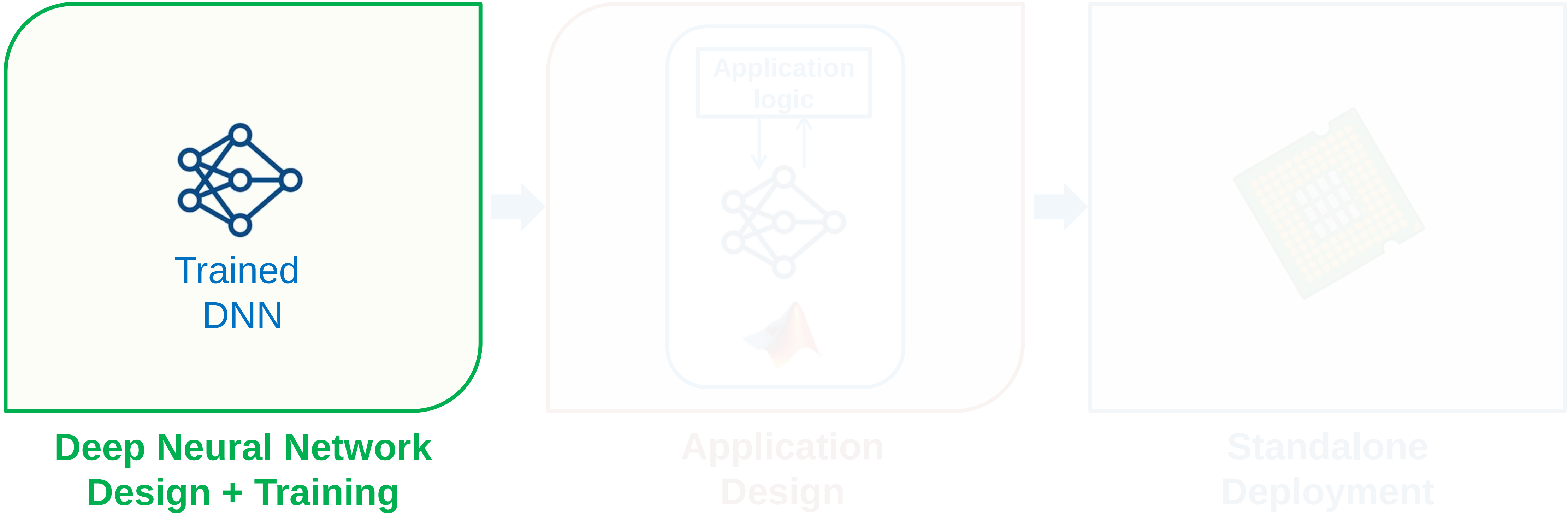
MATLAB EXPO 2019

Deploying Deep Neural Networks to Embedded GPUs and CPUs

Abhijit Bhattacharjee

Deep Learning Workflow in MATLAB

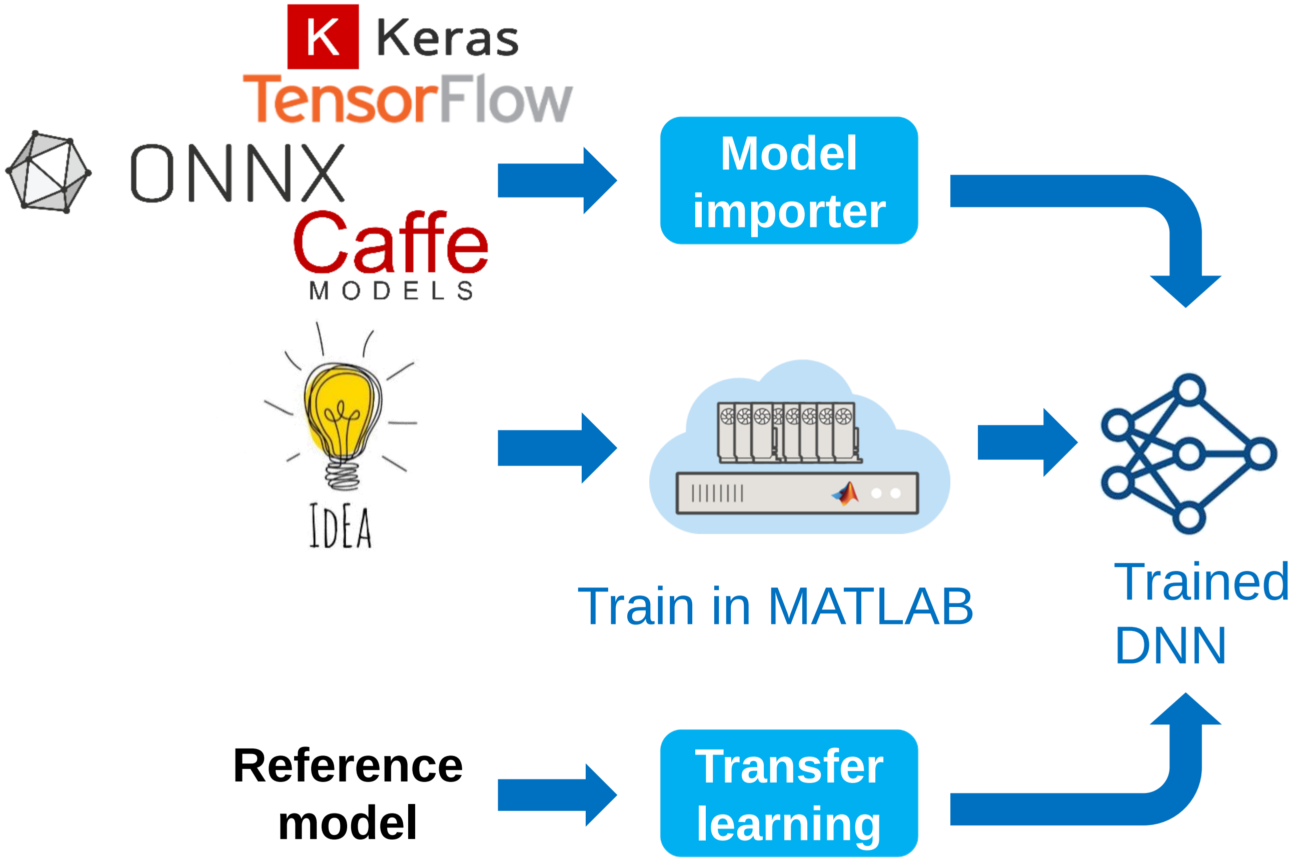


Deep Neural Network Design + Training

Application Design

Standalone Deployment

Deep Neural Network Design and Training



Model importer

Train in MATLAB

Trained DNN

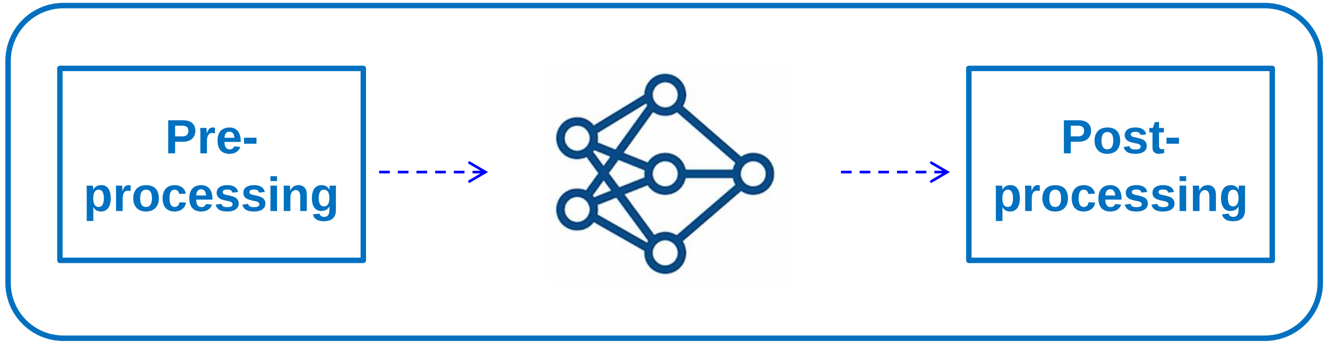
Reference model

Transfer learning

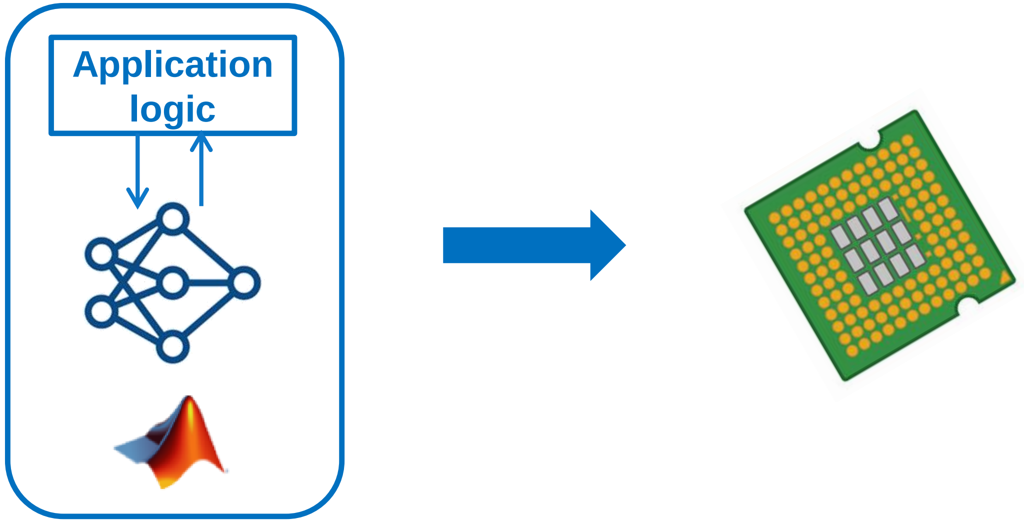
- **Design in MATLAB**
  - **Manage** large data sets
  - **Automate** data labeling
  - **Easy access** to models

- **Training in MATLAB**
  - **Acceleration** with GPU’s
  - **Scale** to clusters

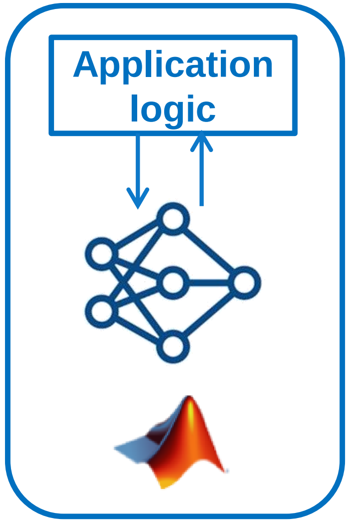
Application Design

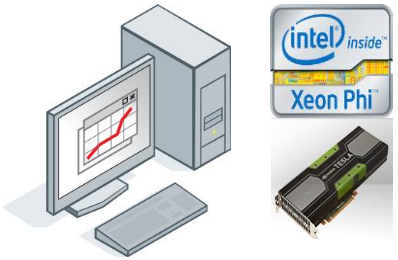


Multi-Platform Deep Learning Deployment



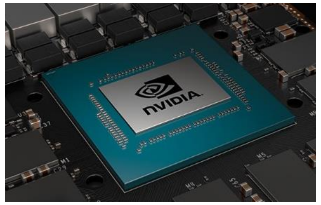
Multi-Platform Deep Learning Deployment



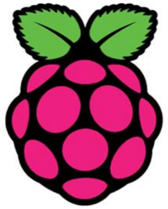


Desktop

Data Center

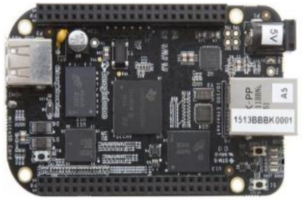


NVIDIA Jetson



Raspberry pi

Mobile

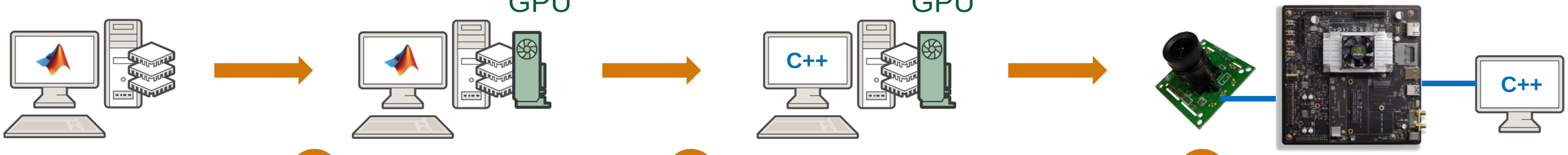


Beaglebone

. . .

