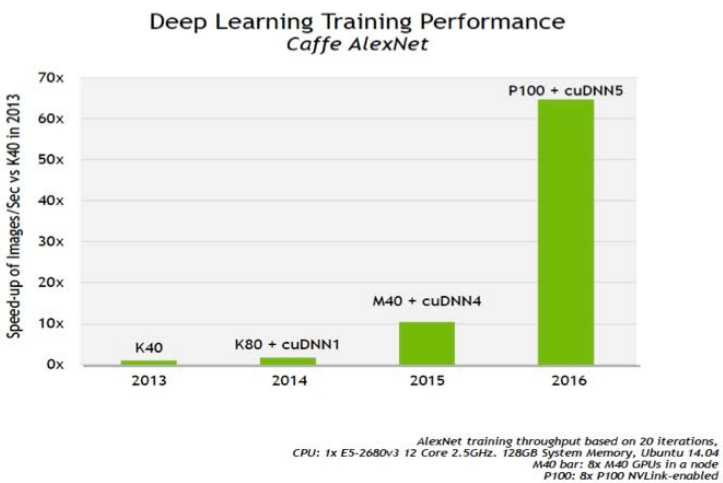
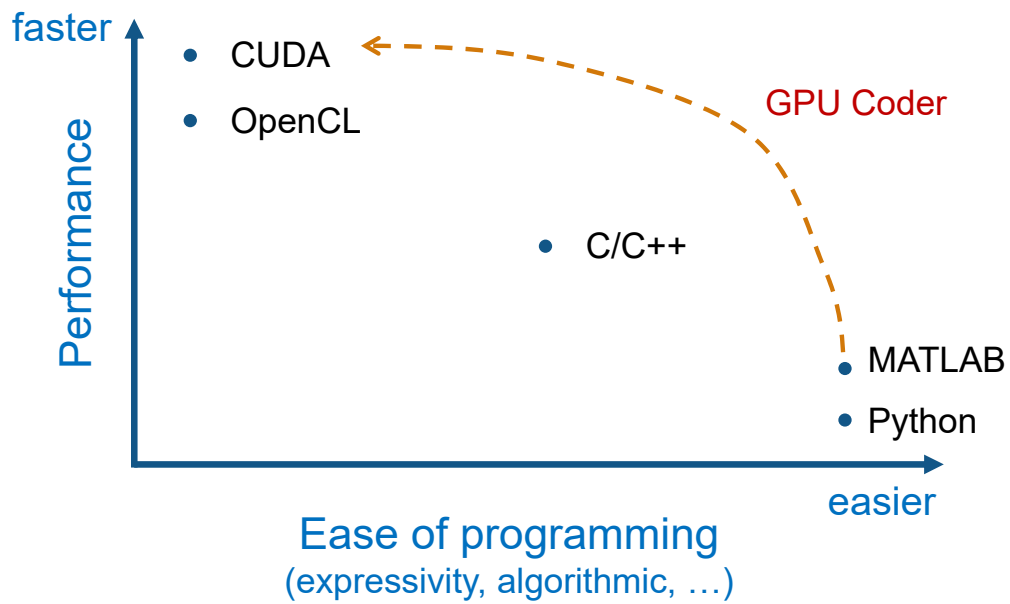
GPU Coder:
Automatic CUDA and TensorRT code generation from MATLAB

Girish Venkataramani
Arvind Jayaraman
Jaya Shankar

GPUs and CUDA programming





GPUs are “hardware on steroids”, … but, programming them is hard

Consider an example: saxpy

Scalarized MATLAB

```
for i = 1:length(x)
    z(i) = a .* x(i) + y(i);
end
```

Vectorized MATLAB

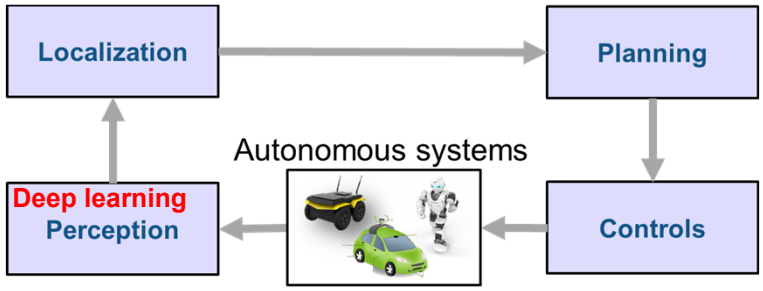
```
z = a .* x + y;
```

GPU Coder

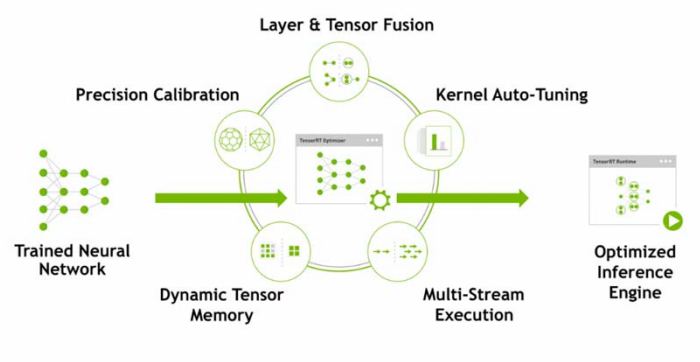
```
static __global__ __launch_bounds__(512, 1) void saxpy_kernel1(const real32_T *y,
  const real32_T *x, real32_T a, real_T *z)
{
  int i = blockIdx.x*blockDim.x + threadIdx.x;
  if (!(i >= 1048576)) {
    z[i] = (real_T)(a * x[i] + y[i]);
  }
}

void saxpy(real32_T a, const real32_T x[1048576], const real32_T y[1048576],
           real_T z[1048576])
{
  real32_T *gpu_y;
  real32_T *gpu_x;
  real_T *gpu_z;
  cudaMalloc(&gpu_z, 8388608UL);
  cudaMalloc(&gpu_x, 4194304UL);
  cudaMalloc(&gpu_y, 4194304UL);
  cudaMemcpy((void *)gpu_y, (void *)&y[0], 4194304UL, cudaMemcpyHostToDevice);
  cudaMemcpy((void *)gpu_x, (void *)&x[0], 4194304UL, cudaMemcpyHostToDevice);
  saxpy_kernel1<<<dim3(2048U, 1U, 1U), dim3(512U, 1U, 1U)>>>(gpu_y, gpu_x, a,
    gpu_z);
  cudaMemcpy((void *)&z[0], (void *)gpu_z, 8388608UL, cudaMemcpyDeviceToHost);
  cudaFree(gpu_y);
  cudaFree(gpu_x);
  cudaFree(gpu_z);
}
```

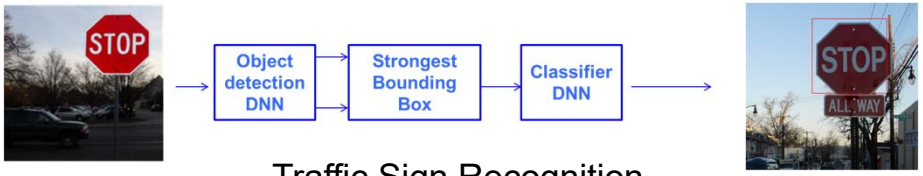
Automatic compilation from an expressive language to a high-performance language

Deep Learning applications



TensorRT





Traffic Sign Recognition

TensorRT is great for inference, ... but, applications require more than inference

GPU Coder is new technology released in September 2017

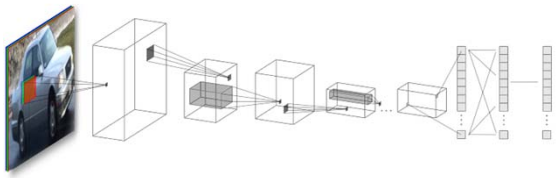
R2017b

MATLAB → GPU Coder → NVIDIA CUDA C/C++

Accelerated implementation of parallel algorithms on GPUs

Neural Networks

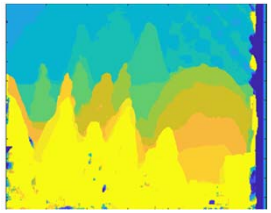
Deep Learning, machine learning



5x faster than TensorFlow
2x faster than mxnet

Image Processing and Computer Vision

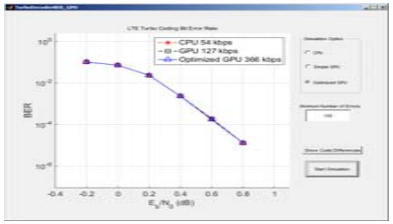
Image filtering, feature detection/extraction



60x faster than CPUs for stereo disparity

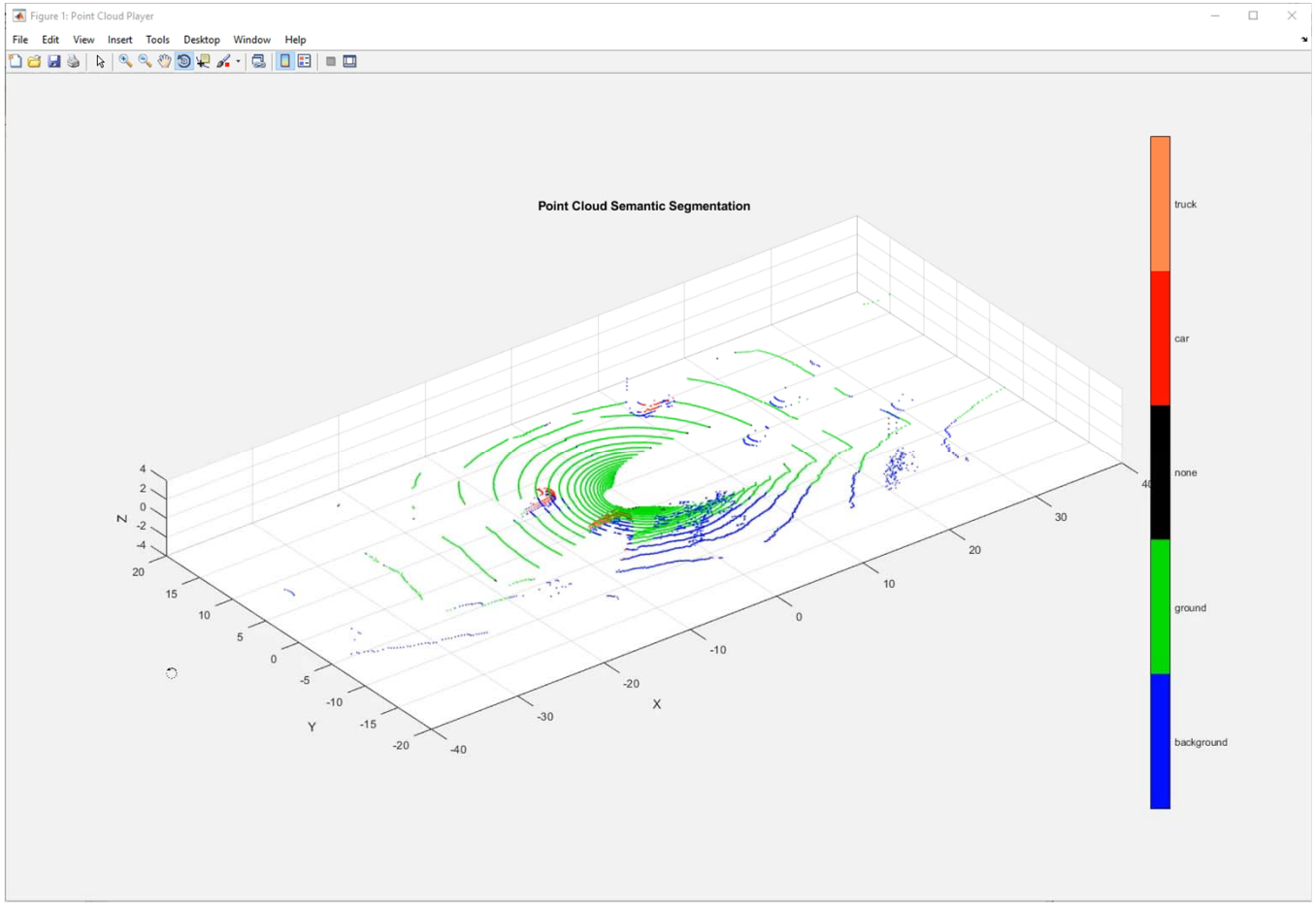
Signal Processing and Communications

FFT, filtering, cross correlation,



20x faster than CPUs for FFTs

Example: Lidar semantic segmentation



Talk outline

- Introduction
- GPU Coder internals
  - Automatic parallelization
  - Memory optimization
  - Deep learning compilation
- Application demo: Lidar processing in MATLAB using deep learning

GPU Coder automatically extracts parallelism from MATLAB

1. Scalarized MATLAB
   ("for-all" loops)

```
%% Pixel processing on the height/width of an image
for i = 1:height
    for j = 1:width
        tmpVal = (width*height);
        for x = 1:width
            disValTmp = temp(x,i);
            dist = ((j-x)^2 + disValTmp^2 );
            if (dist < tmpVal)
                tmpVal = single(dist);
            end
        end
        out(i,j) = tmpVal;
    end
end
```

2. Vectorized MATLAB
   (math operators and library functions)

```
%% Parallel element-wise math to compute
%  Restoration with inverse Koschmieder's law
factor=1.0./(1.0-(diff_im));
restoreOut(:,:,1)= (input(:,:,1)-diff_im).*factor;
restoreOut(:,:,2)= (input(:,:,2)-diff_im).*factor;
restoreOut(:,:,3)= (input(:,:,3)-diff_im).*factor;
```

Infer CUDA kernels from MATLAB loops

3. Composite functions in MATLAB
   (maps to cuBlas, cuFFT, cuSolver, cuDNN, TensorRT)

```
C = A * B; % cuBLAS

y = fft(in); % cuFFT

z = A \ B; % cuSolver
```

