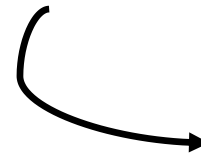
```
f_x >> h|
```

# Processor in the loop verification with Jetson/Drive devices

```
% Set up connection to Jetson device  
hwobj = jetson('gpcoder-xavier-1','ubuntu','ubuntu');
```

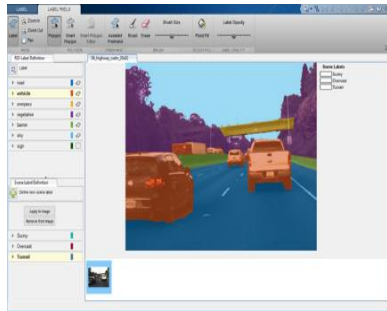
```
% Set up code generation to Processor-in-loop mode  
cfg = coder.gpuConfig('lib');  
cfg.VerificationMode = 'PIL';  
cfg.Hardware = coder.hardware('NVIDIA Jetson');
```

```
% Generate code for application using CUDA and TensorRT  
cfg.DeepLearningConfig = coder.DeepLearningConfig('tensorrt');  
codegen -config cfg detect_lane_yolo_full -args {ones(480,640,3,'uint8')}
```

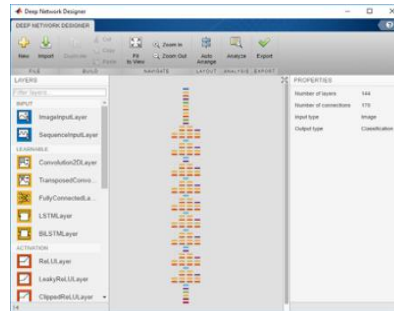


Generates a wrapper  
*detect\_lane\_yolo\_full\_pil*

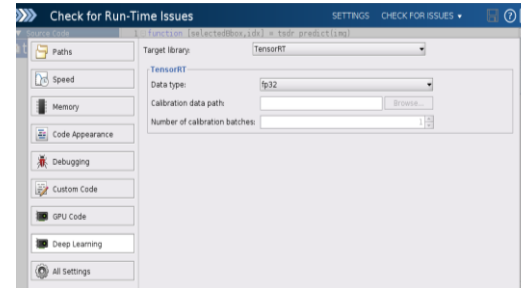
# Outline



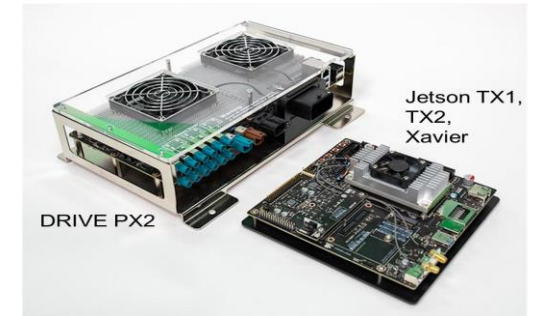
Ground Truth Labeling



Network Design and Training



CUDA and TensorRT Code Generation



Jetson Xavier and DRIVE Xavier Targeting

## Key Takeaways

**Platform Productivity:** Workflow automation, ease of use  
**Framework Interoperability:** ONNX, Keras-TensorFlow, Caffe

## Key Takeaways

**Optimized CUDA and TensorRT code generation**  
**Jetson Xavier and DRIVE Xavier targeting**  
**Processor-in-loop(PIL) testing and system integration**

Thank You