Embedded

Algorithm Design to Embedded Deployment Workflow
Conventional Approach



Desktop GPU

Desktop GPU

Embedded GPU

1 Functional test

2 Deployment unit-test

3 Deployment integration-test

4 Real-time test

High-level language
Deep learning framework
Large, complex software stack

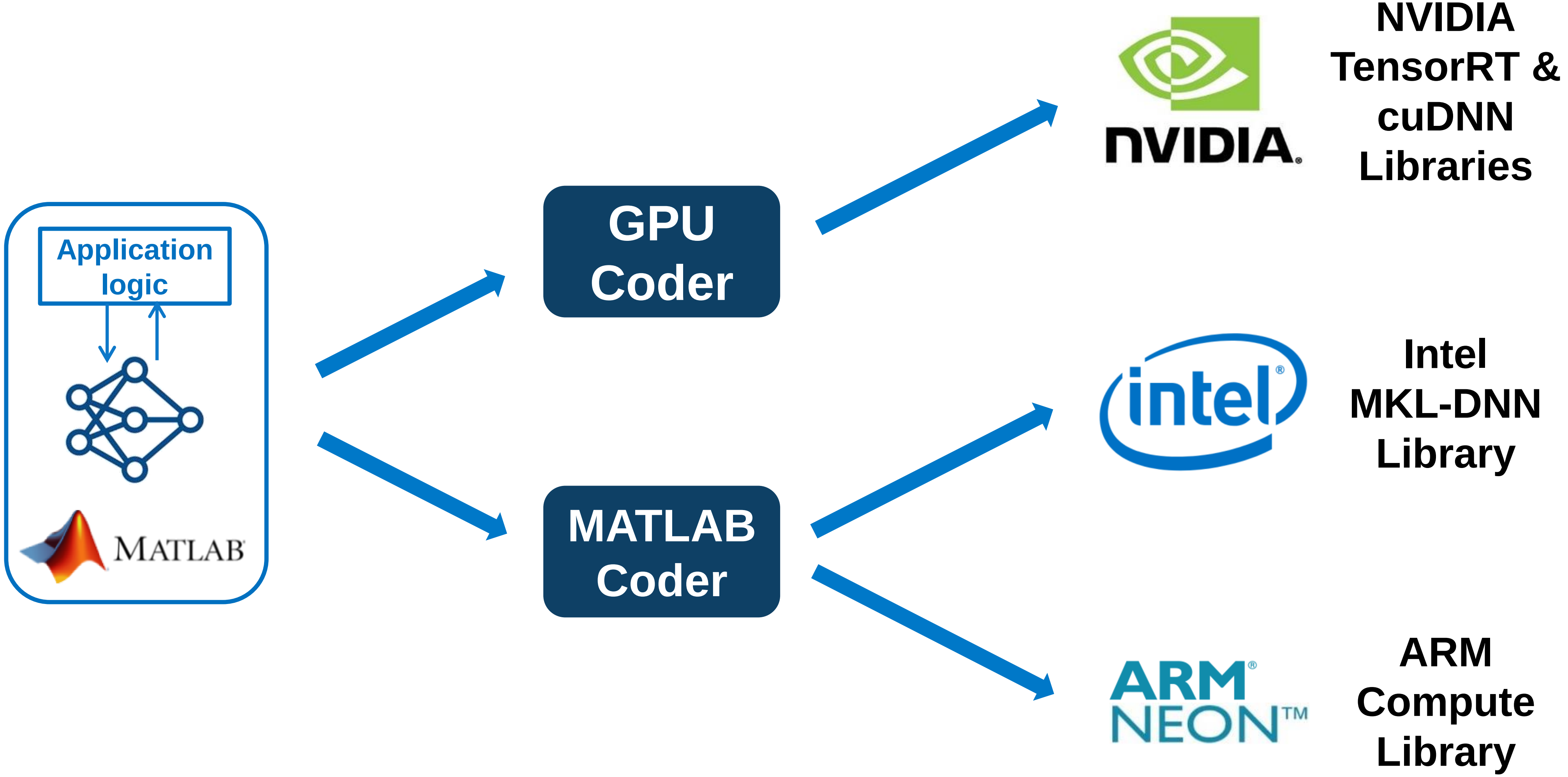
C/C++
Low-level APIs
Application-specific libraries

C/C++
Target-optimized libraries
Optimize for memory & speed

Challenges

- Integrating multiple libraries and packages
- Verifying and maintaining multiple implementations
- Algorithm & vendor lock-in

Solution: Use MATLAB Coder & GPU Coder for Deep Learning Deployment



Target Libraries

Application logic

GPU Coder

MATLAB Coder

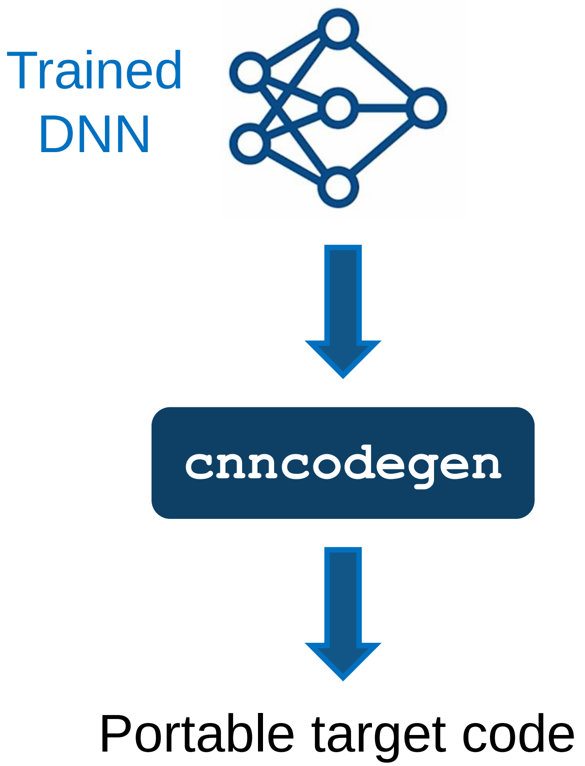
NVIDIA TensorRT & cuDNN Libraries

Intel MKL-DNN Library

ARM Compute Library

Deep Learning Deployment Workflows

INFERENCE ENGINE DEPLOYMENT

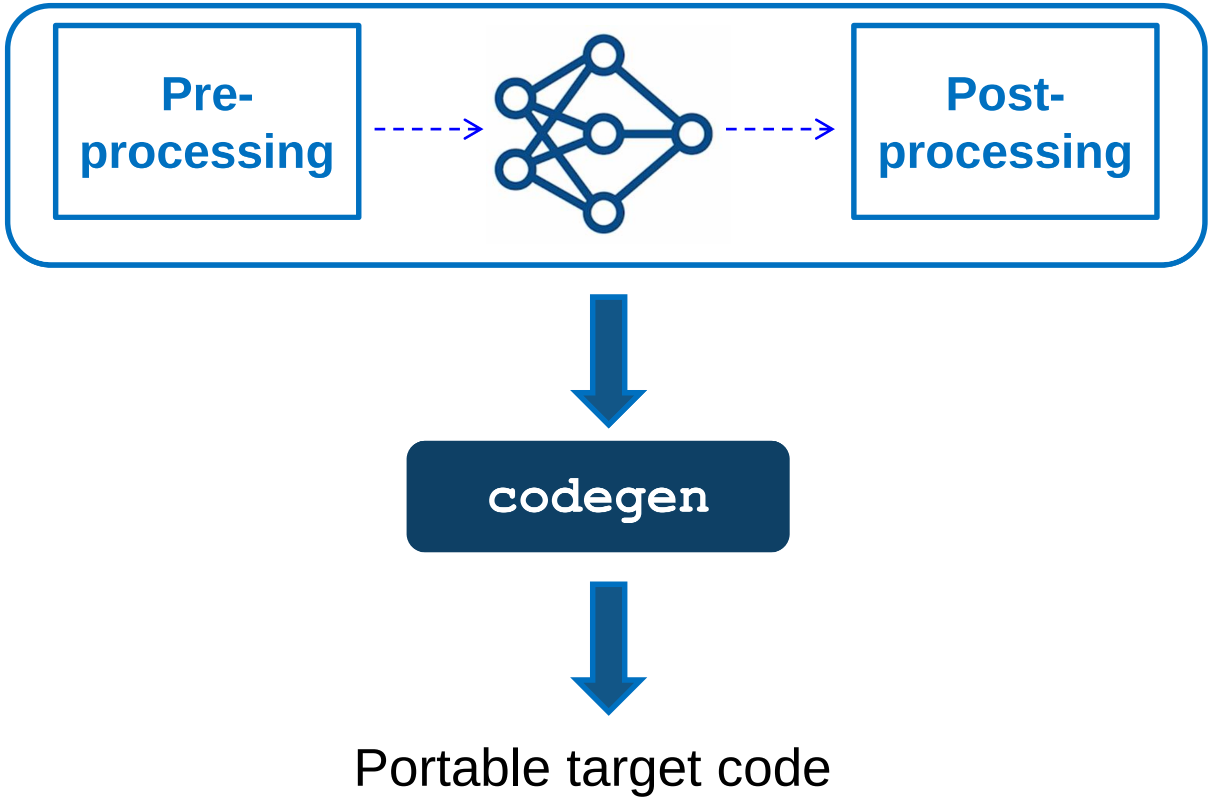


Trained DNN

cnncodegen

Portable target code

INTEGRATED APPLICATION DEPLOYMENT



Pre-processing

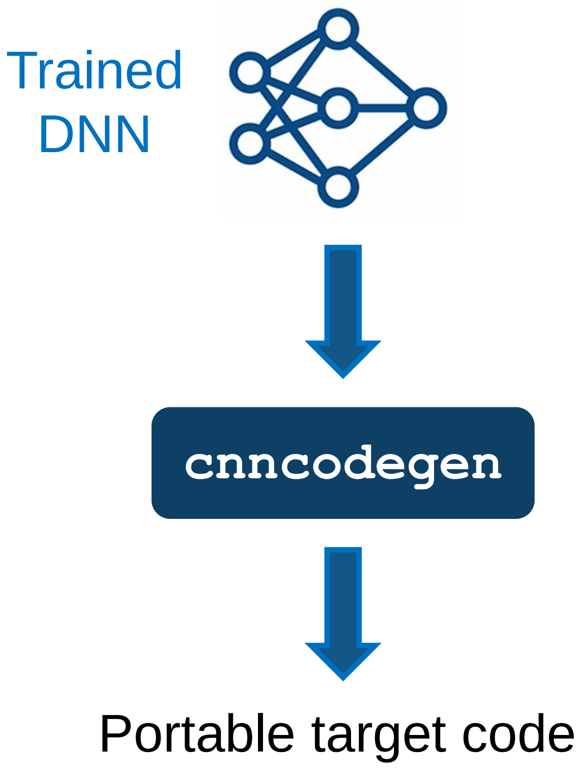
Post-processing

codegen

Portable target code

Workflow for Inference Engine Deployment

INFERENCE ENGINE DEPLOYMENT



Trained DNN

cnncodegen

Portable target code

Steps for inference engine deployment

1. Generate the code for trained model

```
>> cnncodegen(net, 'targetlib’, ‘arm-compute’)
```

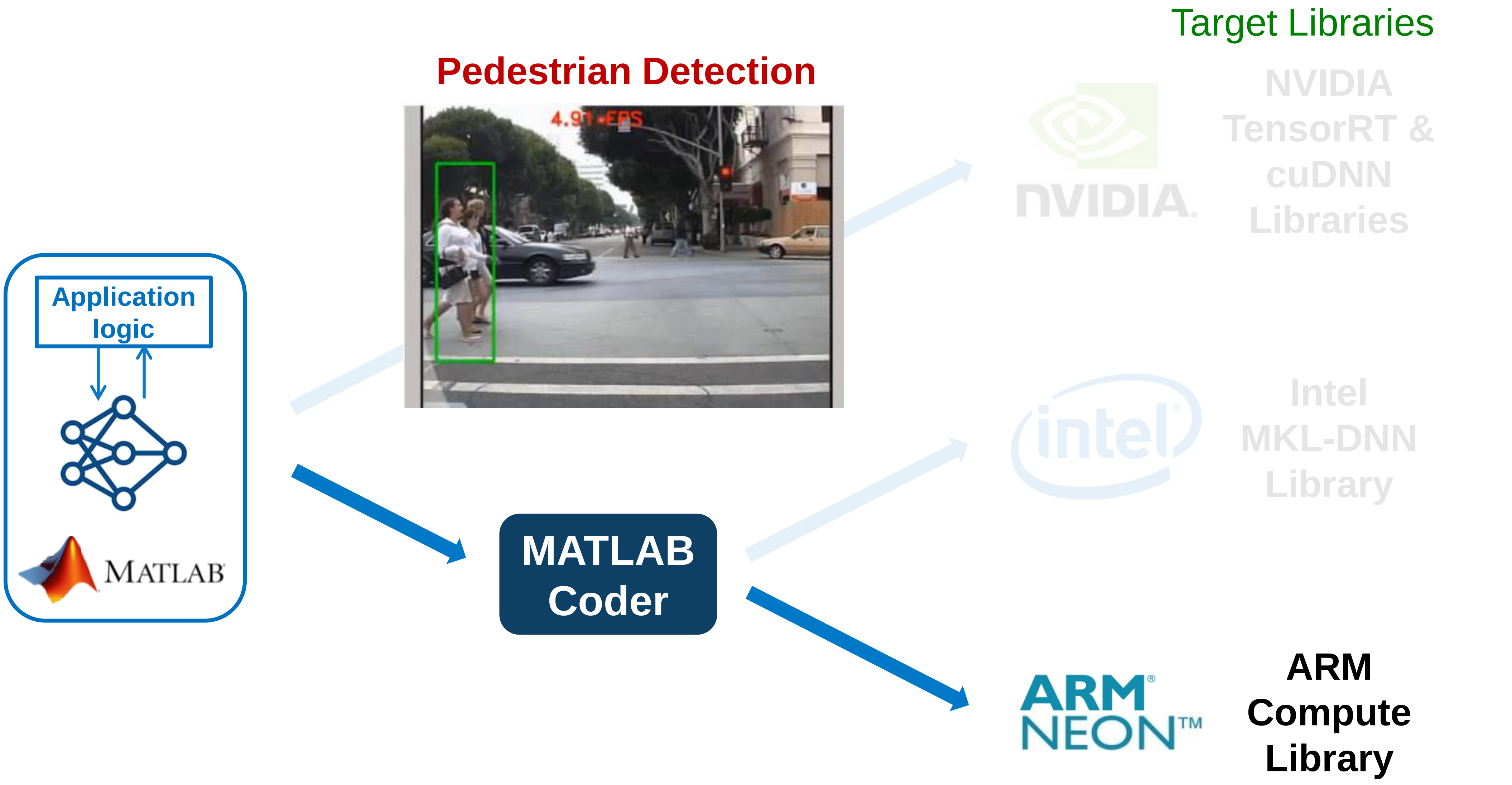
2. Copy the generated code onto target board