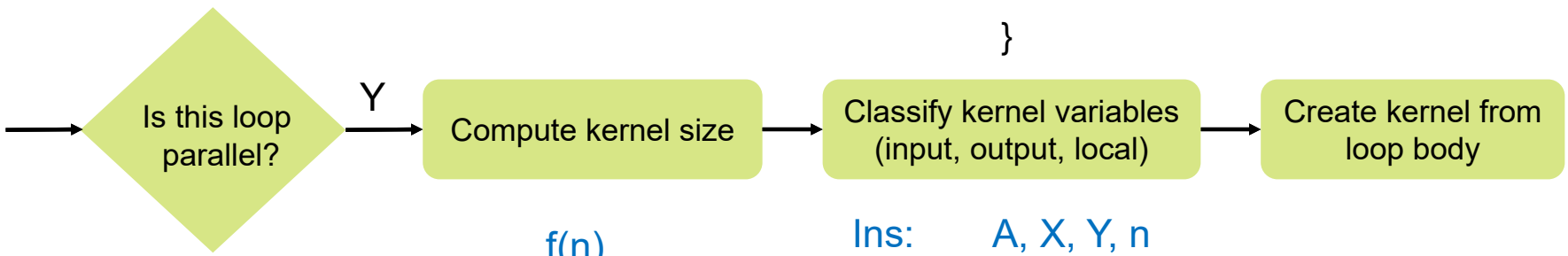
```
predictions = detectionnet.activations(img_rz,56,'OutputAs','channels'); % cuDNN or TensorRT
```

Vendor library replacement

From a loop to a CUDA kernel

```
for k = 1:n
    t = A(k) .* X(k);
    C(k) = t + Y(k);
end
```

```
{ …
    mykernel<<< f(n) >>>(
  …
}

static __global__ mykernel(A, X, Y, C, n)
{



}
```



Dependence analysis to understand the iteration space

f(n)

Ins: A, X, Y, n
Outs: C
Local: t, k

From a loop-nesting to CUDA kernels

Imperfect Loops

```
for i = 1:p
    …(outer prologue code)…
    for j = 1:m
        for k = 1:n
            …(inner loop)…
        end
        …(outer epilogue code)…
    end
end
```

Perfect Loops

```
for i = 1:p
    for j = 1:m
        for k = 1:n
            …(inner loop)…
        end
    end
end
```

Fundamental requirement: Loops need to be contiguous for parallelization

From a loop-nesting to CUDA kernels

Imperfect Loops

```
for i = 1:p
   …(outer prologue code)…
   for j = 1:m
      for k = 1:n
         …(inner loop)…
      end
       …(outer epilogue code)…
    end
end
```

→ **Loop Perfectization (if possible)** →

Perfect Loops

```
for i = 1:p
   for j = 1:m
      for k = 1:n
         … (outer prologue code) …
         …(inner loop)…
         if k == n
            … (outer epilogue code) …
         end
      end
   end
end
```

<u>Fundamental requirement:</u> Loops need to be contiguous for parallelization

From a loop-nesting to CUDA kernels

Example 1

(M x N)

(K x P)

```
for a = 1:M
   for b = 1:N
      …(outer prologue code)…
      for c = 1:K
         for d = 1:P
            …(inner loop)…
         end
      end
   end
end
```

Example 2

(P)

(MxN + KxL)

```
for i = 1:P
   for a = 1:M
      for b = 1:N
         …(inner loop)…
      end
   end
   for x = 1:K
      for y = 1:L
         …(inner loop)…
      end
   end
end
```

Find parallel loops → Partition loop nesting

Dependence analysis

Heuristic may favor larger iteration space

<u>Fundamental requirement:</u> Loops need to be contiguous for parallelization

From a loop-nesting to CUDA kernels

Example 1

(M x N)

(K x P)

```
for a = 1:M
   for b = 1:N
      …(outer prologue code)…
      for c = 1:K
         for d = 1:P
            …(inner loop)…
         end
      end
   end
end
```

Example 2

(P)

(MxN + KxL)

```
for i = 1:P
   for a = 1:M
      for b = 1:N
         …(inner loop)…
      end
   end
   for x = 1:K
      for y = 1:L
         …(inner loop)…
      end
   end
end
```

Find parallel loops → Partition loop nesting → Create kernel for each partition

Dependence analysis

Heuristic may favor larger iteration space

Use process from single loop conversion

<u>Fundamental requirement:</u> Loops need to be contiguous for parallelization

From vectorized MATLAB to CUDA kernels

output(:, 1) = (input(:, 1) – x_im) .* factor;

Assume the following sizes:
'output' : M x 3
'input' : M x 3
'x_im' : M x 1
'factor' : M x 1

```
for i = 1:M
    diff(i) = input(i, 1) – x_im(i);
end
for a = 1:M
    output(i, 1) = diff(i) * factor(i);
end
```

```
for i = 1:M
    diff(i) = input(i, 1) – x_im(i);
    output(i, 1) = diff(i) * factor(i);
end
```

Scalarization → Loop Fusion

Reduce to for-loops

Create larger parallel loops (and hence CUDA kernels)

From vectorized MATLAB to CUDA kernels

Extracting parallelism in MATLAB
1. Scalarized MATLAB (for loops)
2. **Vectorized MATLAB**
3. Composite functions

output(:, 1) = (input(:, 1) – x_im) .* factor;

Assume the following sizes:
'output' : M x 3
'input' : M x 3
'x_im' : M x 1
'factor' : M x 1

```
for i = 1:M
   diff(i) = input(i, 1) – x_im(i);
end
for a = 1:M
   output(i, 1) = diff(i) * factor(i);
end
```

```
for i = 1:M
   diff(i) = input(i, 1) – x_im(i);
   output(i, 1) = diff(i) * factor(i);
end
```

```
for i = 1:M
   tmp = input(i, 1) – x_im(i);
   output(i, 1) = tmp * factor(i);
end
```

Scalarization → Loop Fusion → Scalar Replacement

Reduce to for-loops

Create larger parallel loops (and hence CUDA kernels)

Reduce temp matrices to temp scalars

From composite functions to optimized CUDA

Extracting parallelism in MATLAB
1. Scalarized MATLAB (for loops)
2. Vectorized MATLAB
3. **Composite functions**

Core math

- Matrix multiply (cuBLAS)
- Linear algebra (cuSolver)
- FFT functions (cuFFT)
- Convolution
- …

Image processing Computer vision

- imfilter
- imresize
- imerode
- imdilate
- bwlabel
- imwarp
- …

Neural Networks

- Deep learning inference (cuDNN, TensorRT)
- …

Over 300+ MATLAB functions are optimized for CUDA code generation

Talk outline

- Introduction
- GPU Coder internals
  - Automatic parallelization
  - Memory optimization
  - Deep learning compilation
- Application demo: Lidar processing in MATLAB using deep learning

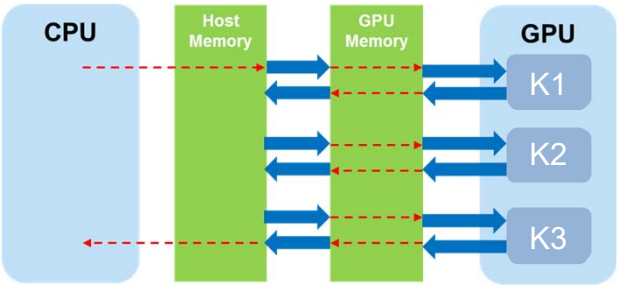
Optimizing CPU-GPU data movement is a challenge

```
A = …
…
for i = 1:N
   … A(i)
end
…

imfilter
…
```

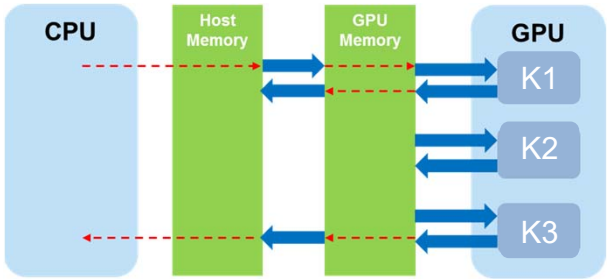
```
A = …
…
cudaMemcpyHtoD(gA, a);
kernel1<<<…>>>(gA)
cudaMemcpyDtoH(…)
…
cudaMemcpyHtoD(…)
imfilter_kernel(…)
cudaMemcpyDtoH(…)
…
```

Where is the ideal placement of memcpy?

Naïve placement



Optimized placement



GPU Coder optimizes memcpy placement

```
A(:) = ….
C(:) = ….

for i = 1:N
    ….
    gB = kernel1(gA);
    gA = kernel2(gB);
    if (some_condition)
        gC = kernel3(gA, gB);
    end
    ….
end

…. = C;
```

cudaMemcpy
definitely needed

cudaMemcpy
not needed

cudaMemcpy
may be needed

Assume gA, gB and gC are mapped to GPU memory

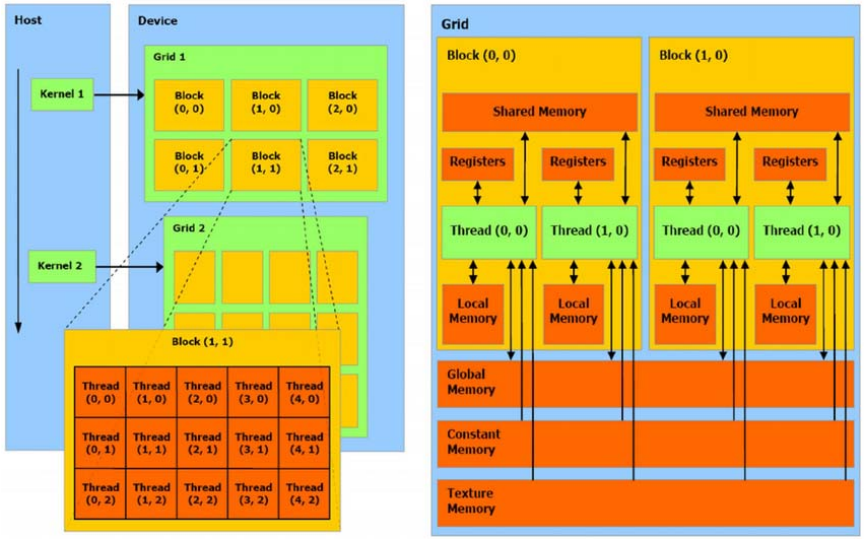
<u>Observations</u>

- Equivalent to Partial redundancy elimination (PRE)
- Dynamic strategy – track memory location with a status flag per variable
- Use-Def to determine where to insert memcpy

Generated (pseudo) code

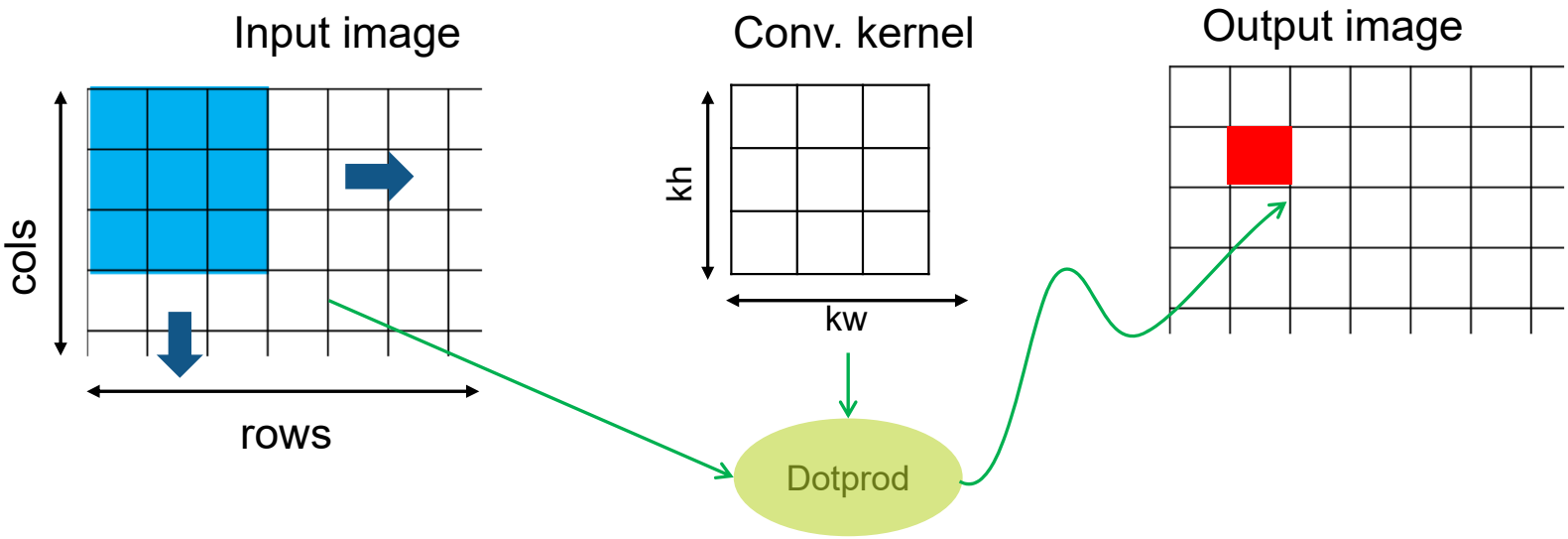
```
A(:) = …
A_isDirtyOnCpu = true;
…
for i = 1:N
   if (A_isDirtyOnCpu)
      cudaMemcpy(gA, A);
      A_isDirtyOnCpu = false;
   end
   gB = kernel1(gA);
   gA = kernel2(gB);
   if (somecondition)
      gC = kernel3(gA, gB);
      C_isDirtyOnGpu = true;
   end
   …
end
…
if (C_isDirtyOnGpu)
   cudaMemcpy(C, gC);
   C_isDirtyOnGpu = false;
end
… = C;
```

GPU memory hierarchy is deep



Memory	Visibility	Heuristics/notes	GPU Coder support
Global memory	CPU + GPU	Share data b/w CPU and GPU	Yes
Local memory/registers	per GPU thread	Thread local data	Yes
Shared memory	per GPU block	Shared data between threads	Yes
Texture memory	CPU + GPU	Shared read-only data with 2D alignment	Future
Constant memory	GPU	Read-only constants	Yes

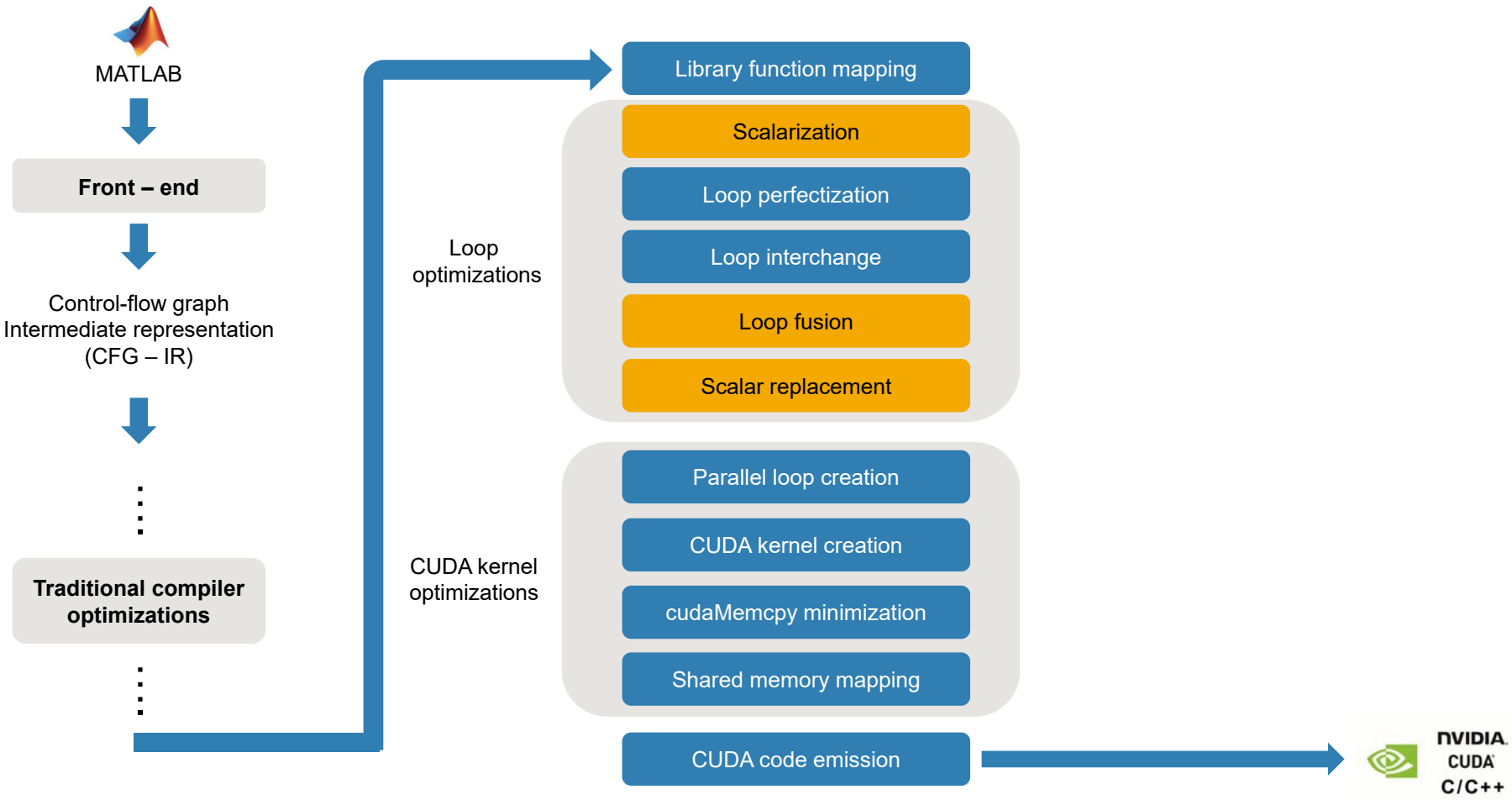
GPU Coder automatically maps data to shared memory



GPU Coder automatically creates shared memory for many MATLAB image processing functions:

`imfilter, imerode, imdilate, conv2, ...`

GPU Coder runs a host of compiler transforms to generate CUDA

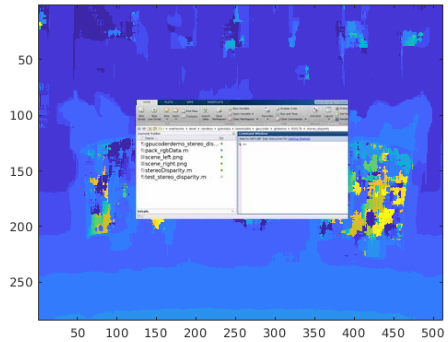


Demo: Stereo disparity

Left camera

Right camera

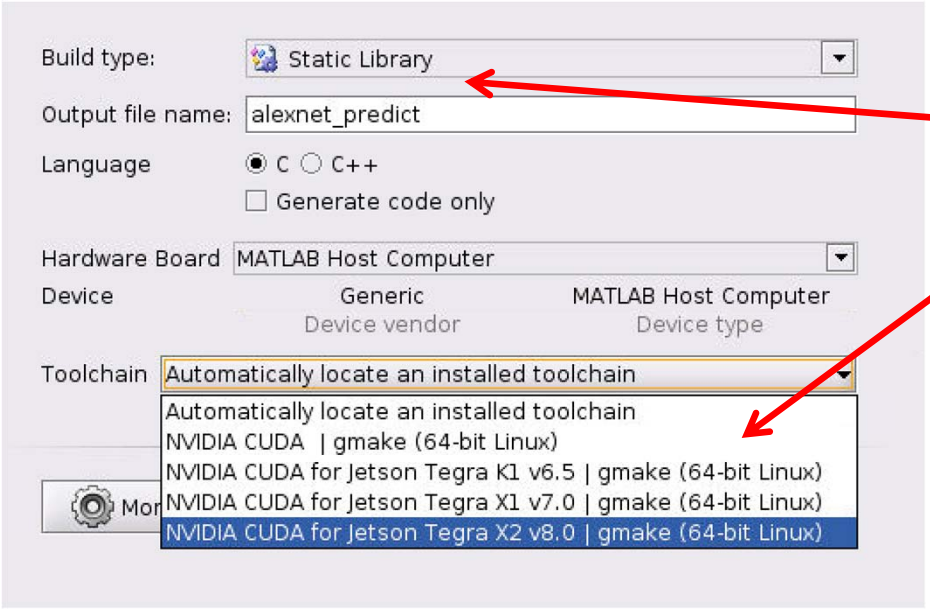
Stereo disparity



Disparity map

Easily re-target to Jetson and Drive platforms

- Cross-compile for NVIDIA boards
  - Jetson boards
  - DrivePX2



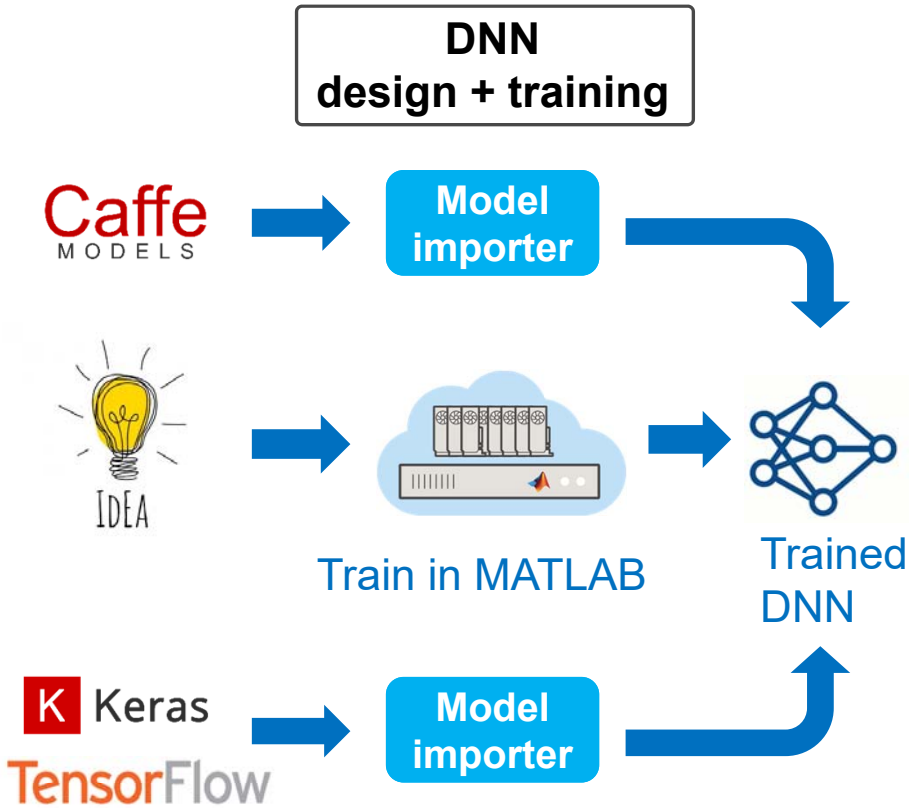
Two small changes

1. Change build-type to 'lib'
2. Select cross-compile toolchain

Talk outline

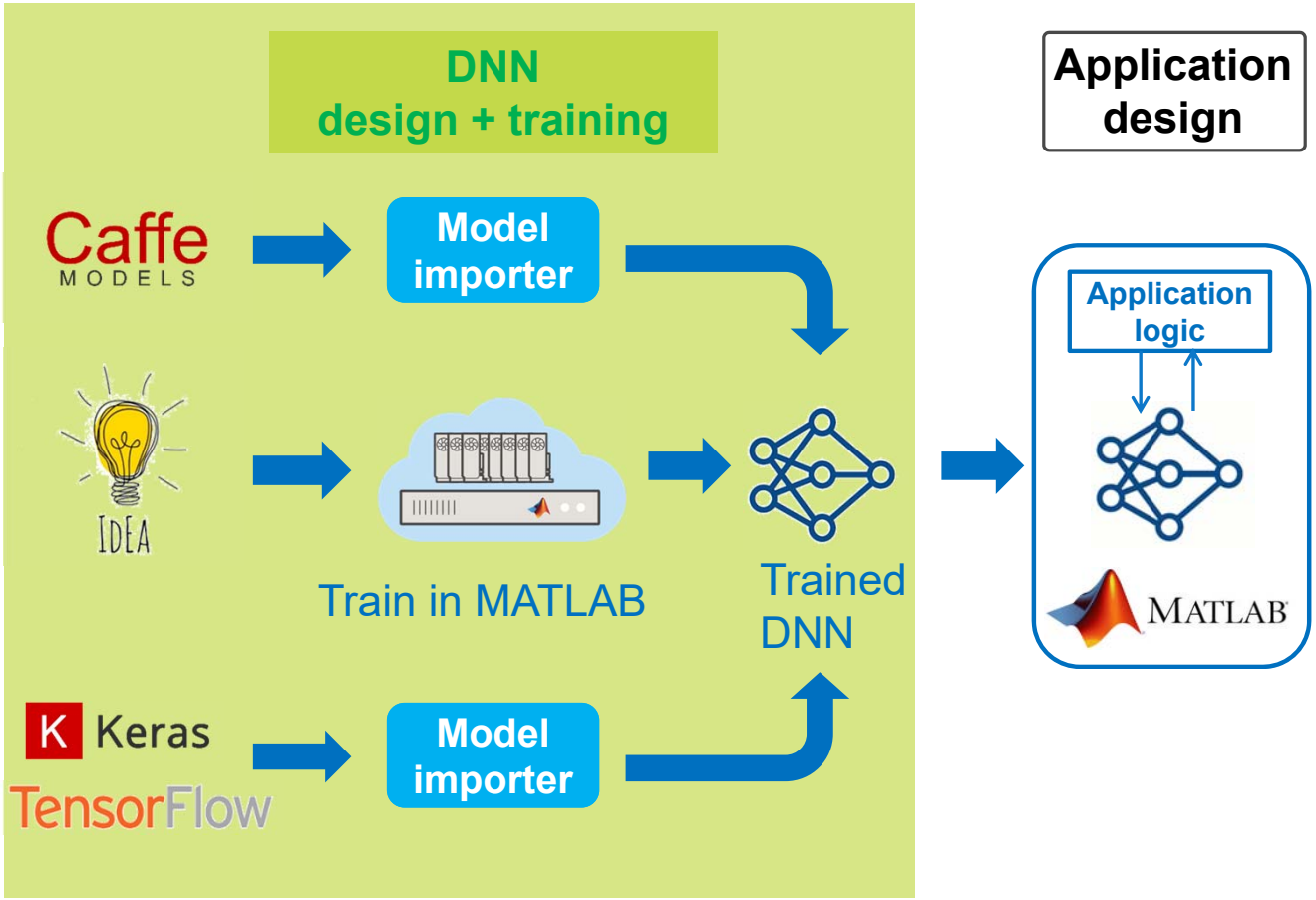
- Introduction
- GPU Coder internals
  - Automatic parallelization
  - Memory optimization
  - Deep learning compilation
- Application demo: Lidar processing in MATLAB using deep learning

Deep learning workflow in MATLAB

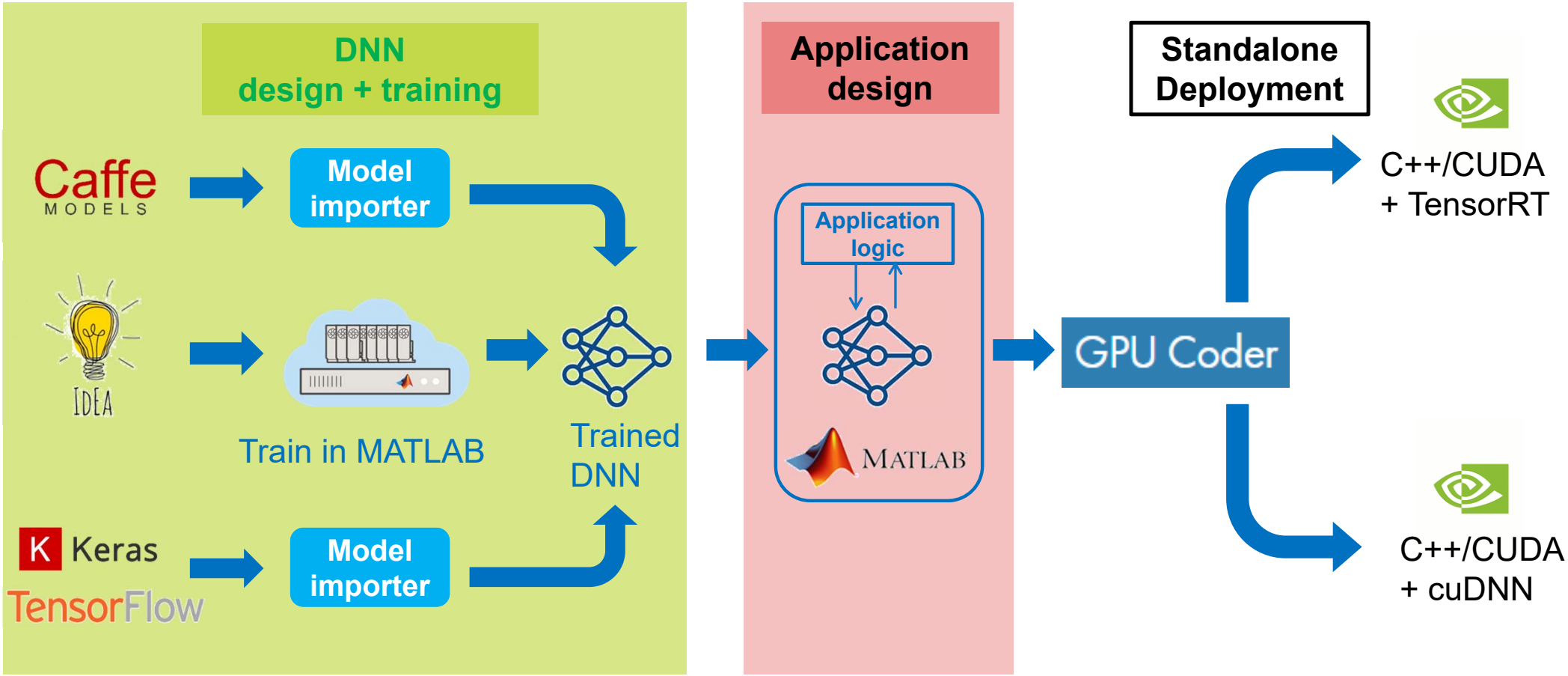


- **Design in MATLAB**
  - **Manage** large image sets
  - **Automate** data labeling
  - **Easy access** to models
- **Training in MATLAB**
  - **Acceleration** with GPU's
  - **Scale** to clusters