3. Use hand written main function to call inference engine

4. Generate the exe and test the executable

```
>> make -C ./ ……
```

Deep Learning Inference Deployment



Target Libraries

Pedestrian Detection

Application logic

MATLAB Coder

NVIDIA TensorRT & cuDNN Libraries

Intel MKL-DNN Library

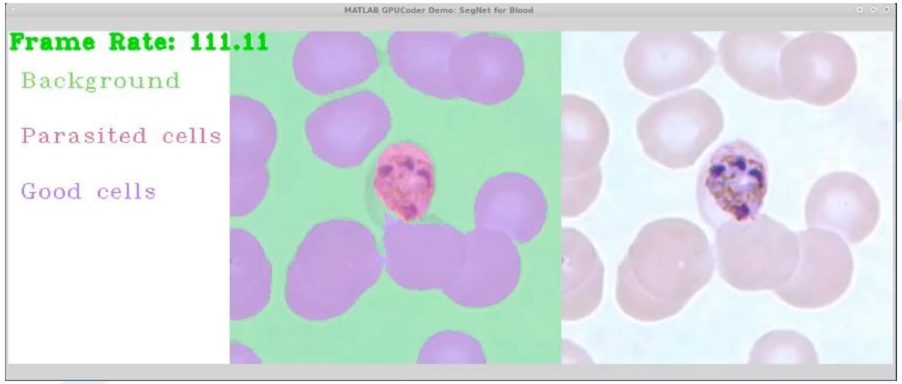
ARM Compute Library

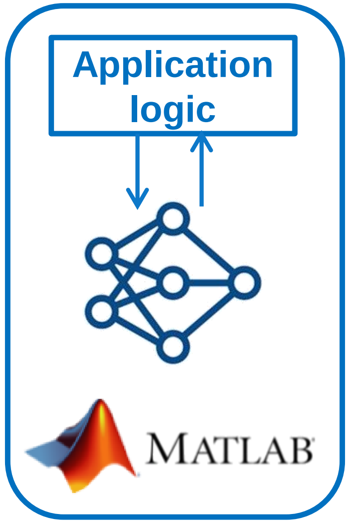
Includes ARM Cortex-A support

Deep Learning Inference Deployment

Target Libraries

Blood Smear Segmentation





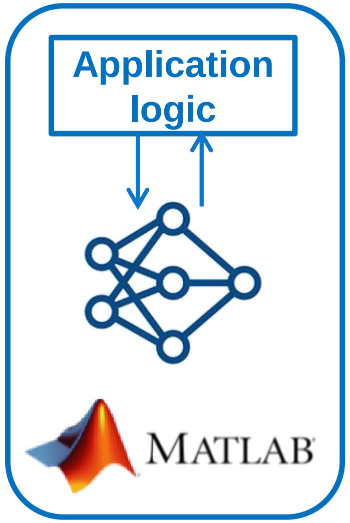
MATLAB Coder

NVIDIA TensorRT & cuDNN Libraries

Intel MKL-DNN Library

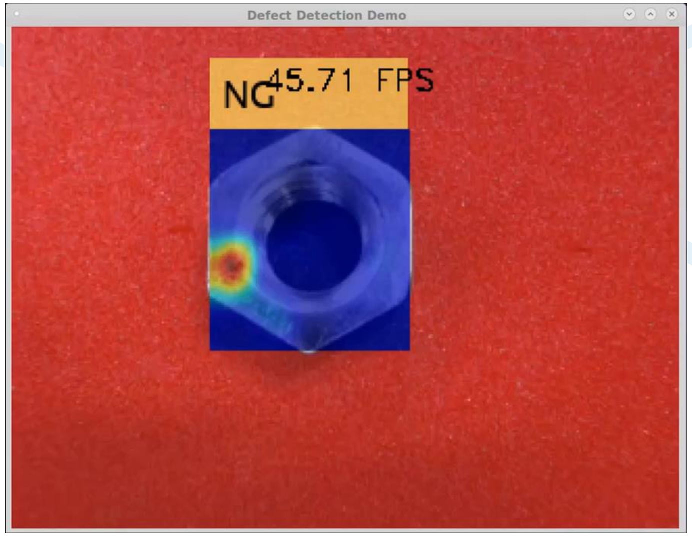
ARM Compute Library

Deep Learning Inference Deployment



GPU Coder

Defect Classification & Detection



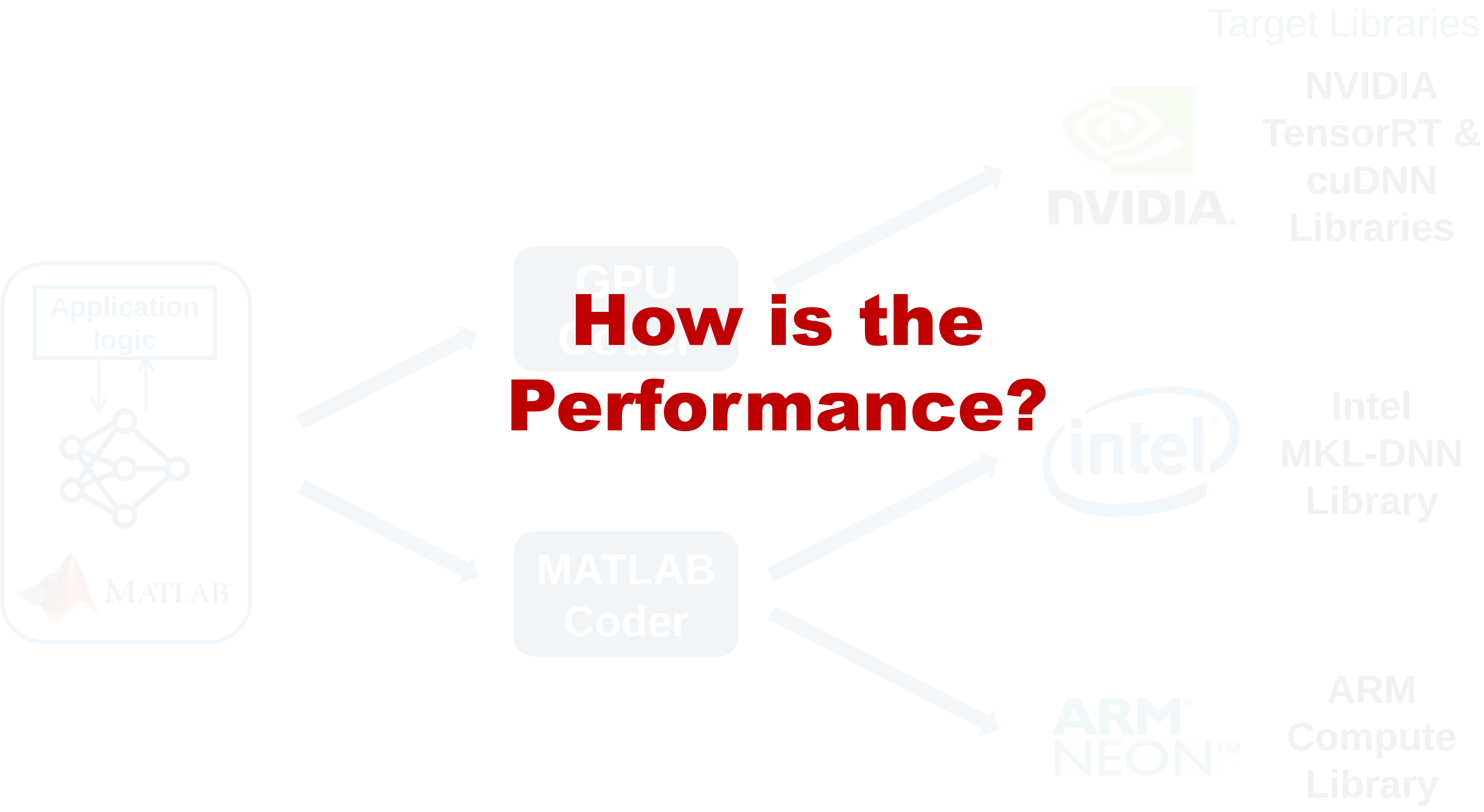
Target Libraries

NVIDIA TensorRT & cuDNN Libraries

Intel MKL-DNN Library

ARM Compute Library

How is the Performance?



Application logic

MATLAB

GPU Coder

MATLAB Coder

Target Libraries

NVIDIA TensorRT & cuDNN Libraries

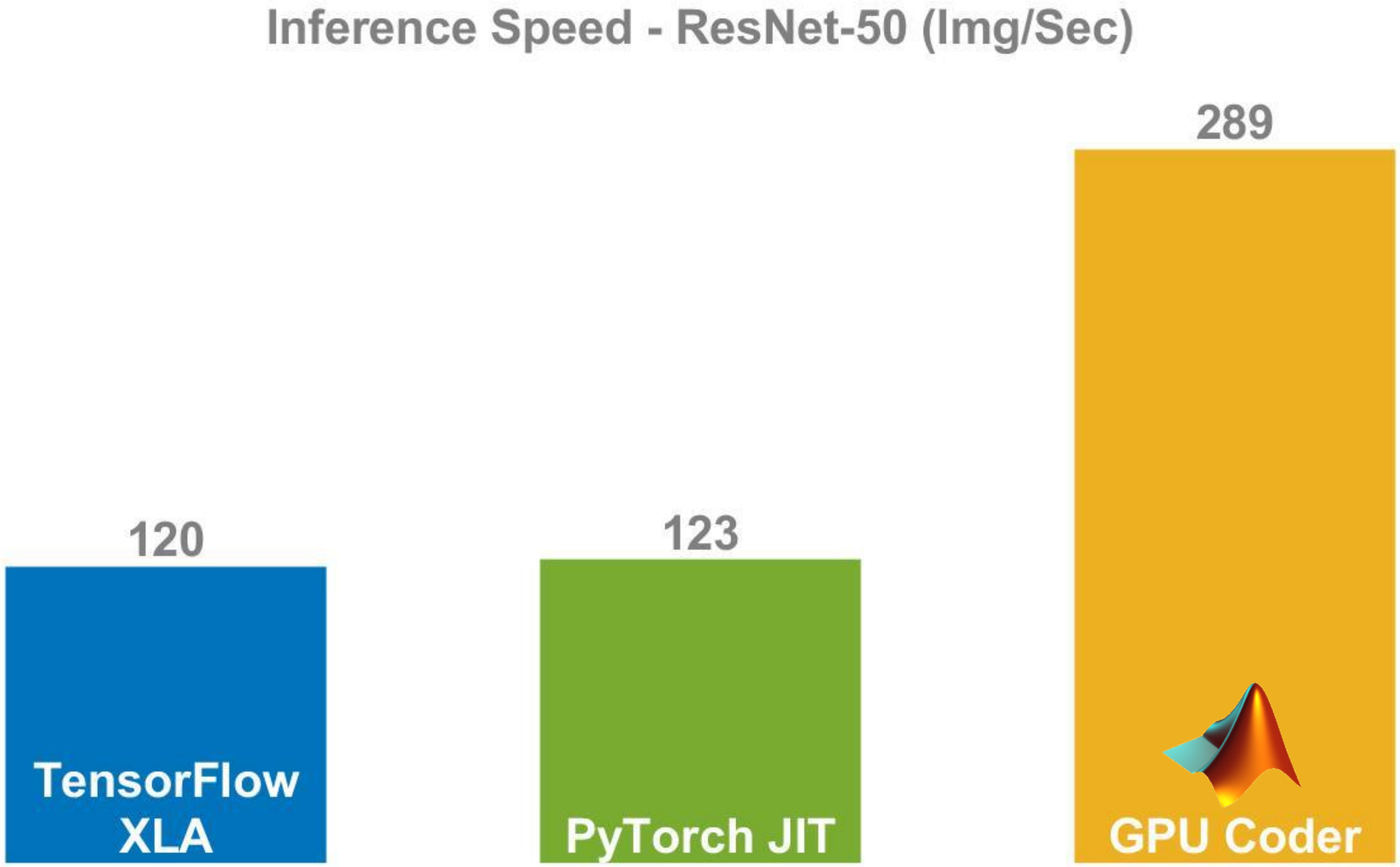
Intel MKL-DNN Library

ARM Compute Library

Performance of Generated Code

- CNN inference (ResNet-50, VGG-16, Inception V3) on Titan V GPU
- CNN inference (ResNet-50) on Jetson TX2
- CNN inference (ResNet-50 , VGG-16, Inception V3) on Intel Xeon CPU

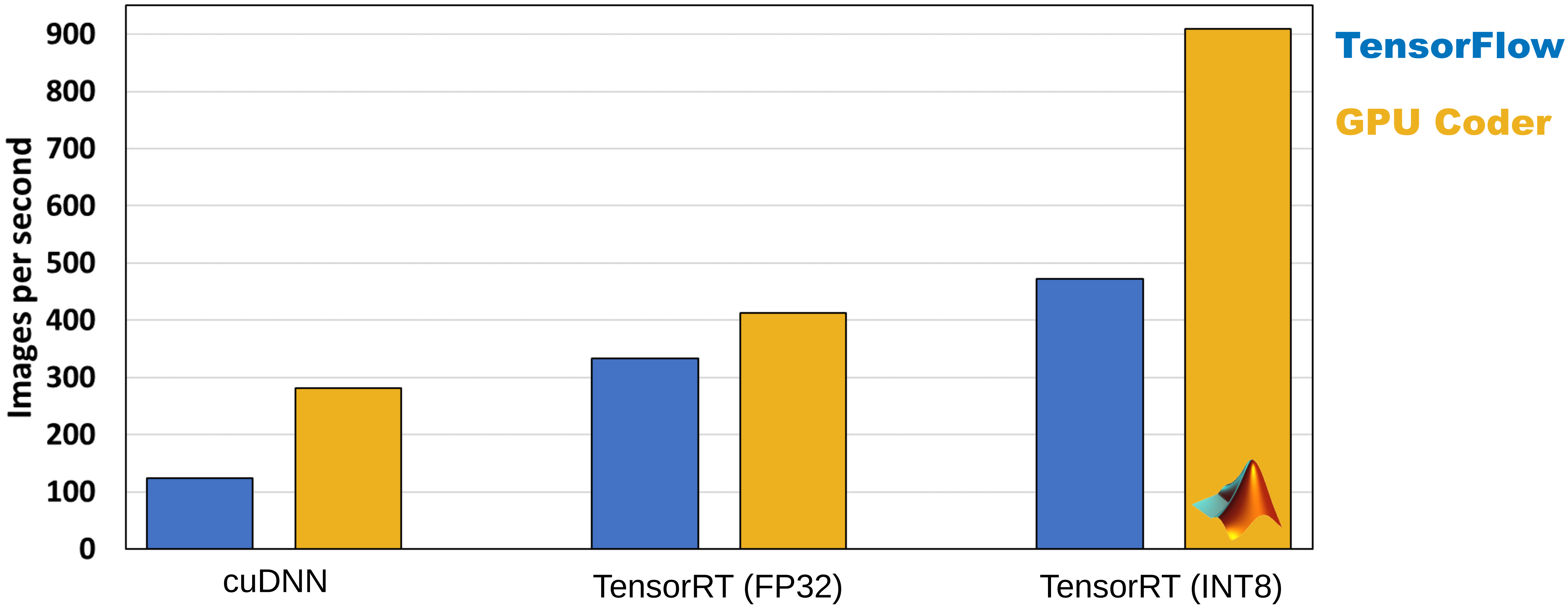
Single Image Inference on Titan V using cuDNN