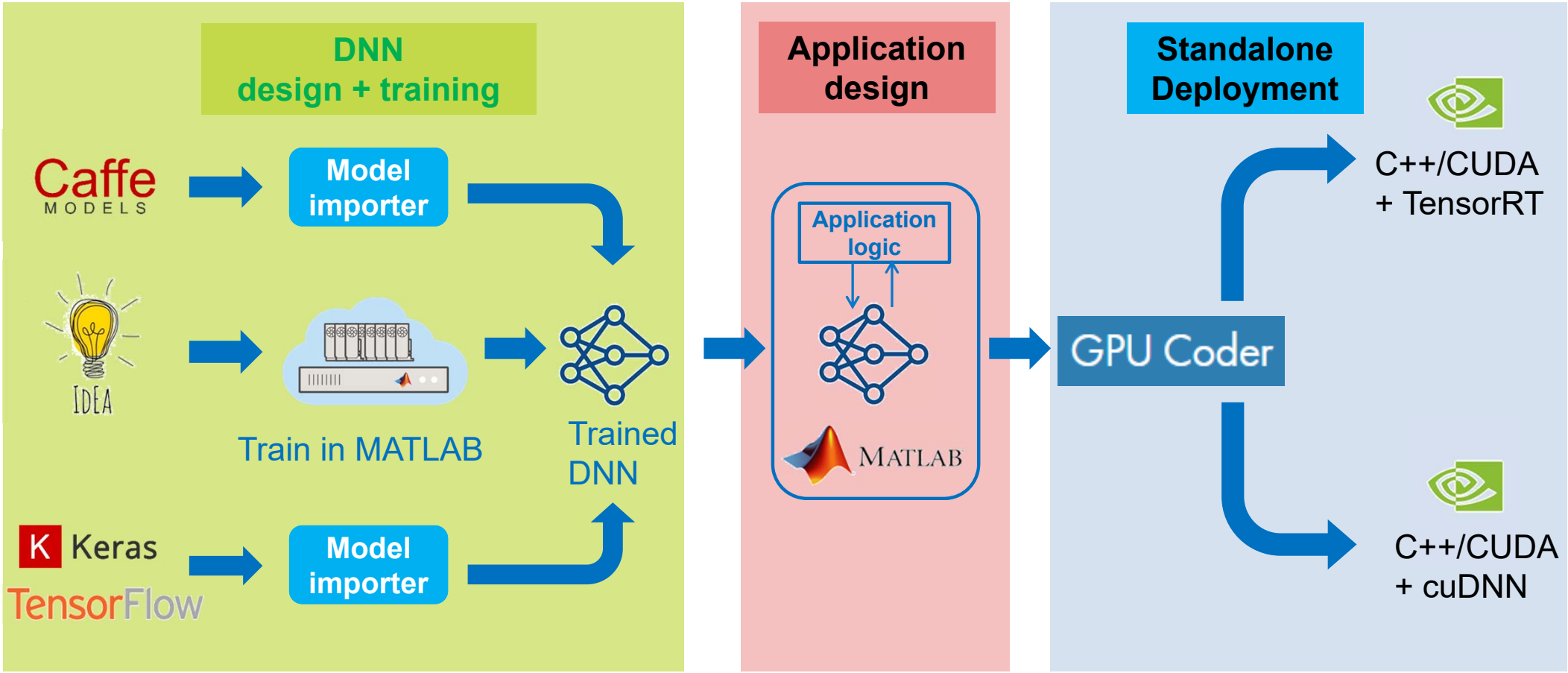
Deep learning workflow in MATLAB



Deep learning workflow in MATLAB



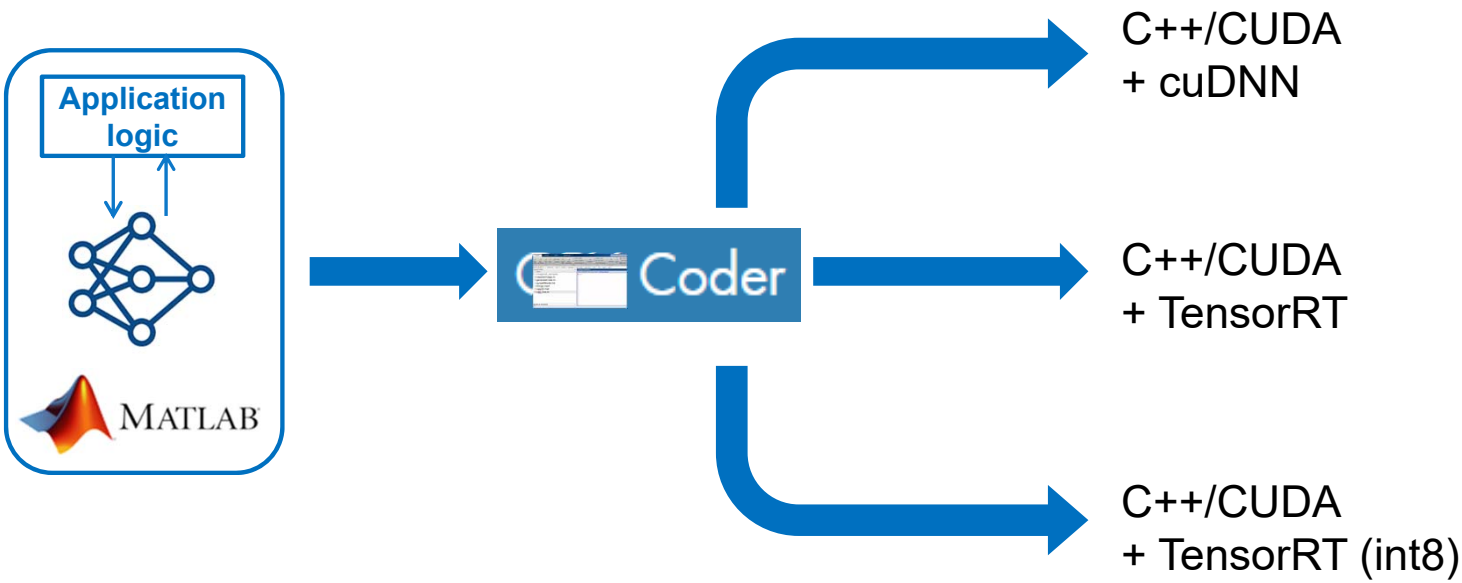
Deep learning workflow in MATLAB



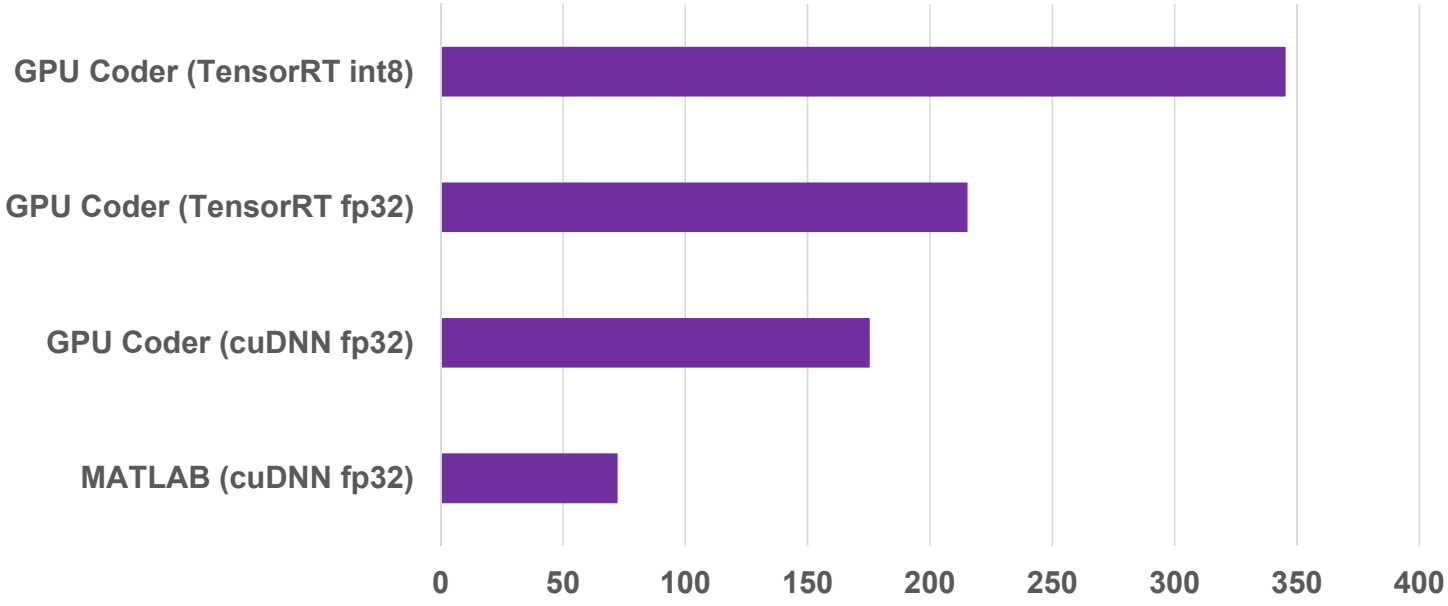
Traffic sign detection and recognition

Object detection DNN → Strongest Bounding Box → Classifier DNN →

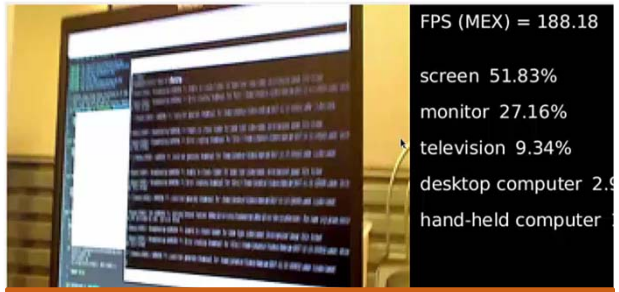
GPU Coder allows for choice in deployment (cuDNN, TensorRT)