Intel® Xeon® CPU 3.6 GHz - NVIDIA libraries: CUDA10.0/1 - cuDNN 7.5.0 - Frameworks: TensorFlow 1.13.1, MXNet 1.4.1 PyTorch 1.1

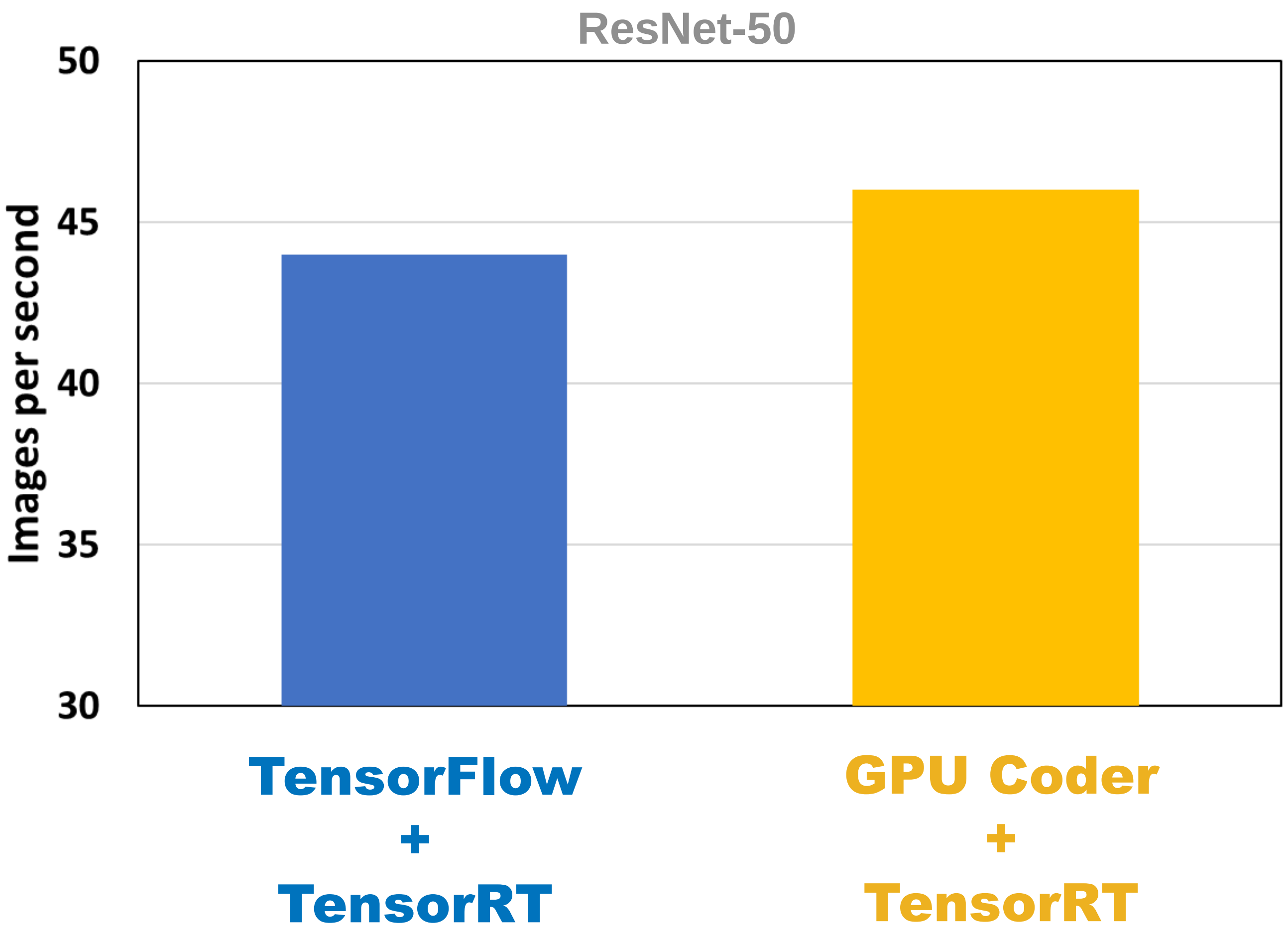
TensorRT Accelerates Inference Performance on Titan V

Single Image Inference with ResNet-50 (Titan V)



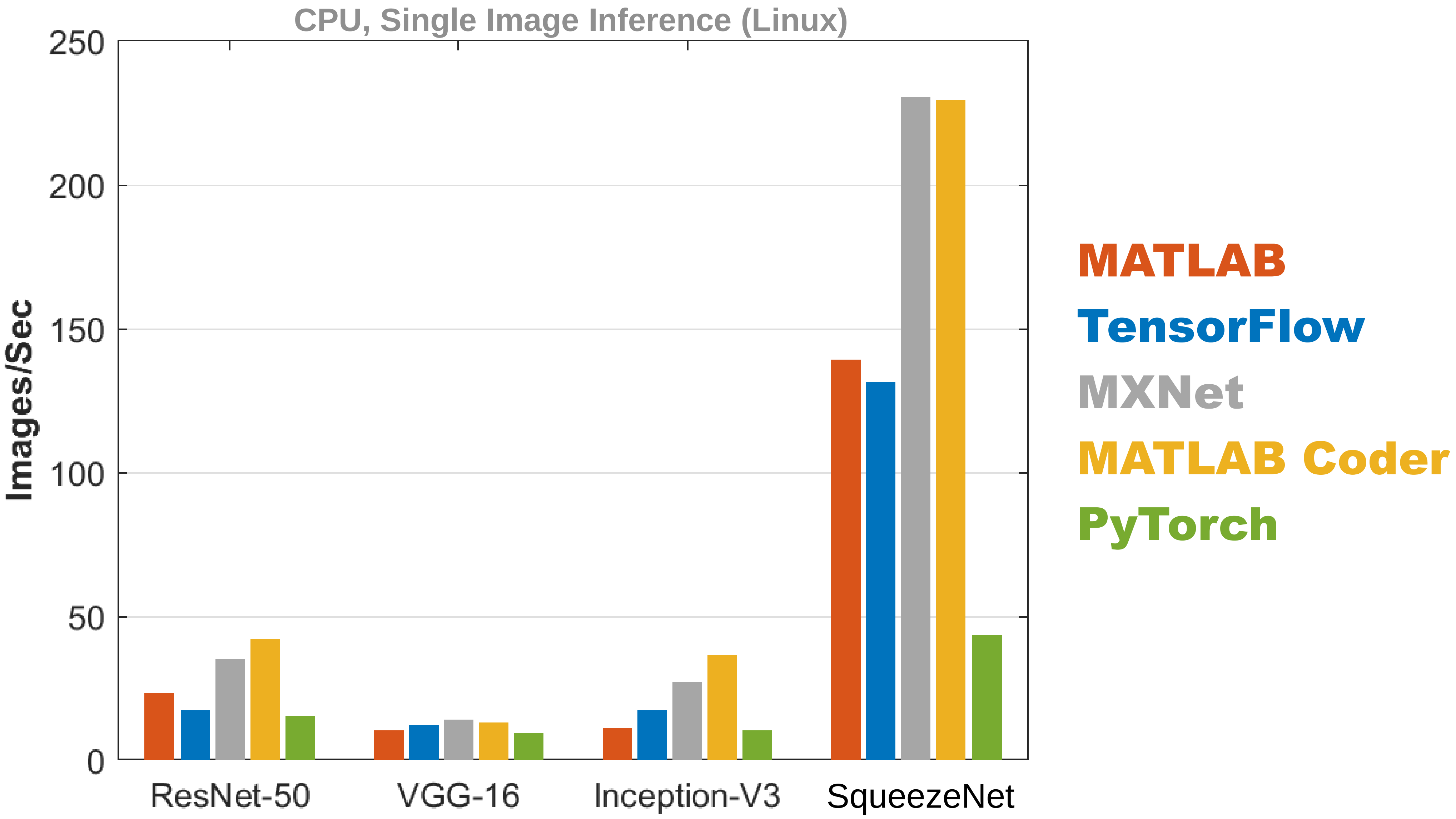
Intel® Xeon® CPU 3.6 GHz - NVIDIA libraries: CUDA10.0/1 - cuDNN 7.5.0 - TensorRT 5.1.2 - Frameworks: TensorFlow 1.13.1

Single Image Inference on Jetson TX2



NVIDIA libraries: CUDA10.0/1 - cuDNN 7.5.0 - TensorRT 5.1.2 - Frameworks: TensorFlow 1.13.1

CPU Performance



CPU, Single Image Inference (Linux)

MATLA

Intel® Xeon® CPU 3.6 GHz - NVIDIA libraries: CUDA10.0/1 - cuDNN 7.5.0 - Frameworks: TensorFlow 1.13.1, MXNet 1.4.1 PyTorch 1.1

Brief Summary

DNN libraries are great for inference, …

MATLAB Coder and GPU Coder generates code that takes advantage of:

NVIDIA® CUDA libraries, including TensorRT & cuDNN

Intel® Math Kernel Library for Deep Neural Networks (MKL-DNN)

ARM® Compute libraries for mobile platforms

Brief Summary

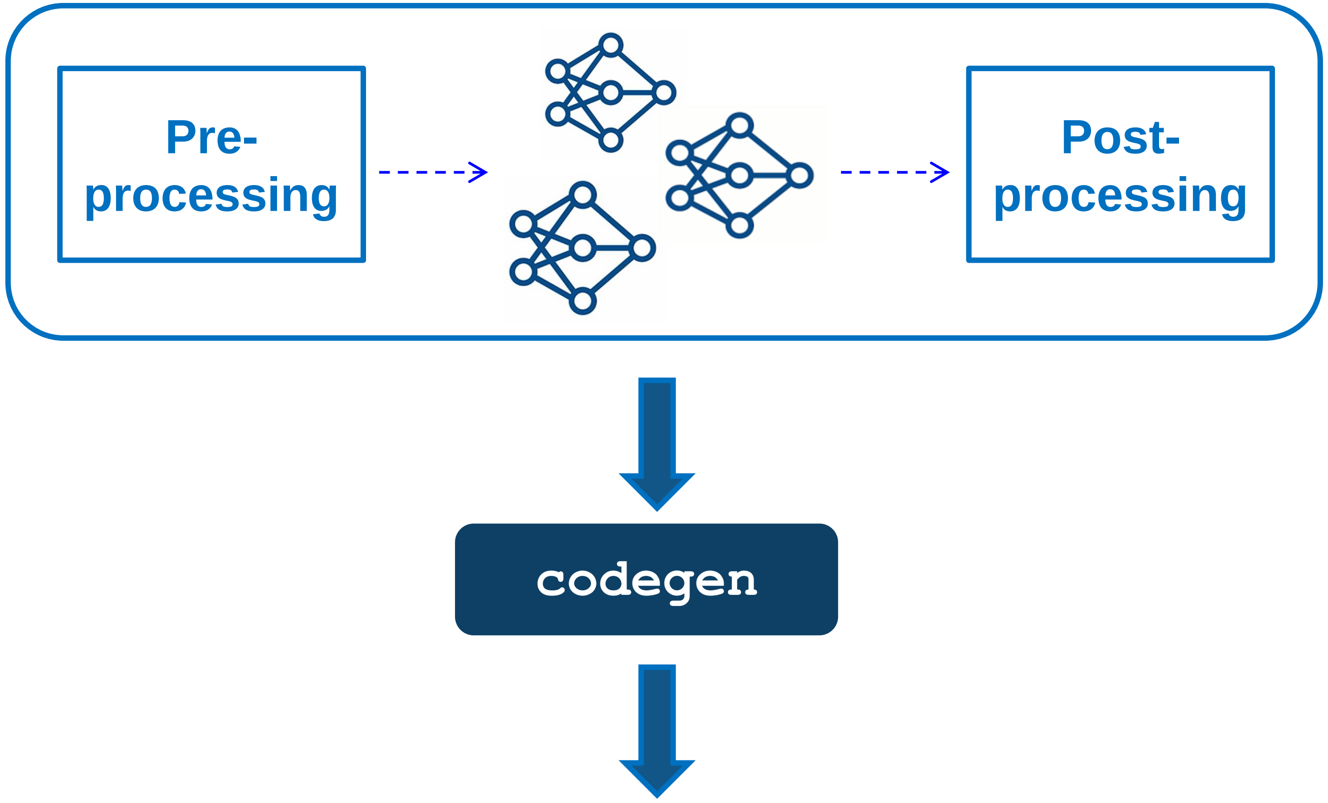
DNN libraries are great for inference, …

MATLAB Coder and GPU Coder generates code that takes advantage of:

- NVIDIA® CUDA libraries, including TensorRT & cuDNN
- Intel® Math Kernel Library for Deep Neural Networks (MKL-DNN)
- ARM® Compute libraries for mobile platforms

But, Applications Require More than just Inference

Deep Learning Workflows: Integrated Application Deployment



Pre-processing

Post-processing

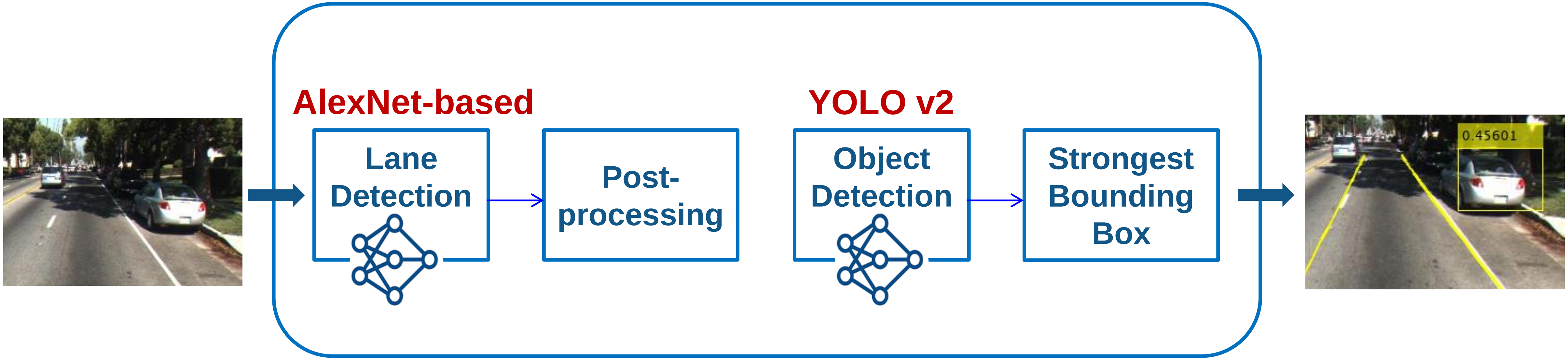
codegen

Portable target code

intel NVIDIA ARM NEON

Lane and Object Detection using YOLO v2

MATLAB



AlexNet-based

Lane Detection → Post-processing

YOLO v2

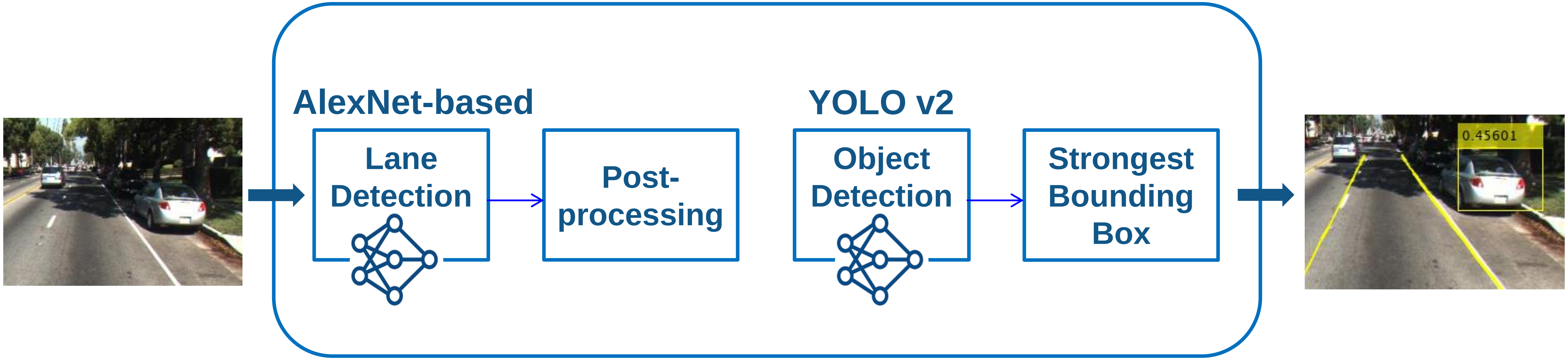
Object Detection → Strongest Bounding Box

Workflow:

1) Test in MATLAB on CPU
2) Generate code and test on desktop GPU
3) Generate code and test on Jetson AGX Xavier GPU