Performance summary (VGG-16) on TitanXP

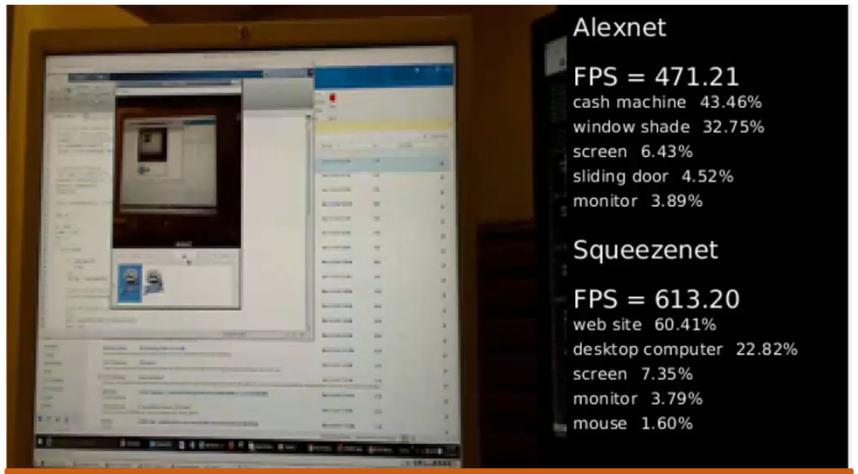


MATLAB and GPU Coder support state-of-art classification networks



GoogLeNet

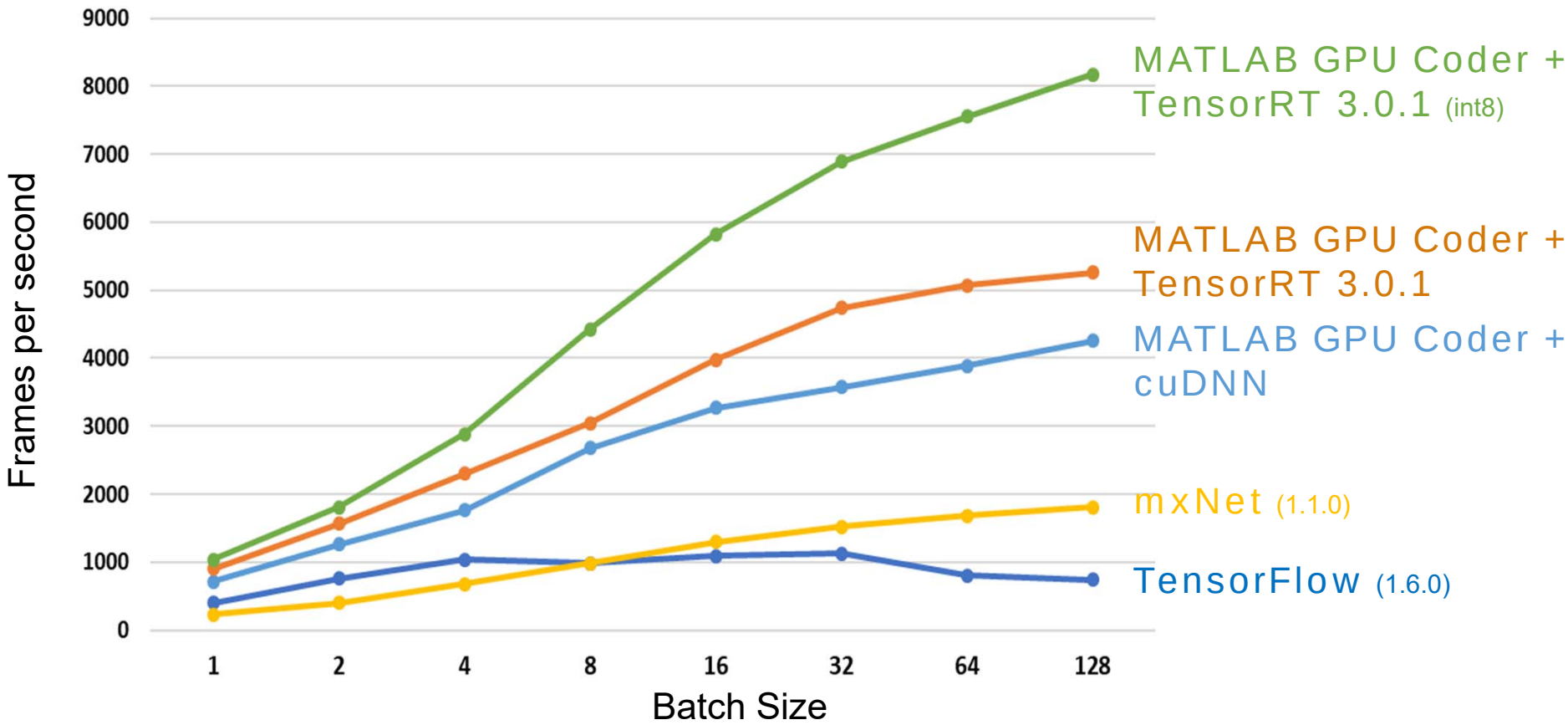
ResNet50



Alexnet vs Squeezenet

Network	# Layers	Size	Frame-rate (GPU Coder)
Alexnet	25	227 MB	500 Fps
ResNet50	177	96 MB	160 Fps
GoogLeNet	144	27 MB	190 Fps
Squeezenet	68	5 MB	615 Fps

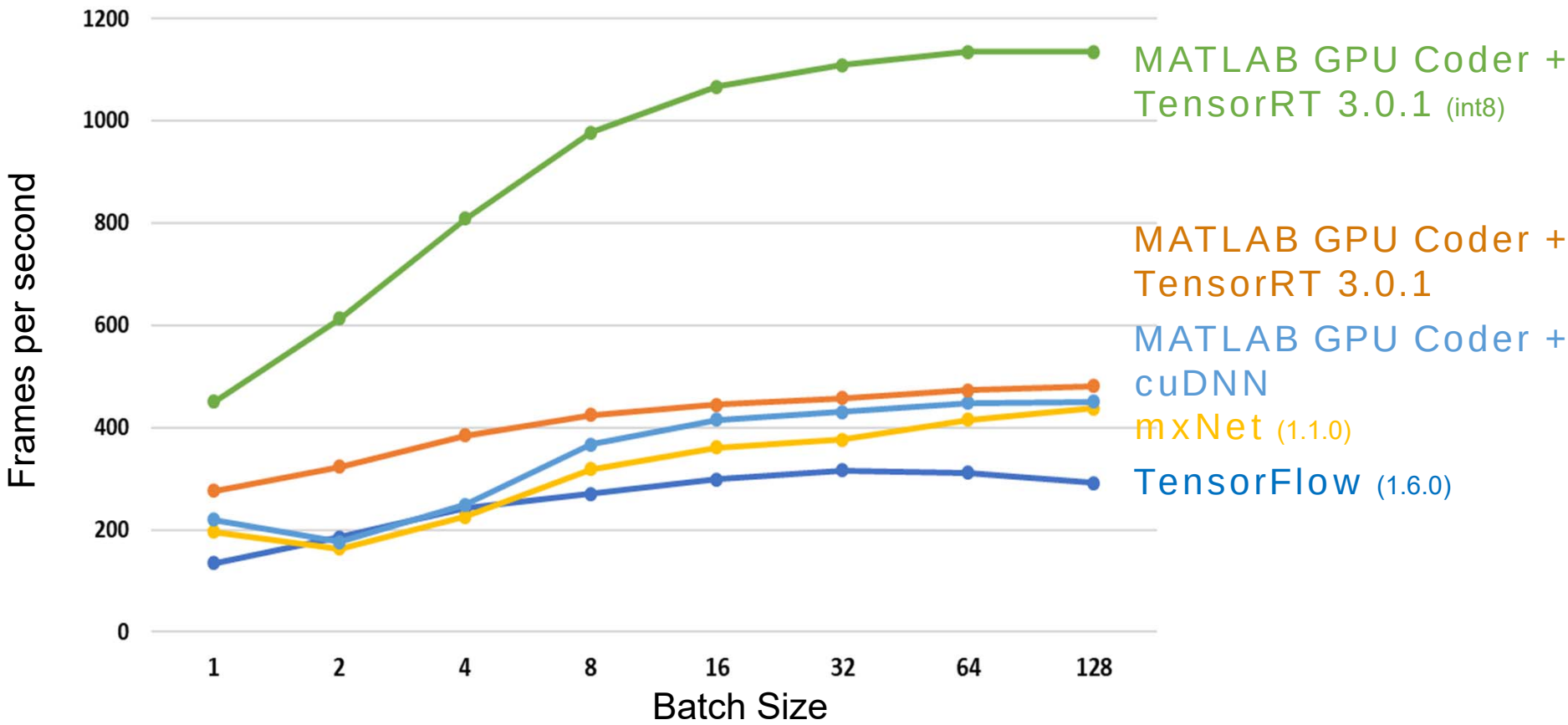
Alexnet Inference on NVIDIA **Titan Xp**



Testing platform

CPU	Intel(R) Xeon(R) CPU E5-1650 v3 @ 3.50GHz
GPU	Pascal Titan Xp
cuDNN	v7

VGG-16 Inference on NVIDIA **Titan Xp**

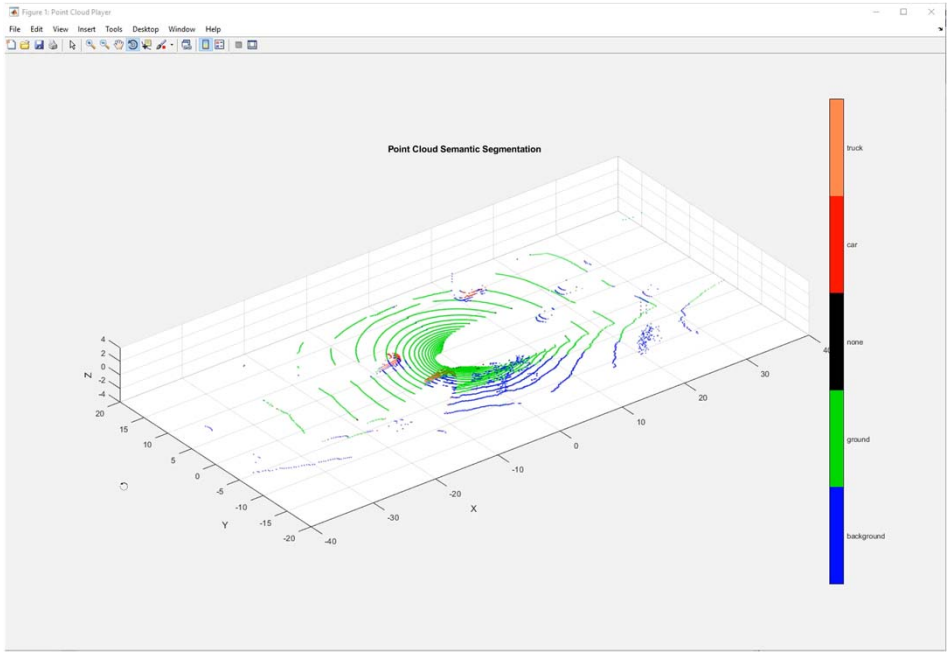


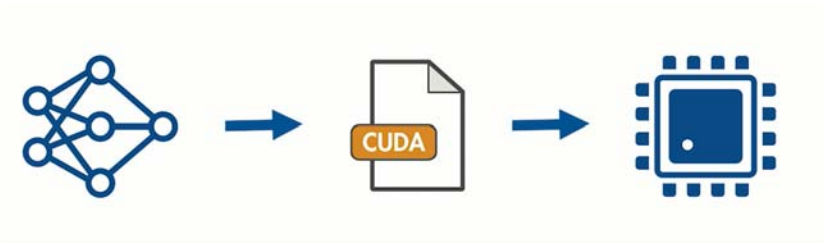
Testing platform	CPU	Intel(R) Xeon(R) CPU E5-1650 v3 @ 3.50GHz
	GPU	Pascal Titan Xp
	cuDNN	v7

Talk outline

- Introduction
- GPU Coder internals
  - Automatic parallelization
  - Memory optimization
  - Deep learning compilation
- Application demo: Lidar processing in MATLAB using deep learning

LiDAR Processing for Autonomous Vehicles





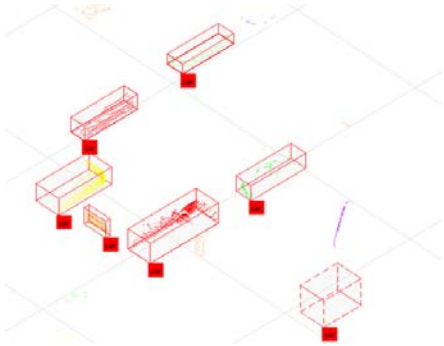
Design Deep learning-based LiDAR algorithm in MATLAB

- **Automate** ground-truth labeling
- **Pre-process** LiDAR data for training
- **GPU accelerated** training

High Performance Inference

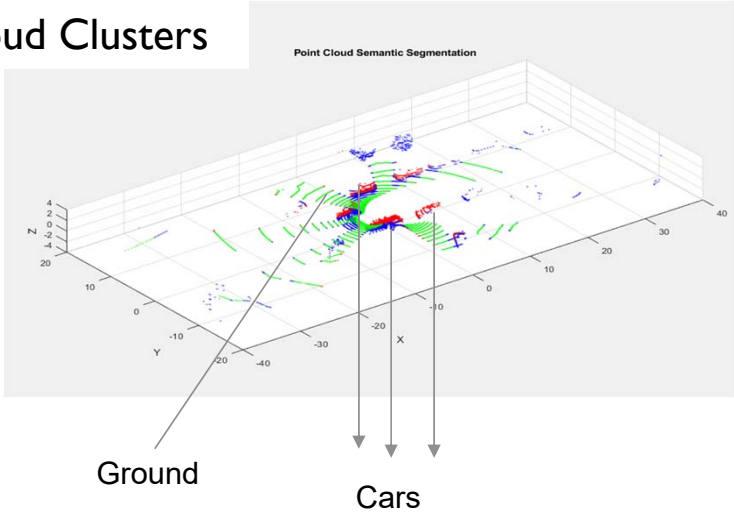
Why Use LiDAR and Deep Learning for Autonomous Vehicles ?

- Why use LiDAR ?
  - Provides accurate 3-D structure of scene
  - Required sensor for autonomous driving



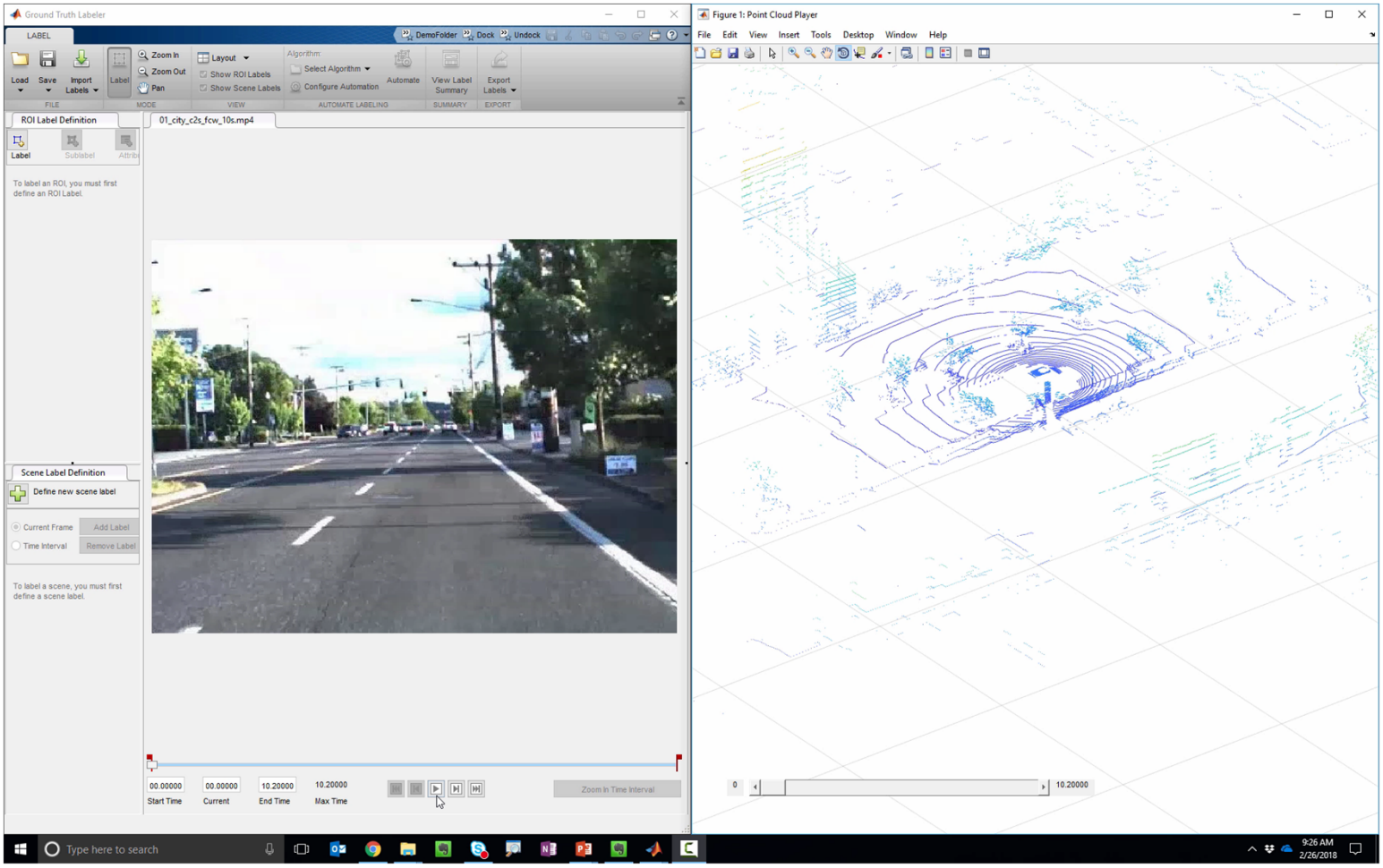
Classifying Point Cloud Clusters

- Why use deep learning ?
  - Best accuracy
  - High-performance inference with GPU Coder