(1) Test in MATLAB on CPU

MATLAB



AlexNet-based

Lane Detection

Post-processing

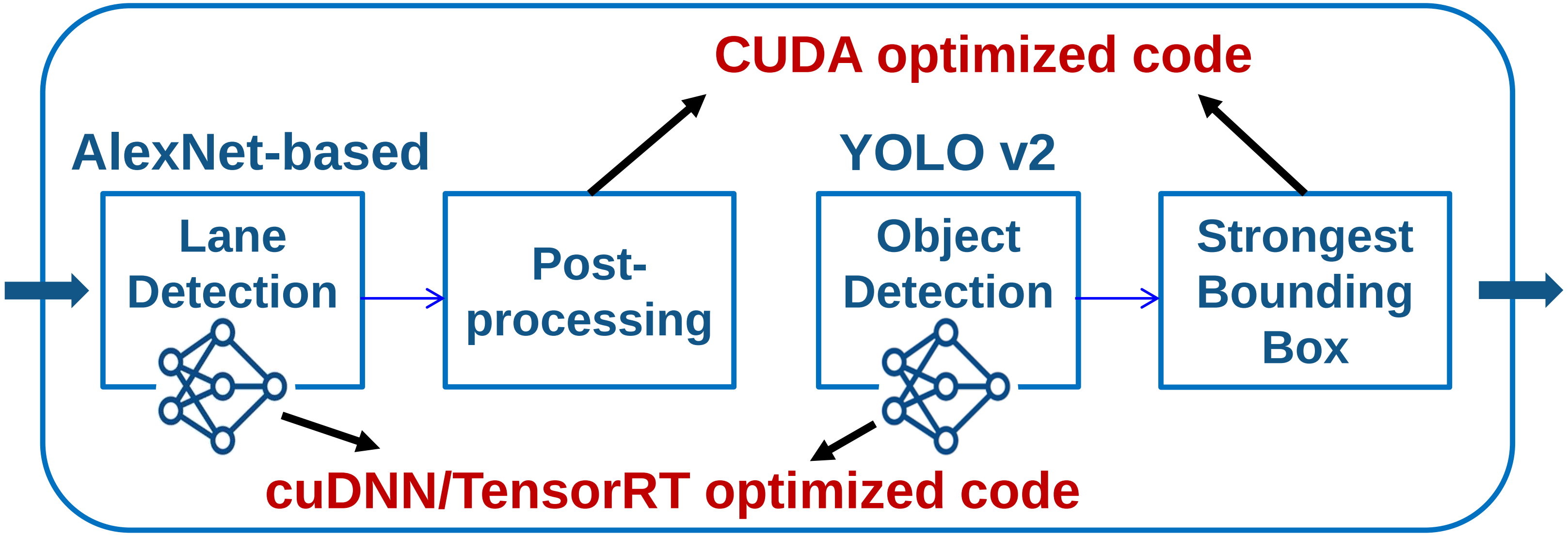
YOLO v2

Object Detection

Strongest Bounding Box

(2) Generate Code and Test on Desktop GPU

MATLAB



CUDA optimized code

AlexNet-based

Lane Detection

Post-processing

YOLO v2

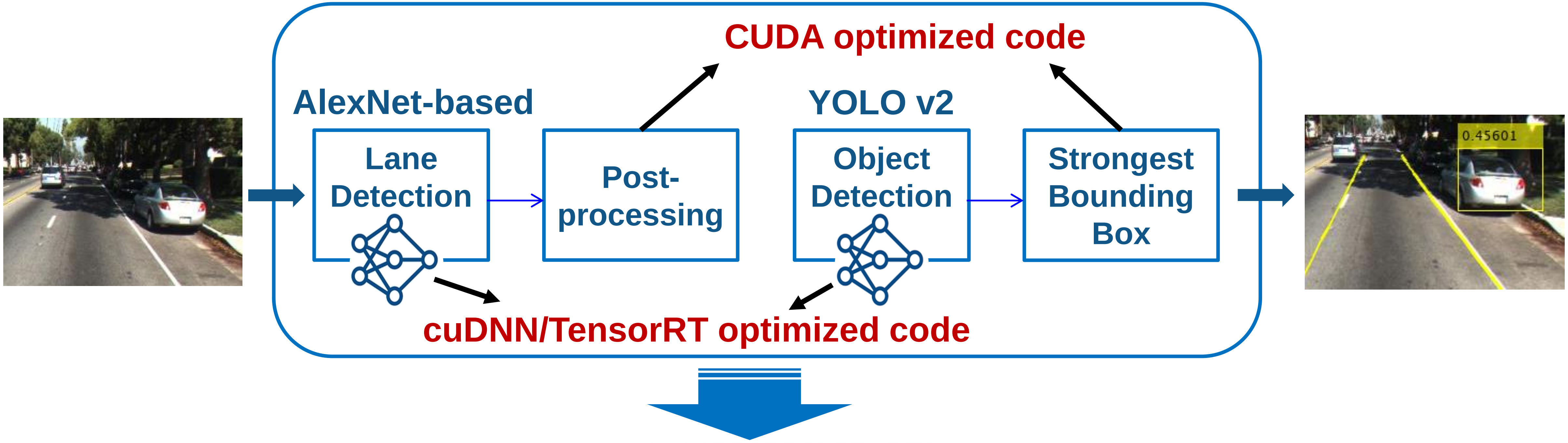
Object Detection

Strongest Bounding Box

cuDNN/TensorRT optimized code

(3) Generate Code and Test on Jetson AGX Xavier GPU

MATLAB



CUDA optimized code

AlexNet-based

Lane Detection

Post-processing

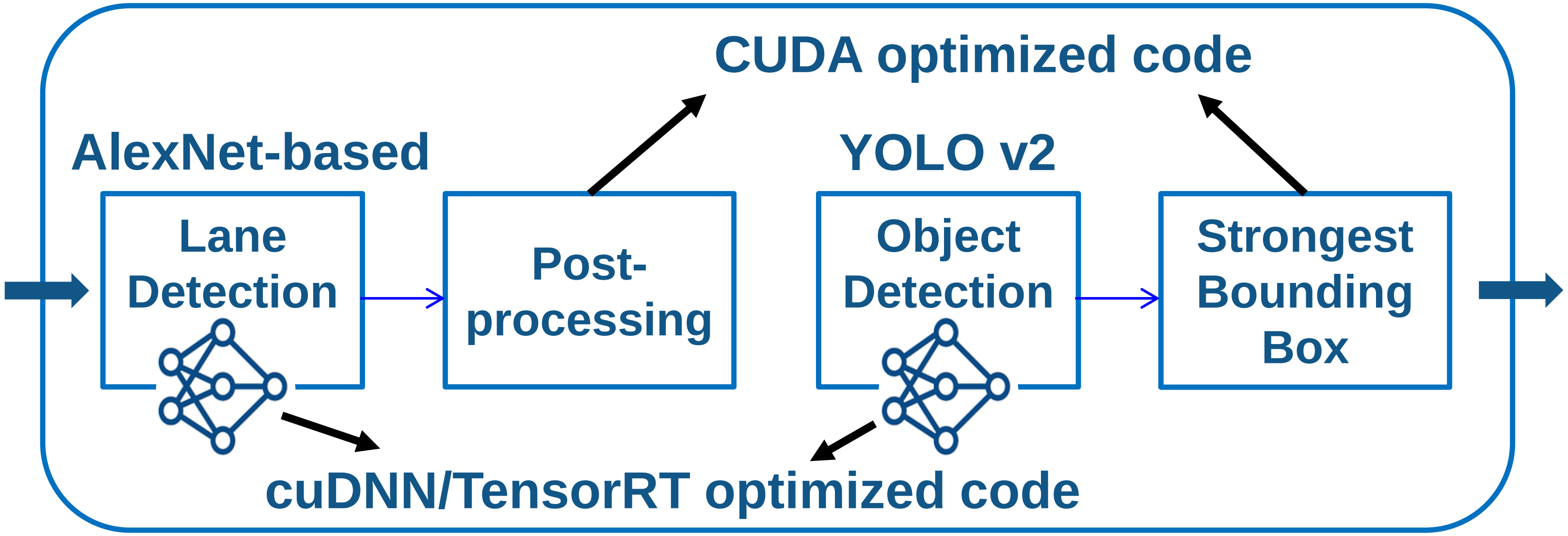
YOLO v2

Object Detection

Strongest Bounding Box

cuDNN/TensorRT optimized code

Lane and Object Detection using YOLO v2



MATLAB

CUDA optimized code

AlexNet-based

Lane Detection

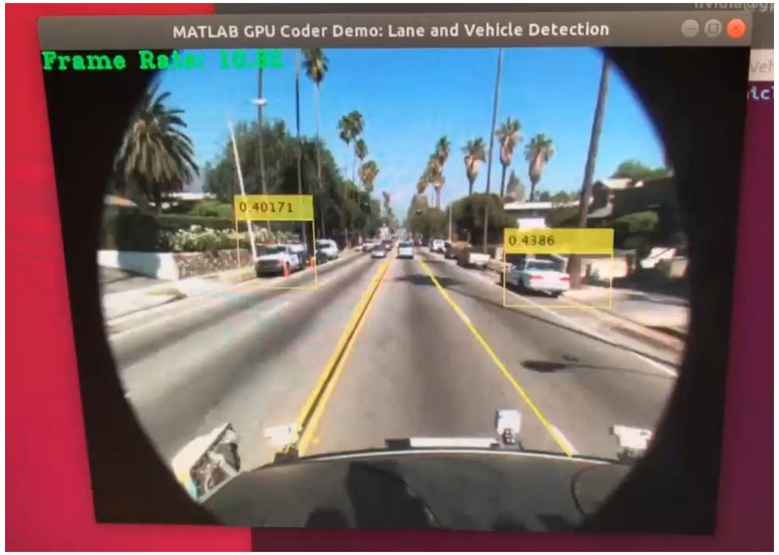
Post-processing

YOLO v2

Object Detection

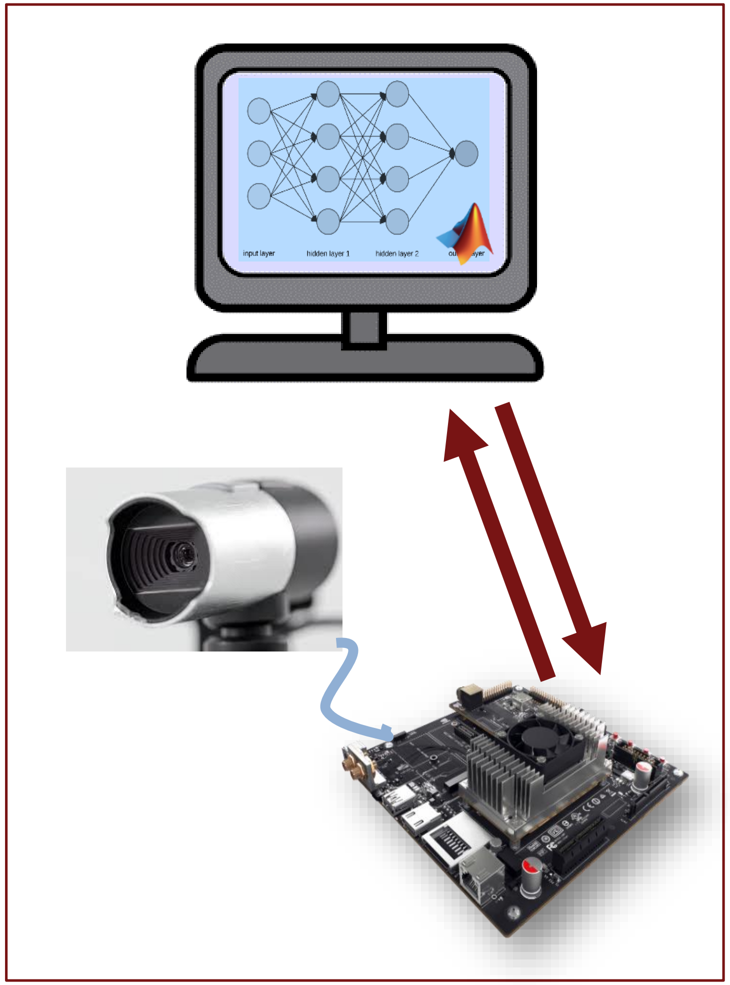
Strongest Bounding Box

cuDNN/TensorRT optimized code

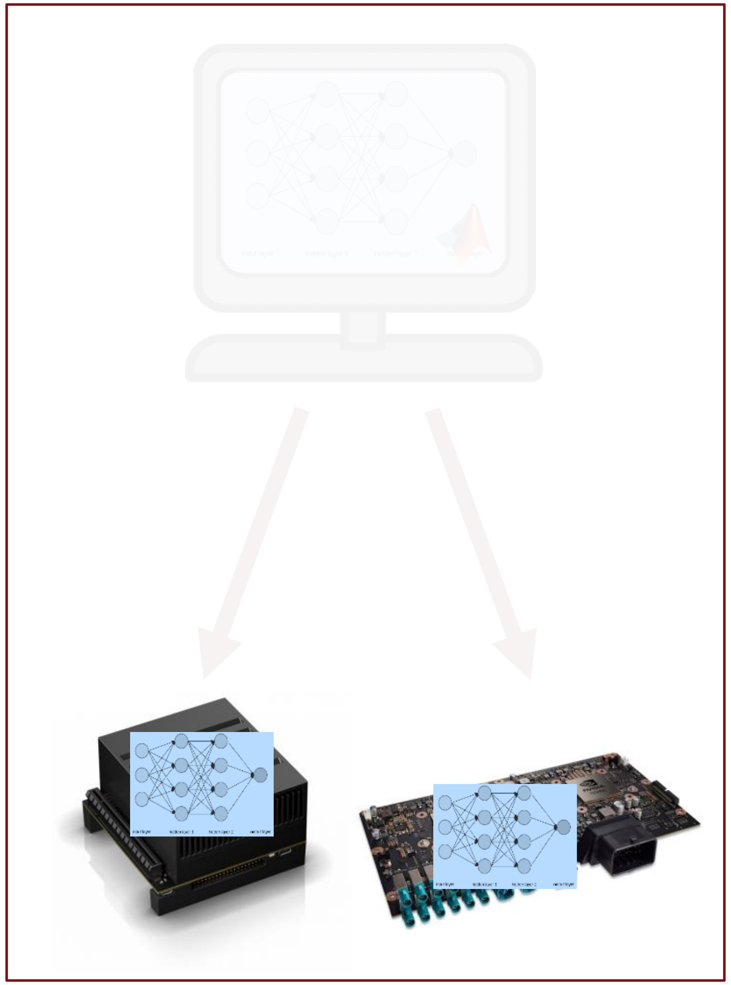


1) Running on CPU
2) 7X faster running generate code on desktop GPU
3) Generate code and test on Jetson AGX Xavier GPU

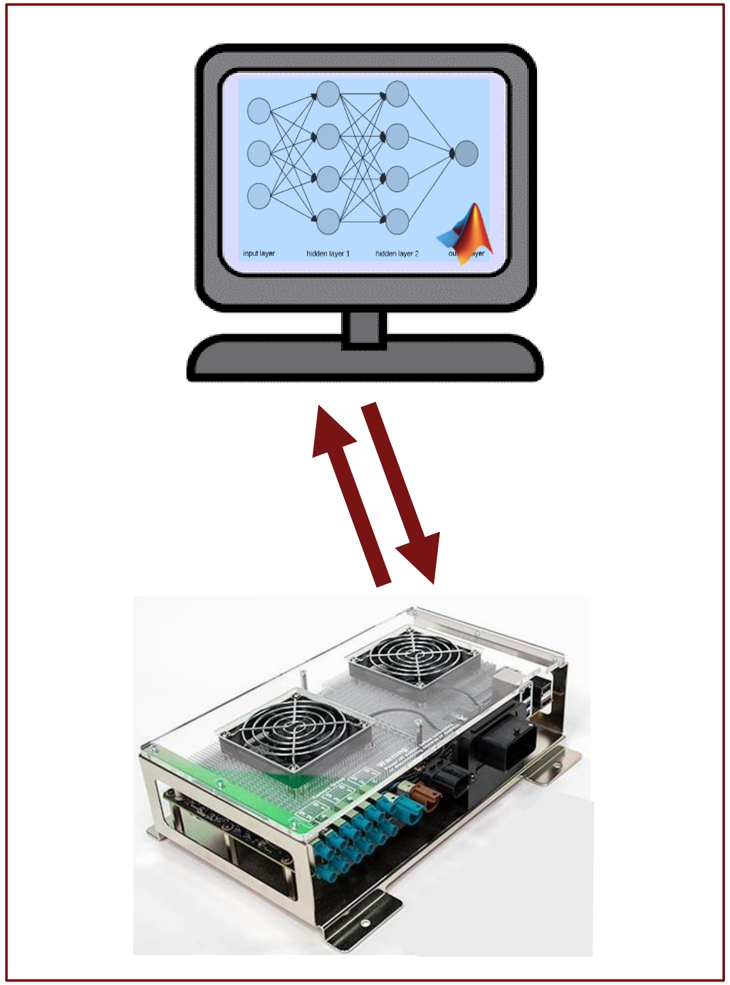
Accessing Hardware



Access Peripheral from MATLAB

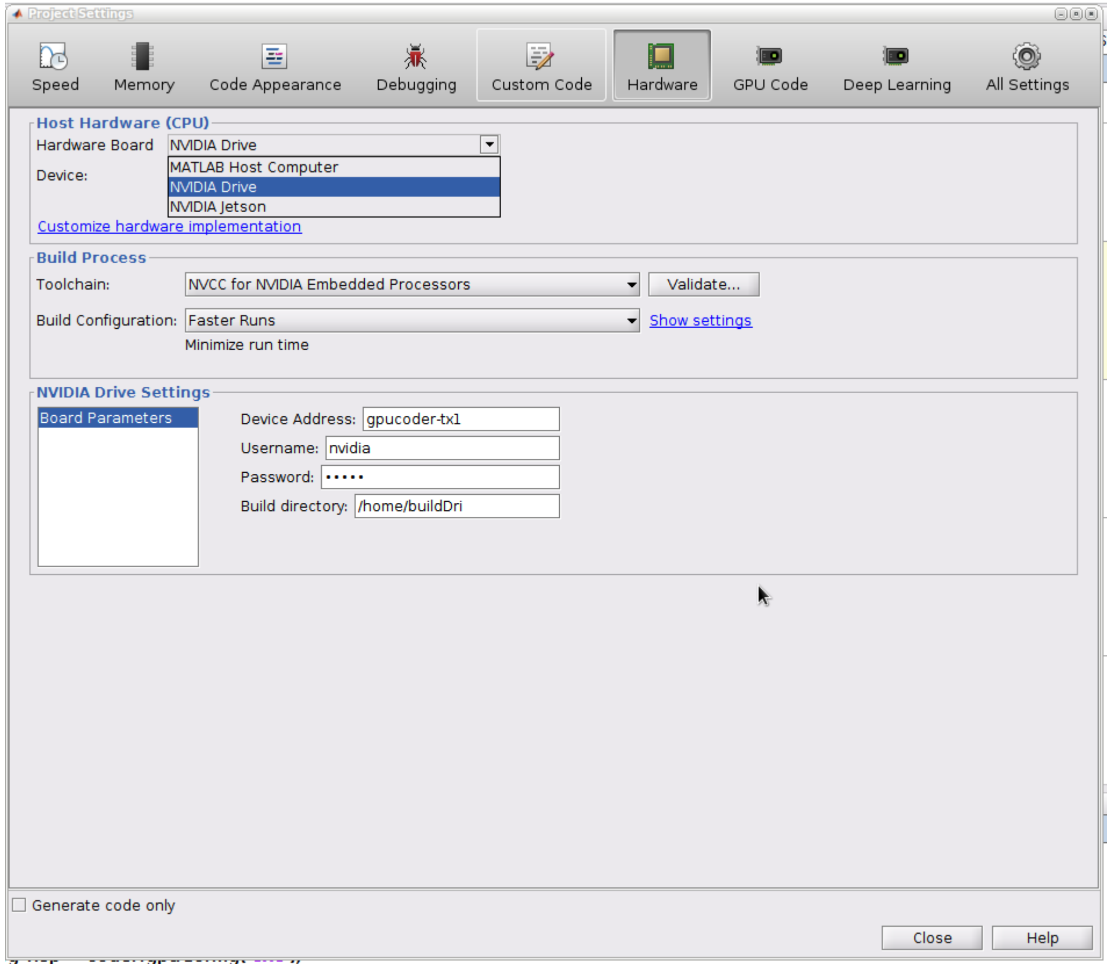


Deploy Standalone Application



Processor-in-Loop Verification

Deploy to Target Hardware via Apps and Command Line



Host Hardware (CPU)

Hardware Board: NVIDIA Drive

- MATLAB Host Computer
- NVIDIA Drive
- NVIDIA Jetson

Device:

Customize hardware implementation

Build Process

Toolchain: NVCC for NVIDIA Embedded Processors — Validate...

Build Configuration: Faster Runs — Show settings

Minimize run time

NVIDIA Drive Settings

Board Parameters

Device Address: gpucoder-tx1

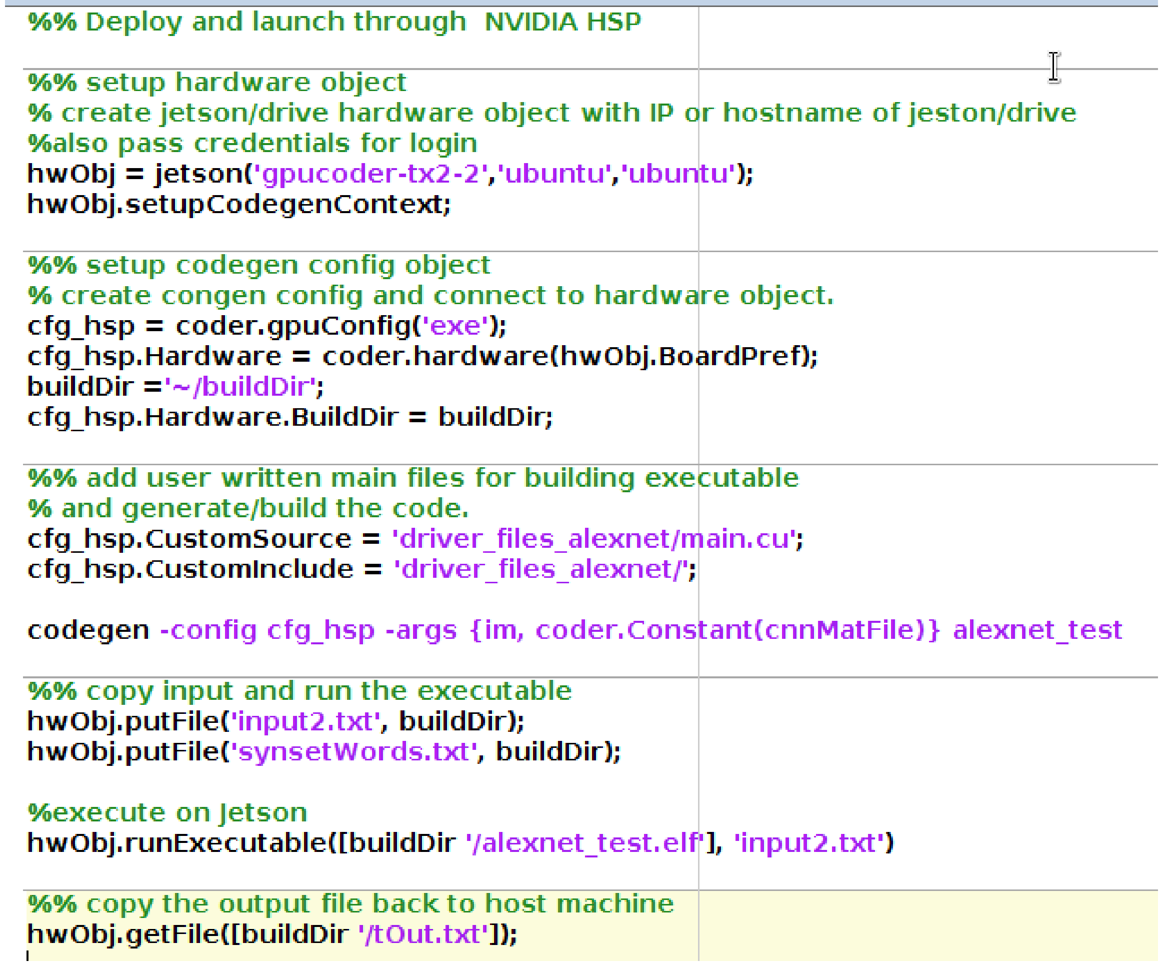
Username: nvidia

Password: •••••

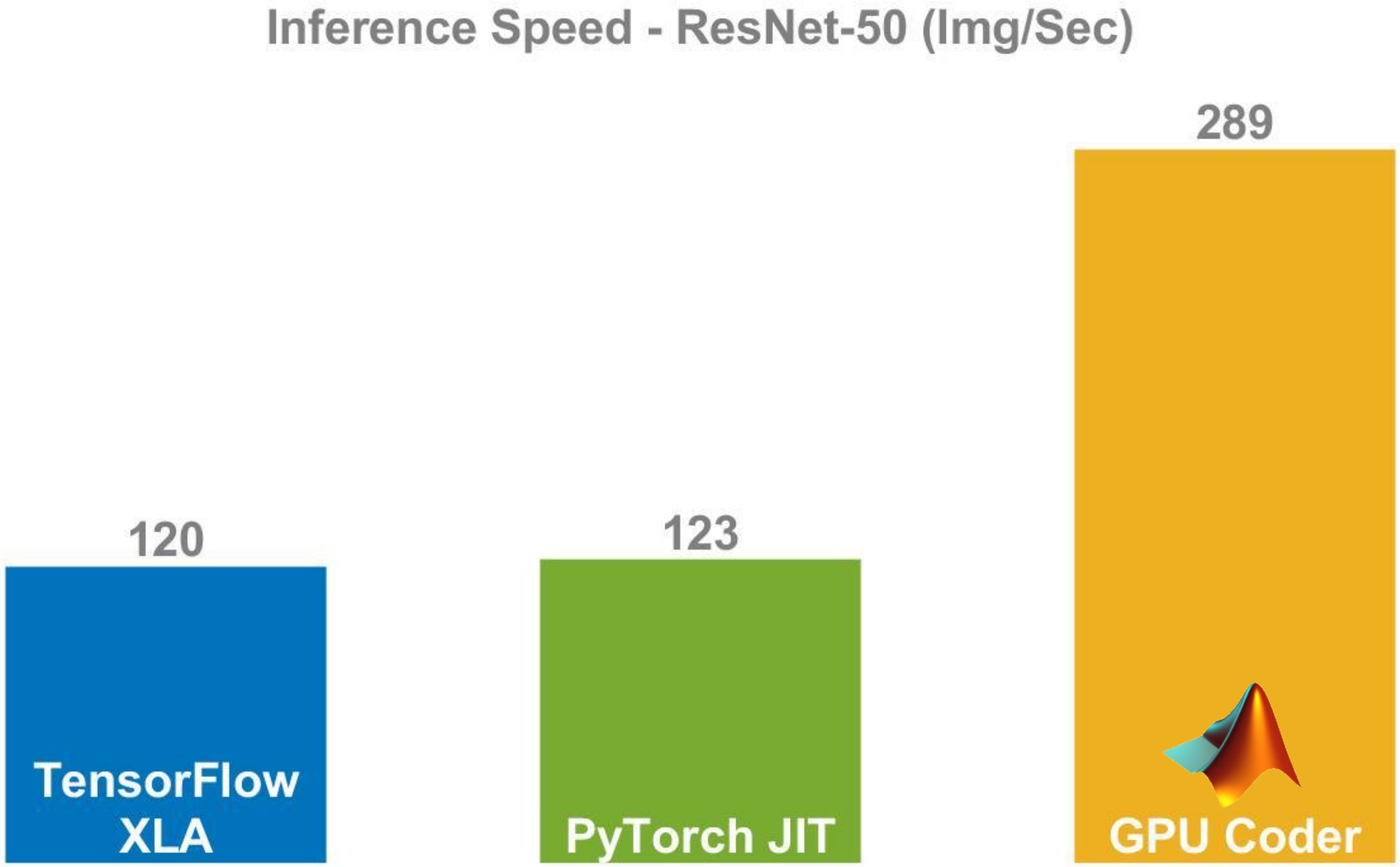
Build directory: /home/buildDri

□ Generate code only

Close Help

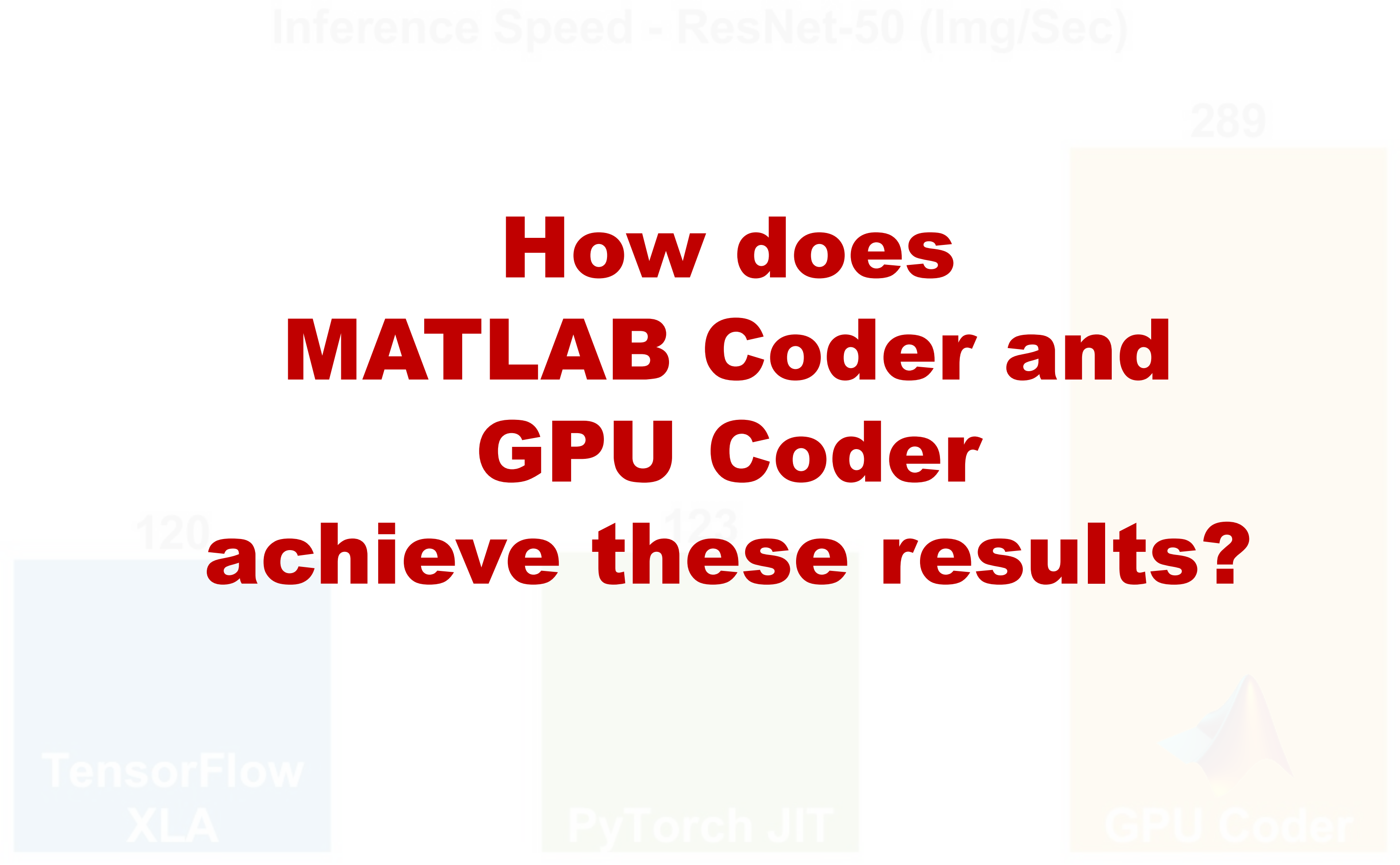


```
%% Deploy and launch through  NVIDIA HSP

%% setup hardware object
% create jetson/drive hardware object with IP or hostname of jeston/drive
%also pass credentials for login
hwObj = jetson('gpucoder-tx2-2','ubuntu','ubuntu');
hwObj.setupCodegenContext;

%% setup codegen config object
% create congen config and connect to hardware object.
cfg_hsp = coder.gpuConfig('exe');
cfg_hsp.Hardware = coder.hardware(hwObj.BoardPref);
buildDir ='~/buildDir';
cfg_hsp.Hardware.BuildDir = buildDir;

%% add user written main files for building executable
% and generate/build the code.
cfg_hsp.CustomSource = 'driver_files_alexnet/main.cu';
cfg_hsp.CustomInclude = 'driver_files_alexnet/';

codegen -config cfg_hsp -args {im, coder.Constant(cnnMatFile)} alexnet_test

%% copy input and run the executable
hwObj.putFile('input2.txt', buildDir);
hwObj.putFile('synsetWords.txt', buildDir);

%execute on Jetson
hwObj.runExecutable([buildDir '/alexnet_test.elf'], 'input2.txt')

%% copy the output file back to host machine
hwObj.getFile([buildDir '/tOut.txt']);
```

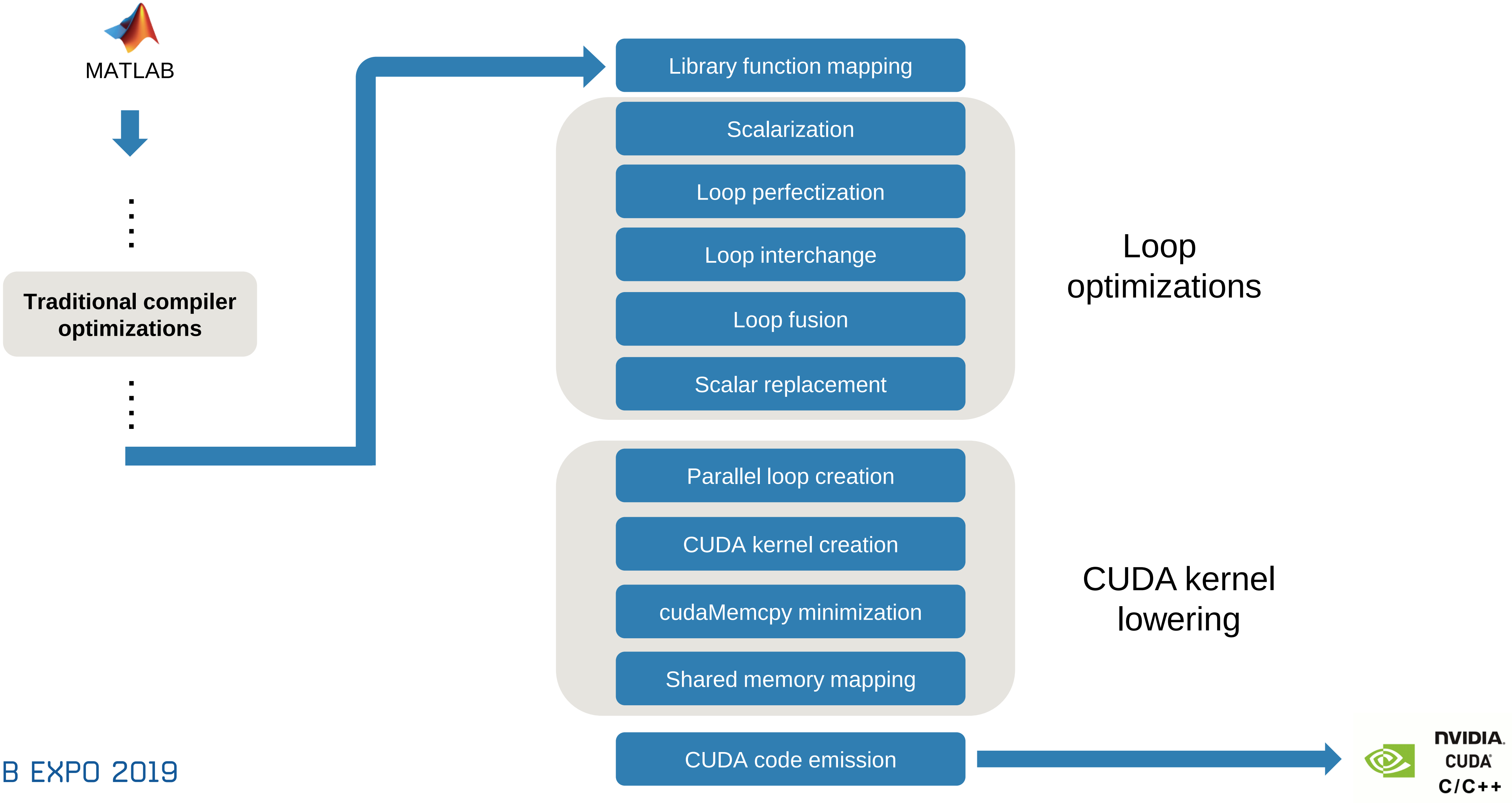

Inference Speed - ResNet-50 (Img/Sec)
120
123
289
TensorFlow XLA
PyTorch JIT
GPU Coder

R2019b

How does MATLAB Coder and GPU Coder achieve these results?