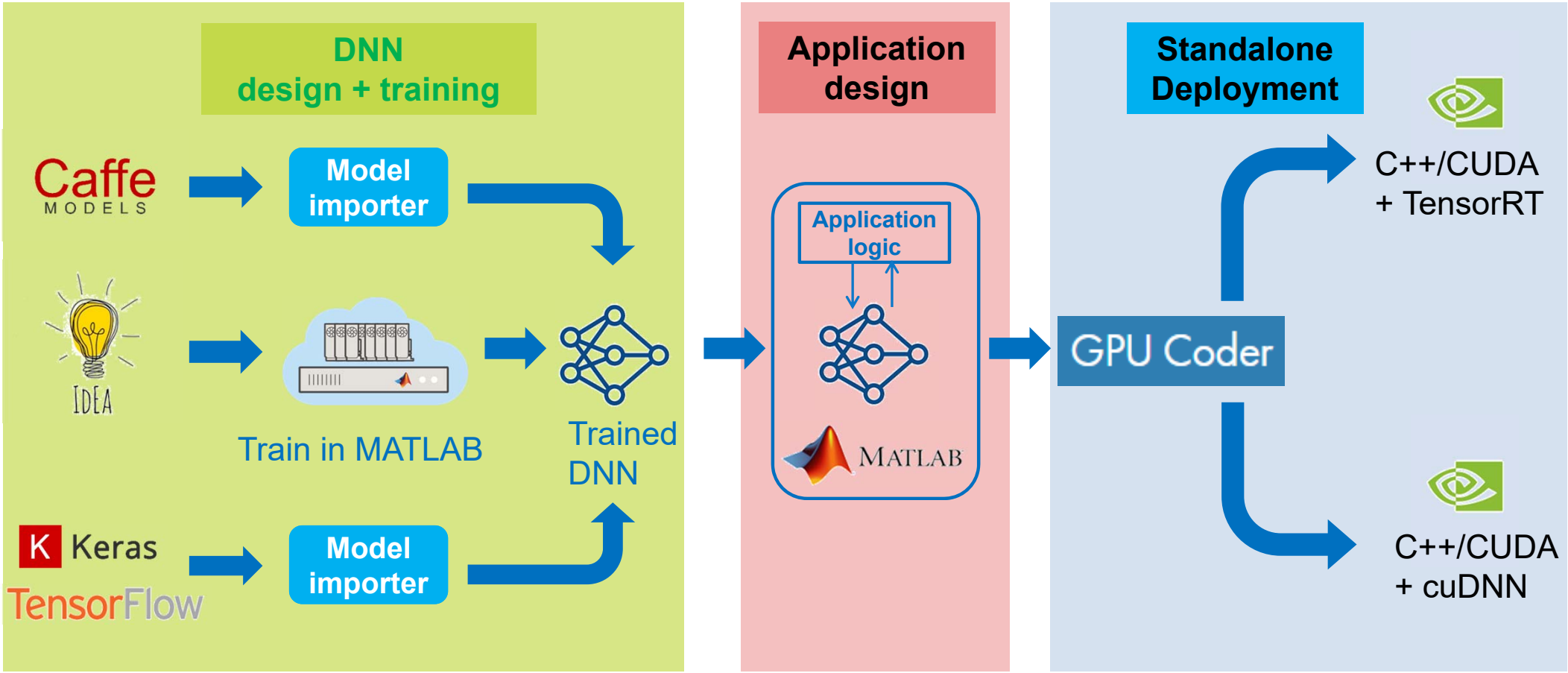


LiDAR Sematic Segmentation

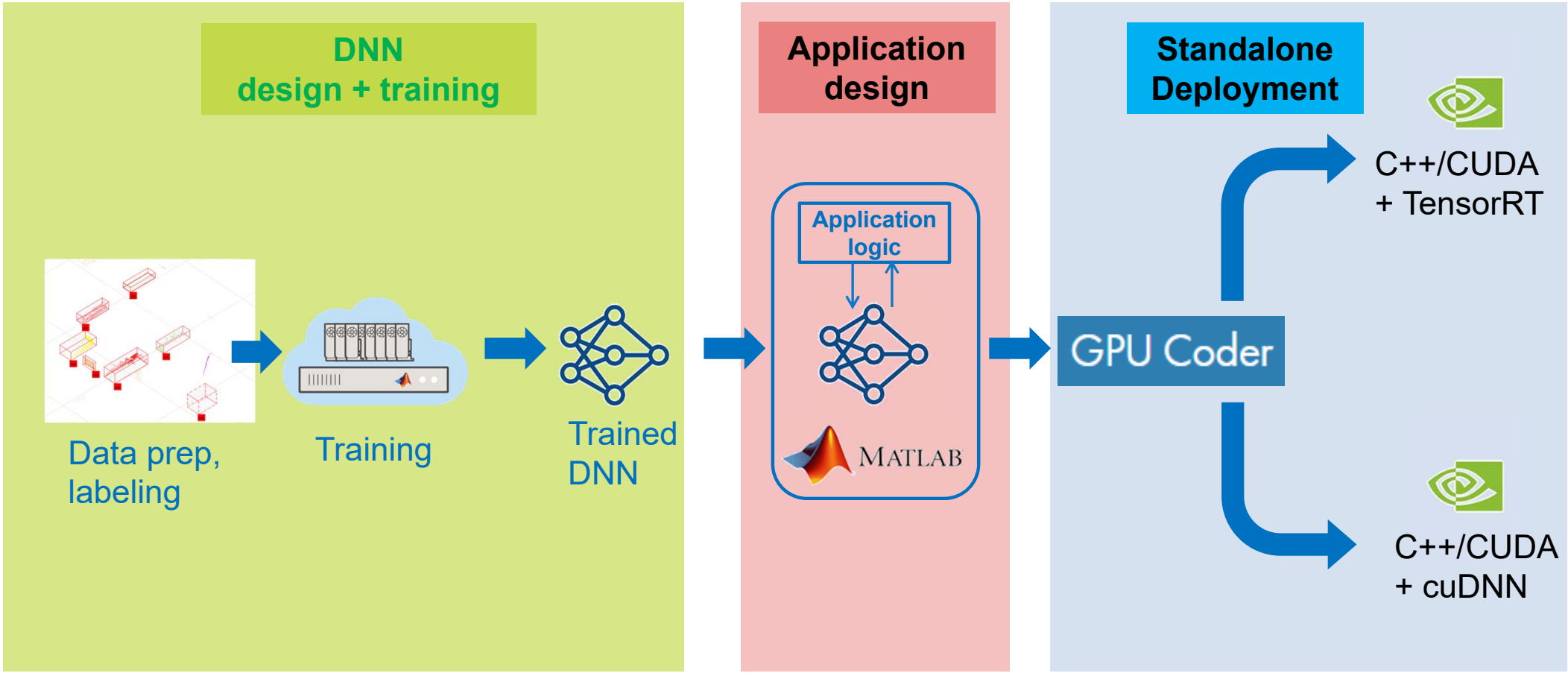
What Does LiDAR Data Look Like ?



Lidar processing application design is easy in MATLAB

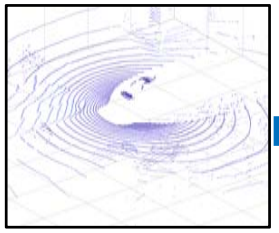


Lidar processing application design is easy in MATLAB

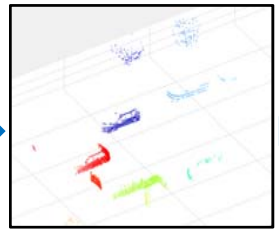


Data preparation and labeling of Lidar is a challenge

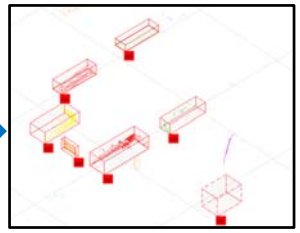
DNN
design + training



Accessing lidar data



Lidar pre-processing

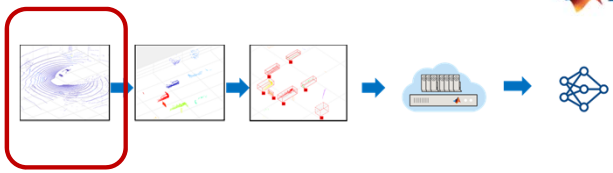


Labeling lidar data

Training

Trained DNN

Access and Visualize LiDAR Data

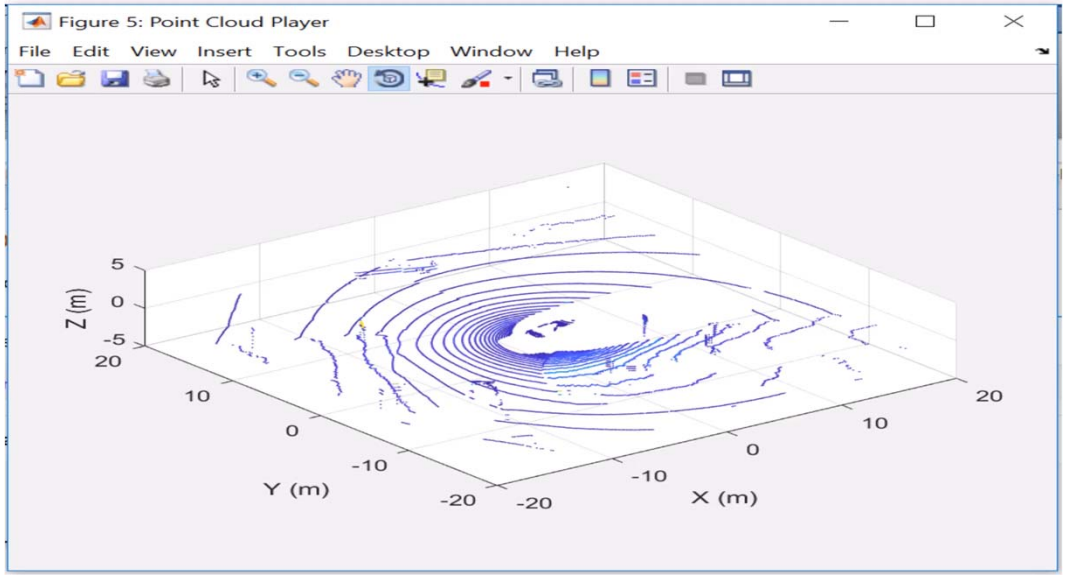


Access Stored LiDAR Data

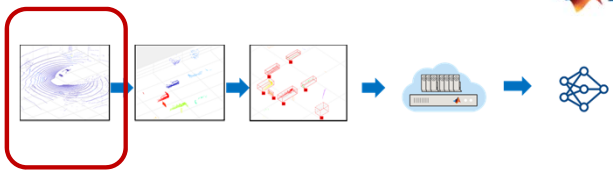
- Velodyne file I/O (pcap)
- Individual point clouds (.pcd,ply)

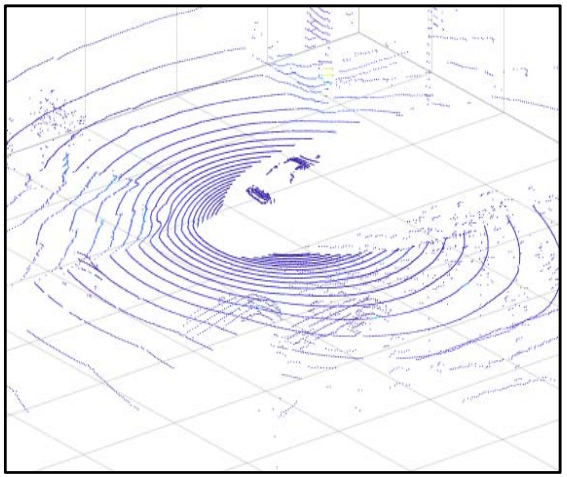
Visualize LiDAR Data

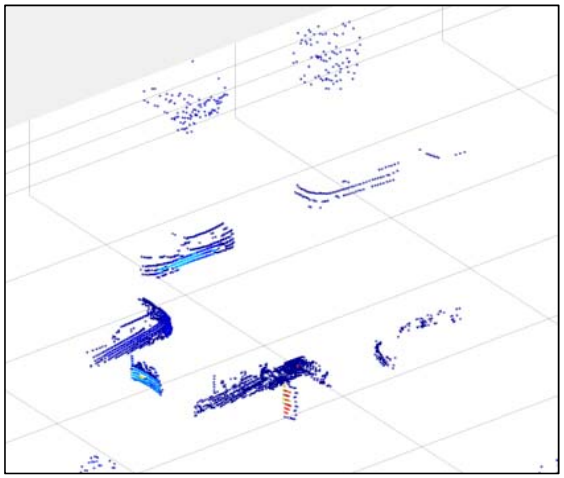
- Streaming LiDAR player
- Static point cloud display
- Point cloud differences



LiDAR Preprocessing

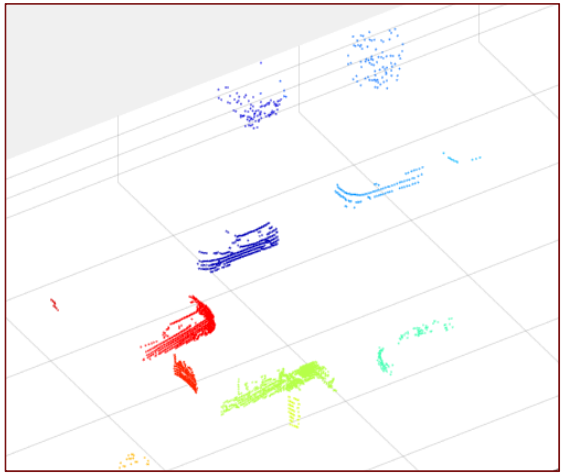






ROI + Remove Ground

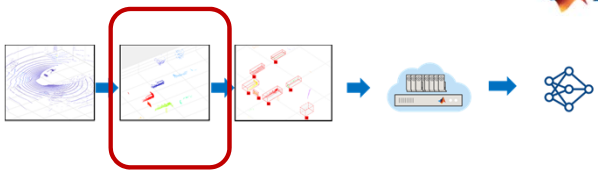
- Fit plane using RANSAC

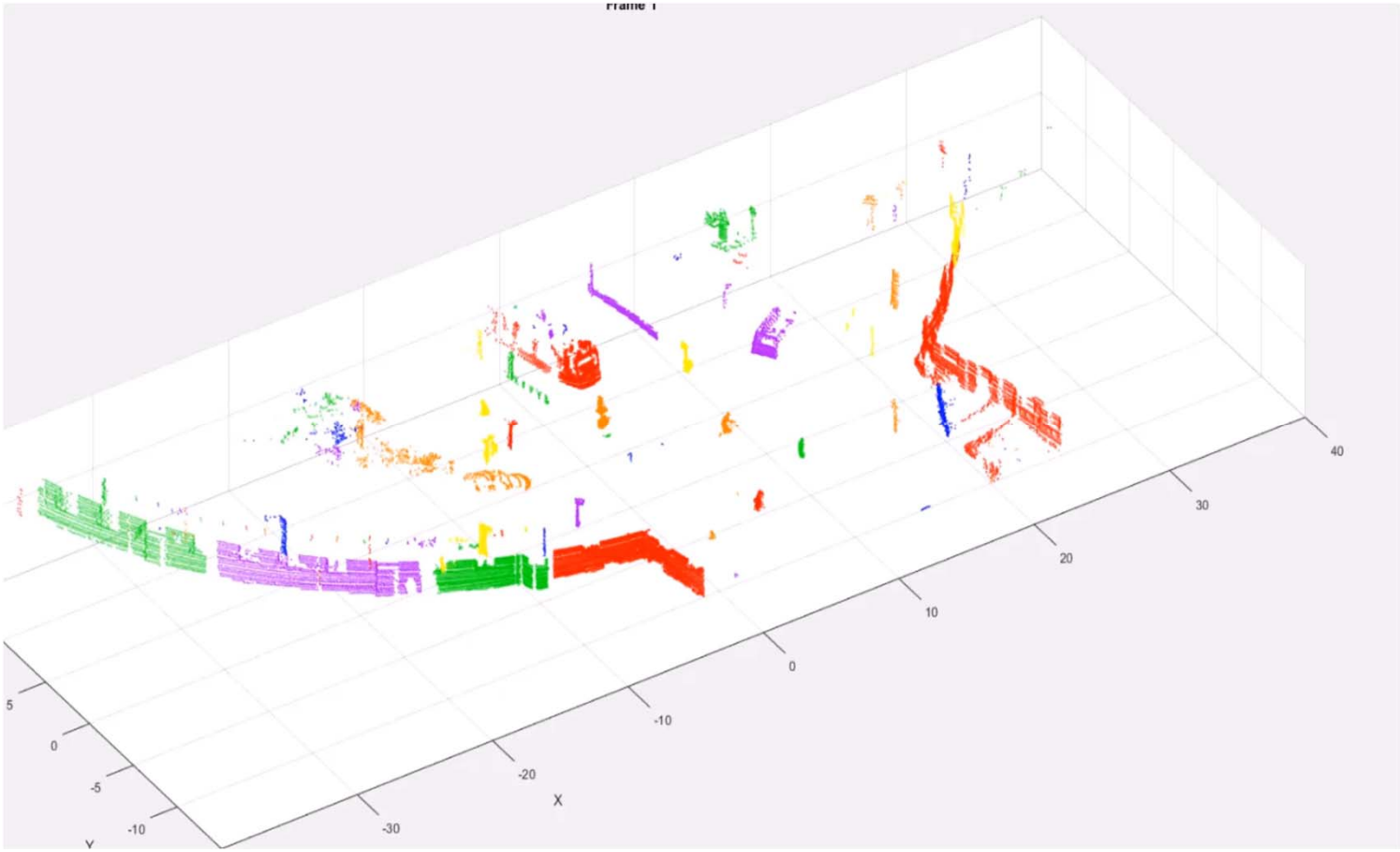


Cluster

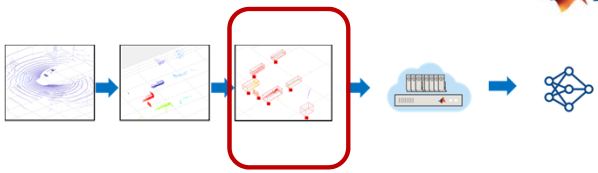
- Segment clusters using Euclidean distance

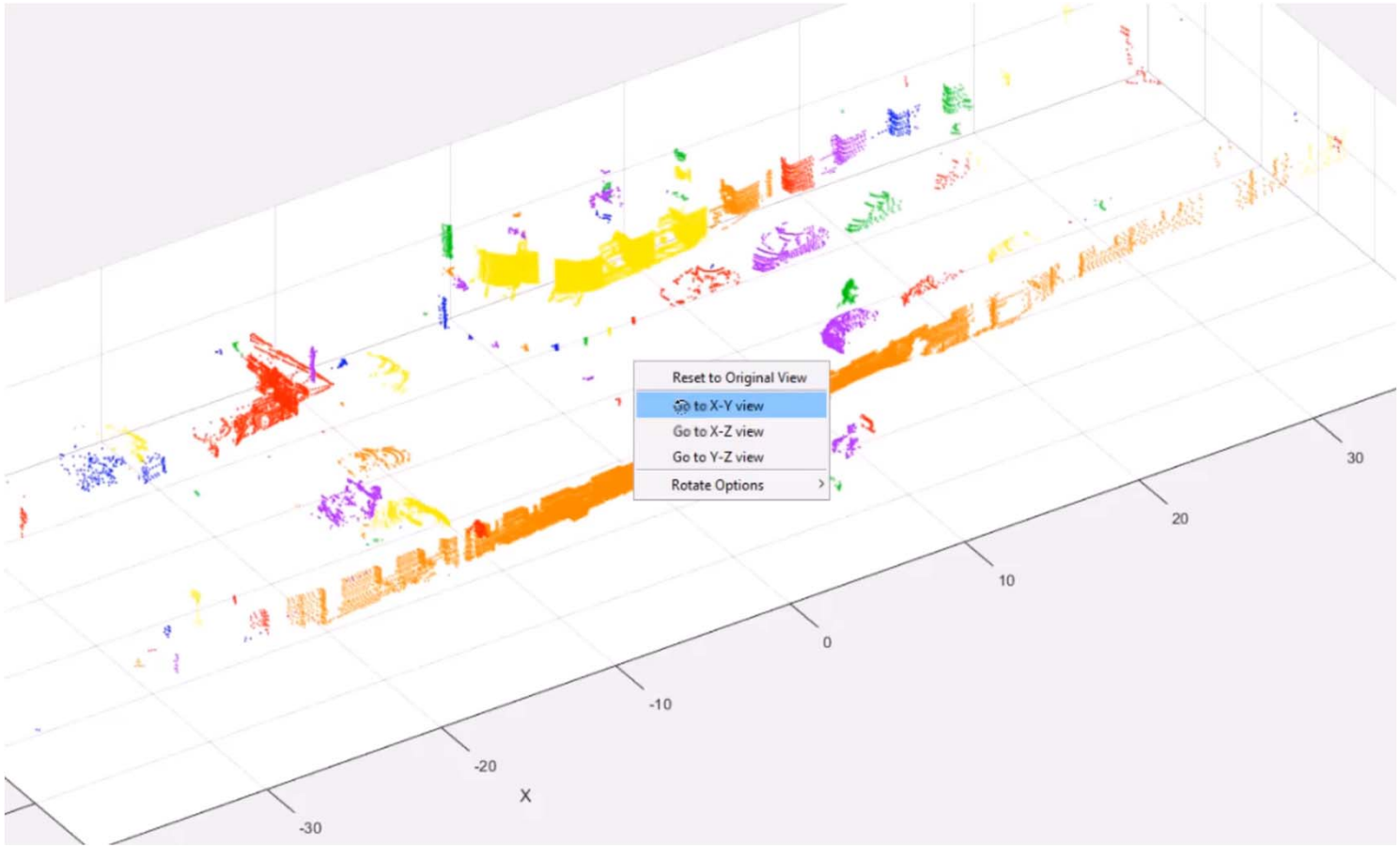
Ground Truth Labeling of LiDAR Data



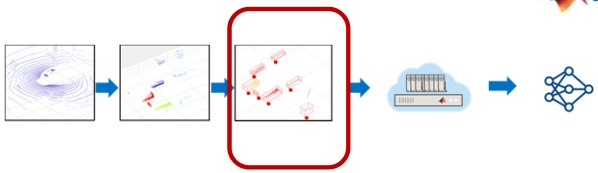


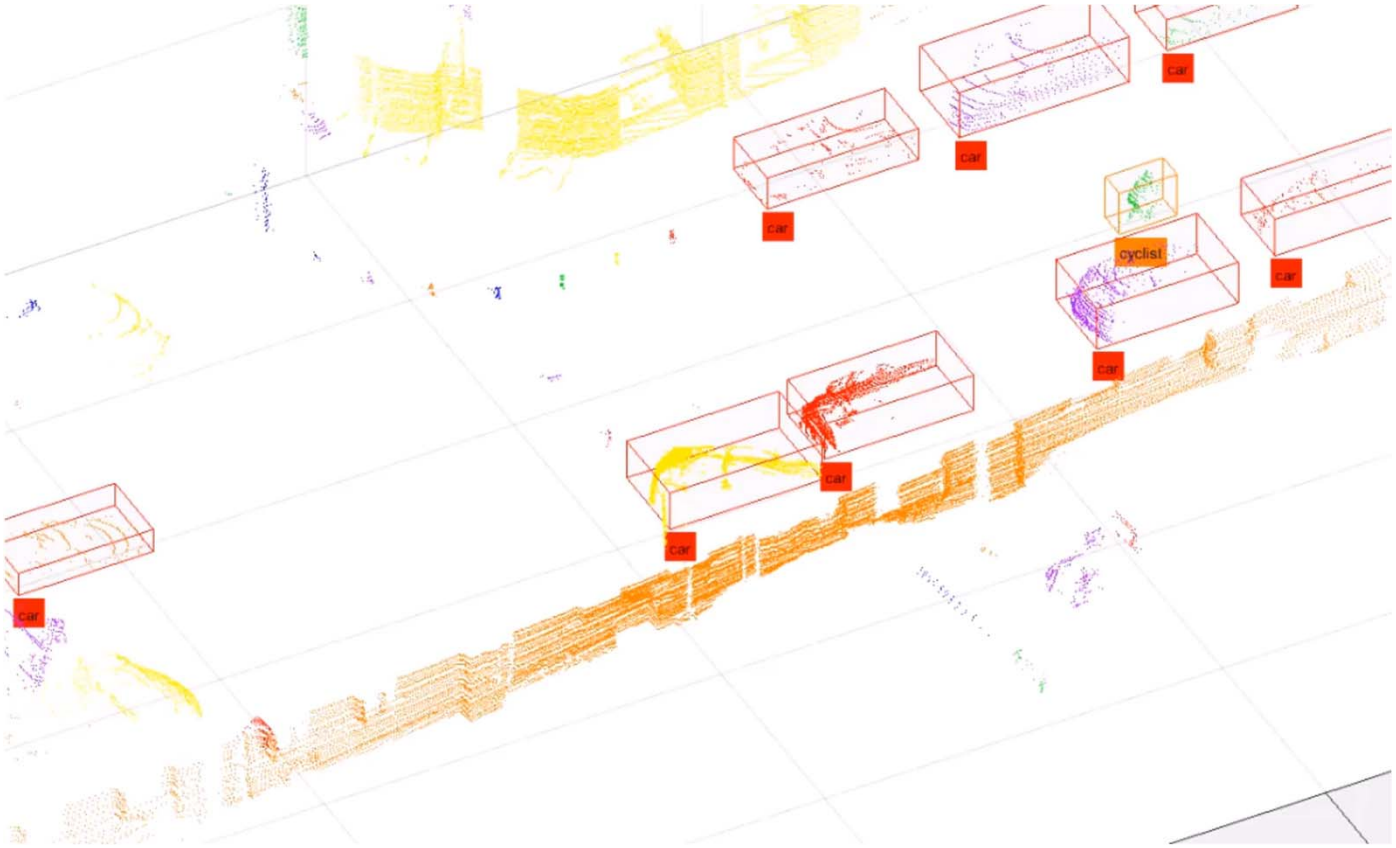
Ground Truth Labeling of LiDAR Data



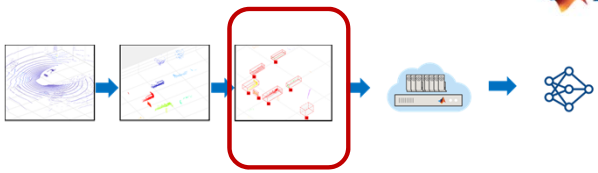


Ground Truth Labeling of LiDAR Data





Organize Data for Training



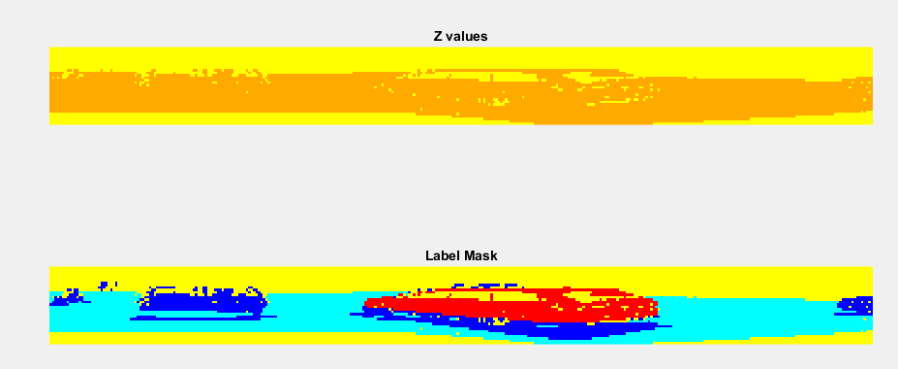
Raw Point Cloud Data

32x1085 double

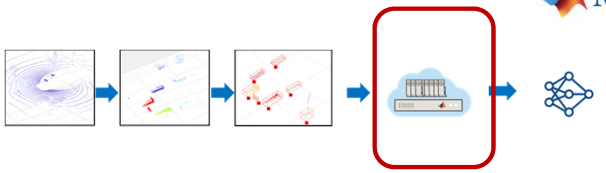
	1	2	3	4	5	6	7
1	NaN	0.1761	0.2727	0.3605	0.4606	0.5649	NaN
2	0.0770	0.1932	0.3064	0.4593	0.5258	0.5784	0.6767
3	NaN	NaN	0.2822	0.4204	0.5332	0.5561	0.8364
4	0.0834	0.2183	0.2875	0.3901	0.5028	0.6081	NaN
5	NaN	NaN	NaN	NaN	NaN	0.7261	0.8329
6	0.0791	0.2409	0.4033	0.5664	0.7491	0.9279	1.1082
7	0.0745	0.2363	0.3987	0.5615	0.7447	0.9238	1.1042
8	0.0651	0.2266	0.3885	0.5512	0.7337	0.9124	1.0923
9	0.0559	0.2177	0.3799	0.5424	0.7255	0.9042	1.0841
10	0.0512	0.2130	0.3749	0.5376	0.7205	0.8993	1.0798
11	0.0419	0.2037	0.3657	0.5283	0.7115	0.8901	1.0706
12	0.0311	0.1824	0.3295	0.4724	0.6289	0.7765	0.9202
13	0.0074	0.0489	0.0886	0.1270	0.1690	0.2084	0.2455
14	0.0046	0.0436	0.0814	0.1174	NaN	0.1951	0.2314
15	0.0023	0.0407	0.0778	0.1138	0.1529	0.1902	0.2262
16	0.0011	0.0394	0.0765	0.1188	0.1591	0.1928	0.2278
17	0.0281	0.0663	0.1033	NaN	0.1793	NaN	0.2528
18	0.0259	NaN	NaN	0.1376	0.1784	NaN	0.2517
19	0.0237	0.0621	0.0997	0.1537	0.1776	NaN	0.2515
20	0.0234	0.0631	0.0982	0.1340	0.1734	0.2102	0.2467
21	0.0192	0.0555	0.0910	0.1251	0.1627	0.1979	0.2322
22	0.0168	0.0493	0.0829	0.1145	0.1491	0.1816	0.2132
23	0.0137	0.0449	0.0752	0.1046	0.1366	0.1669	0.1962
24	0.0111	0.0401	0.0681	0.0954	0.1252	0.1532	0.1805
25	0.0095	0.0366	0.0627	0.0883	0.1162	0.1420	0.1675
26	0.0074	0.0327	0.0571	0.0810	0.1071	0.1314	0.1550
27	NaN	NaN	NaN	NaN	NaN	NaN	NaN
28	NaN	NaN	NaN	NaN	NaN	NaN	NaN
29	NaN	NaN	NaN	NaN	NaN	NaN	NaN
30	NaN	NaN	NaN	NaN	NaN	NaN	NaN
31	NaN	NaN	NaN	NaN	NaN	NaN	NaN
32	NaN	NaN	NaN	NaN	NaN	NaN	NaN

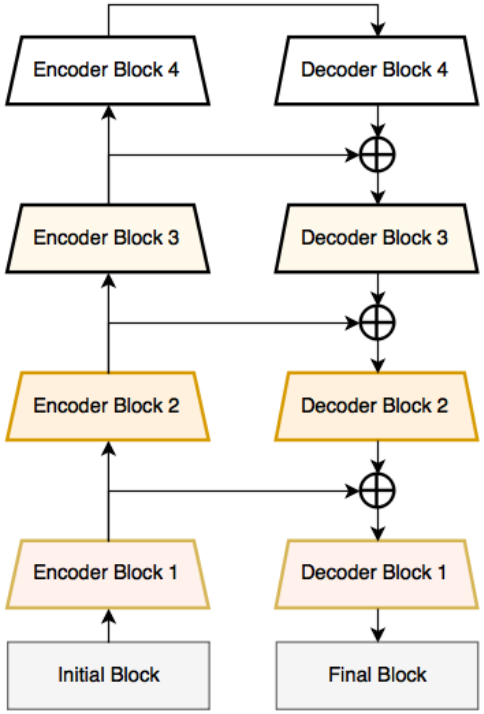
Project to 2D

Ground Truth Labels Transformed to Label Mask



Create Network Architecture

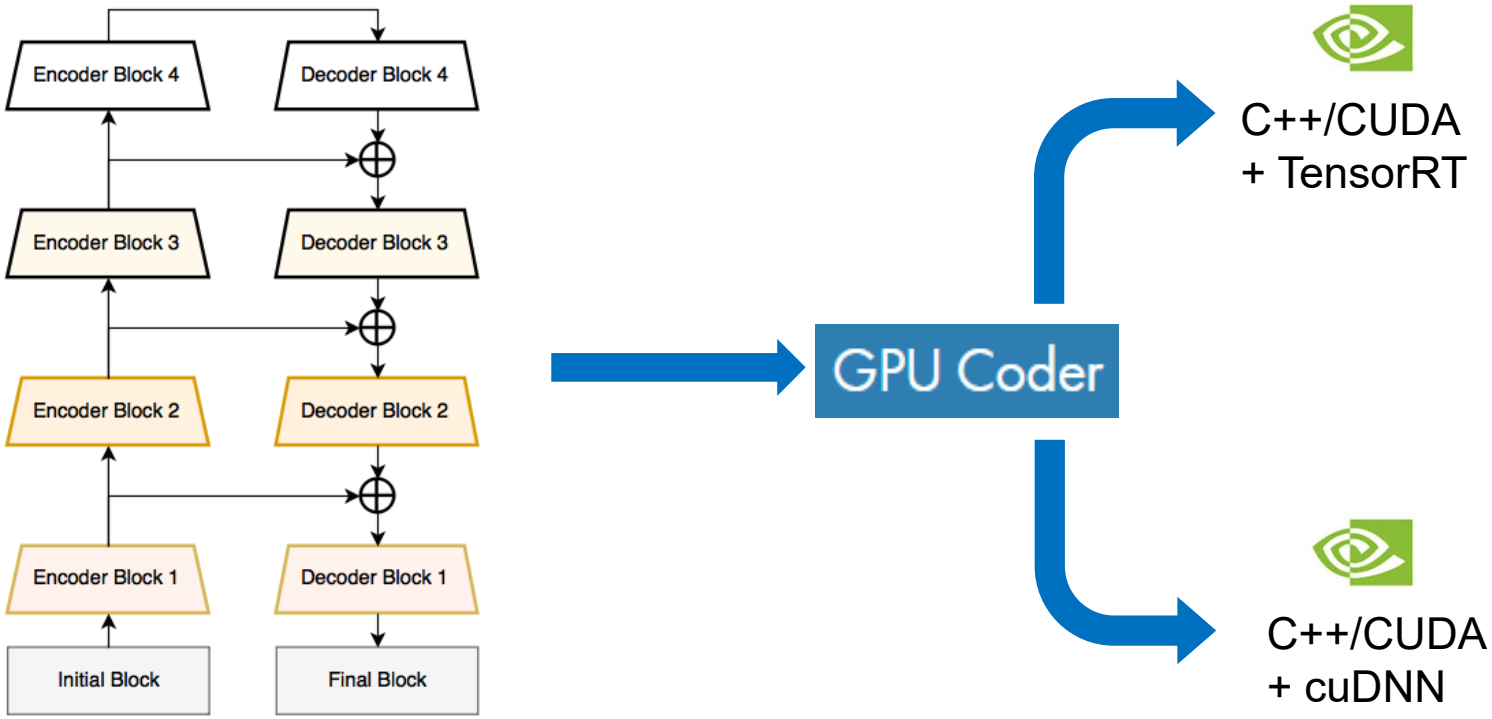




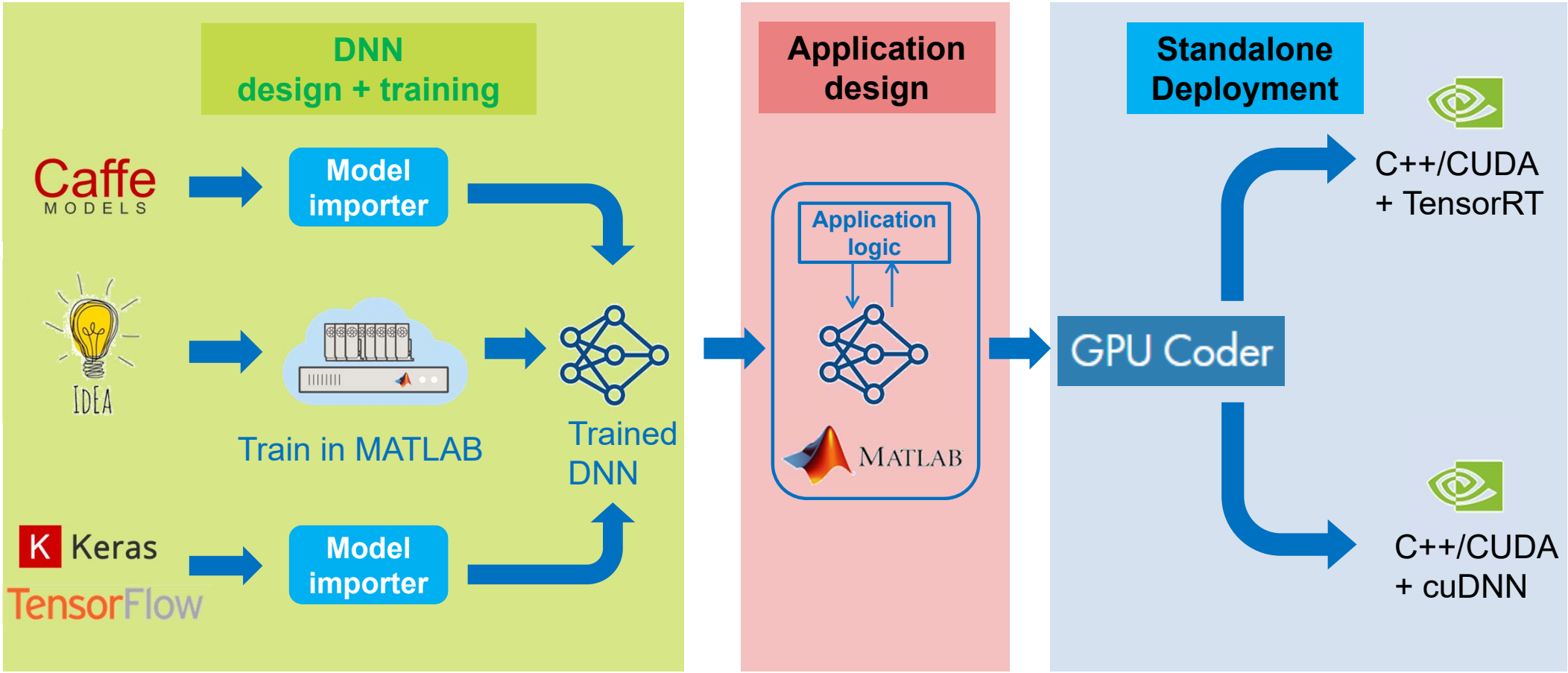
Easy API to create network structure

```
%build encoder
nOutputs = 64;
inputLayerName = 'init_maxpool';
for blockIdx = 1:encoderDepth
    [lGraph, layerOutName] = encoderBlock(lGraph, blockIdx, nOutputs, inputLayerName);
    nOutputs = nOutputs * 2;
    inputLayerName = layerOutName;
end

%build decoder
nInputs = nOutputs;
inputLayerName = layerOutName;
for blockIdx = encoderDepth:-1:1
    nOutputs = min(nInputs/2, 64);
    [lGraph, decoderLayerOutName] = decoderBlock(lGraph, blockIdx, nInputs, nOutputs, inputLayerName);
    if blockIdx ~= 1
        inputLayerName = ['res_add' num2str(blockIdx)];
        lGraph = addLayers(lGraph, additionLayer(2, 'Name', inputLayerName) );
        lGraph = connectLayers(lGraph, ['enc' num2str(blockIdx-1) '_addout'],  [inputLayerName '/in2']);
        lGraph = connectLayers(lGraph, decoderLayerOutName,  [inputLayerName '/in1']);
    end
    nInputs = nInputs/2;
end
```

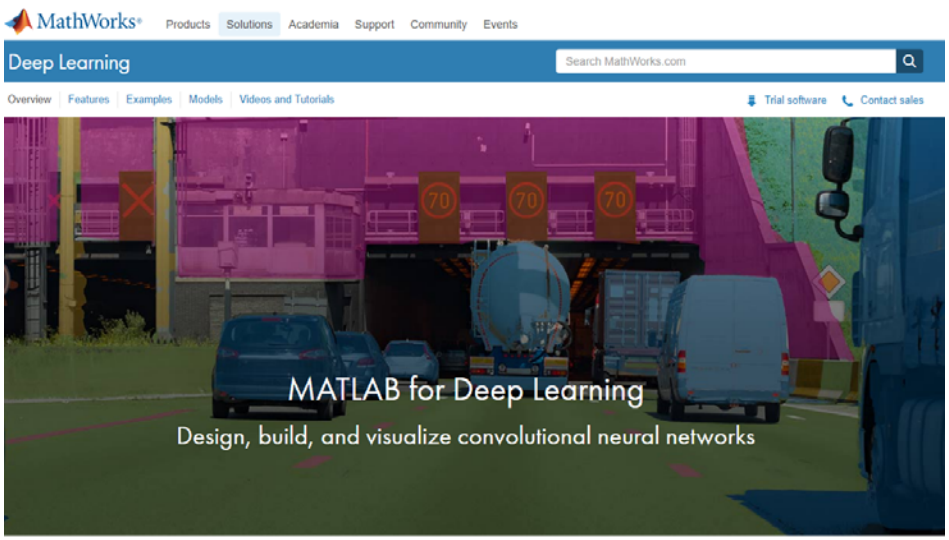
Deployment using GPU Coder



Deep learning workflow in MATLAB



Check Out Deep Learning in MATLAB and GPU Coder



GPU Coder
https://www.mathworks.com/products/gpu-coder.html

Deep learning in MATLAB
https://www.mathworks.com/solutions/deep-learning.html

Deep learning On-Ramp: Free self-paced, online training
https://matlabacademy.mathworks.com/R2017b/portal.html?course=deeplearning