Coders Apply Various Optimizations



MATLAB

Traditional compiler optimizations

Library function mapping

Scalarization

Loop perfectization

Loop interchange

Loop fusion

Scalar replacement

Loop optimizations

Parallel loop creation

CUDA kernel creation

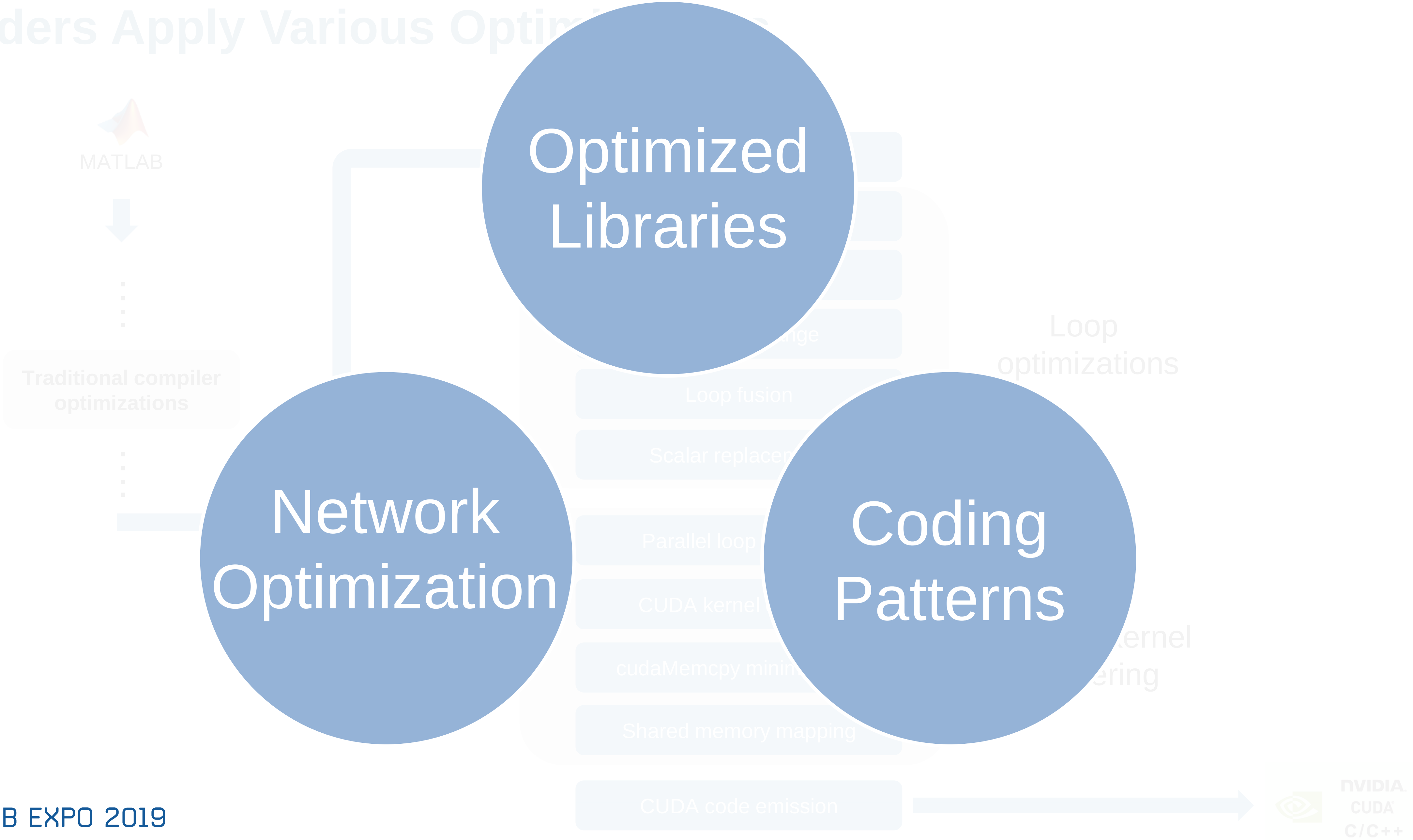
cudaMemcpy minimization

Shared memory mapping

CUDA kernel lowering

CUDA code emission

Coders Apply Various Optim



Optimized Libraries

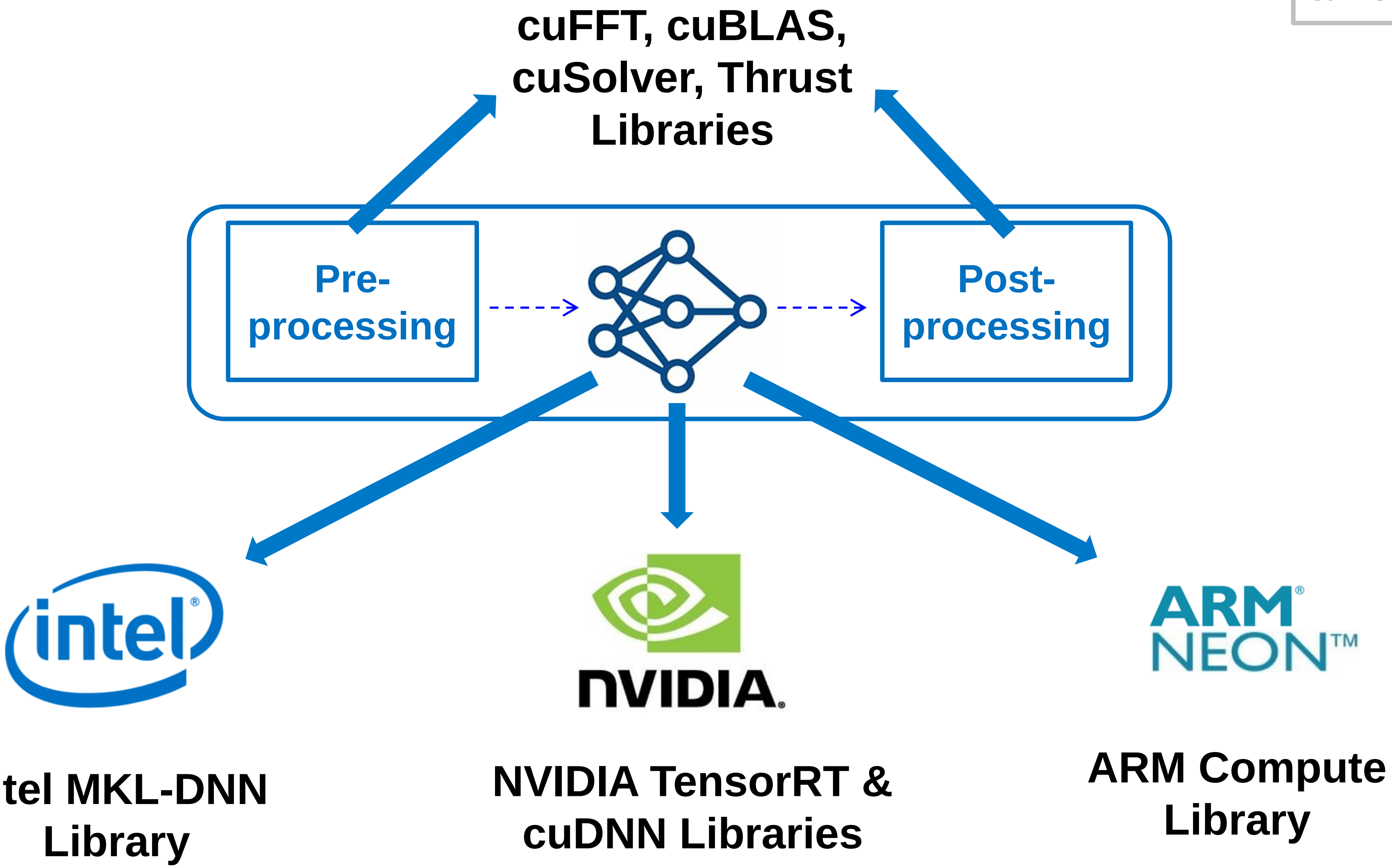
Network Optimization

Coding Patterns

Generated Code Calls Optimized Libraries

Performance

1. **Optimized Libraries**
2. Network Optimizations
3. Coding Patterns



cuFFT, cuBLAS, cuSolver, Thrust Libraries

Pre-processing

Post-processing

Intel MKL-DNN Library

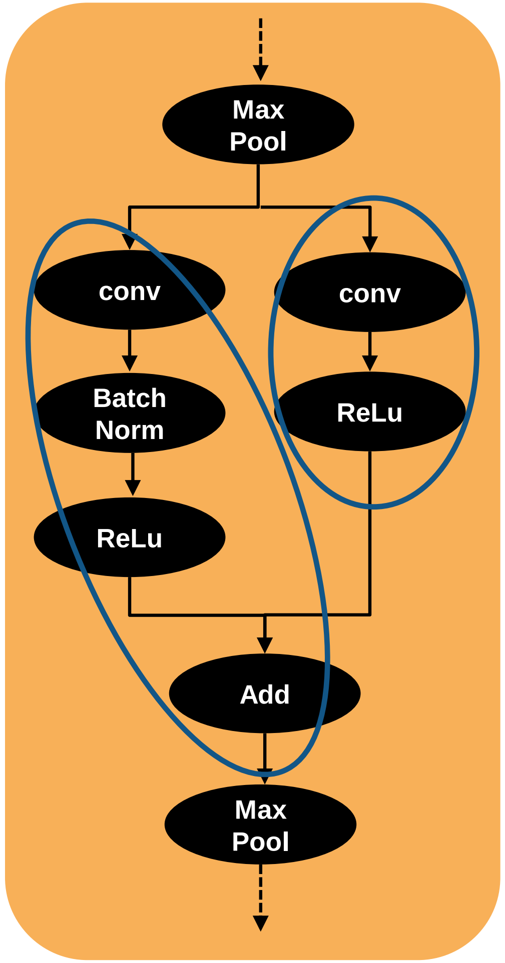
NVIDIA TensorRT & cuDNN Libraries

ARM Compute Library

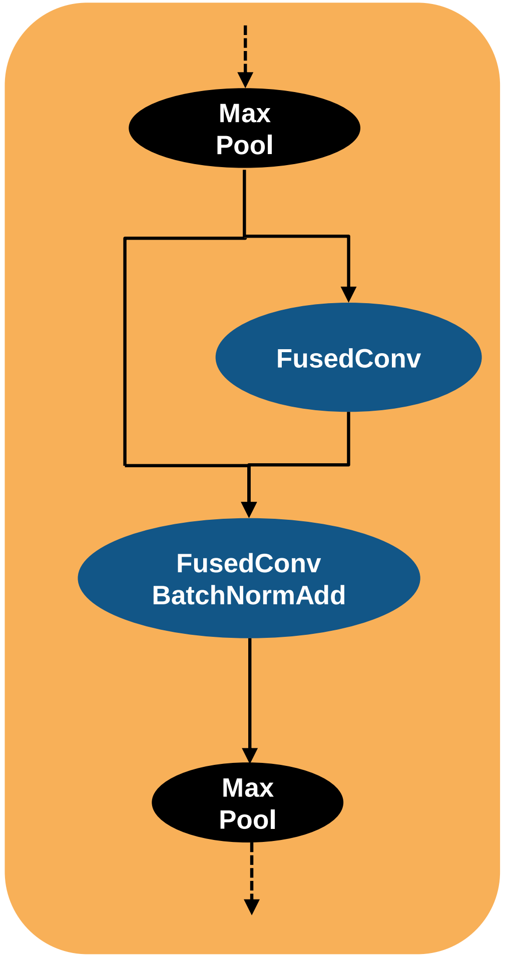
Deep Learning Network Optimization

Performance

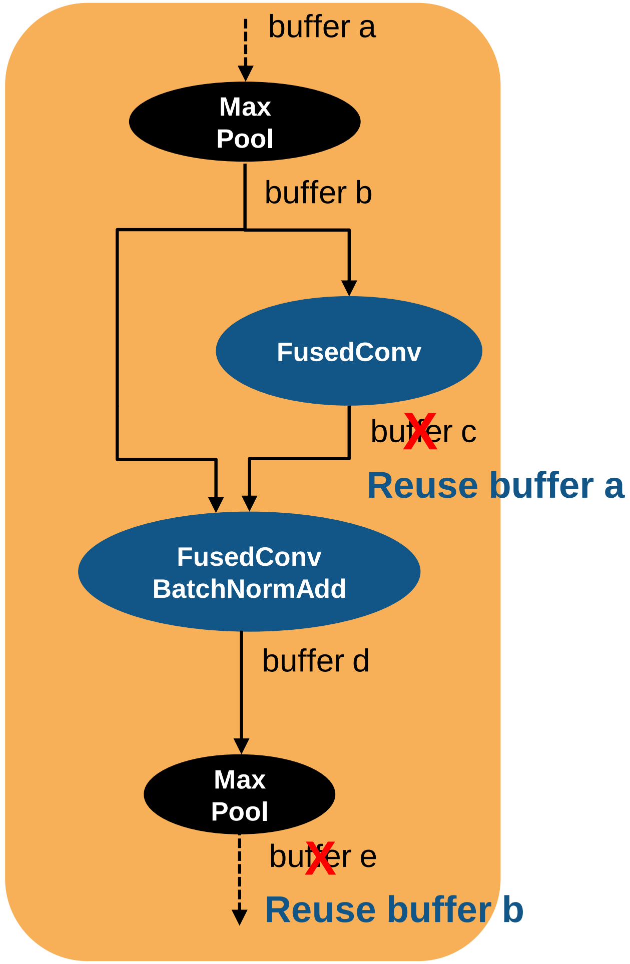
1. Optimized Libraries
2. **Network Optimizations**
3. Coding Patterns



Network



Layer fusion
Optimized computation

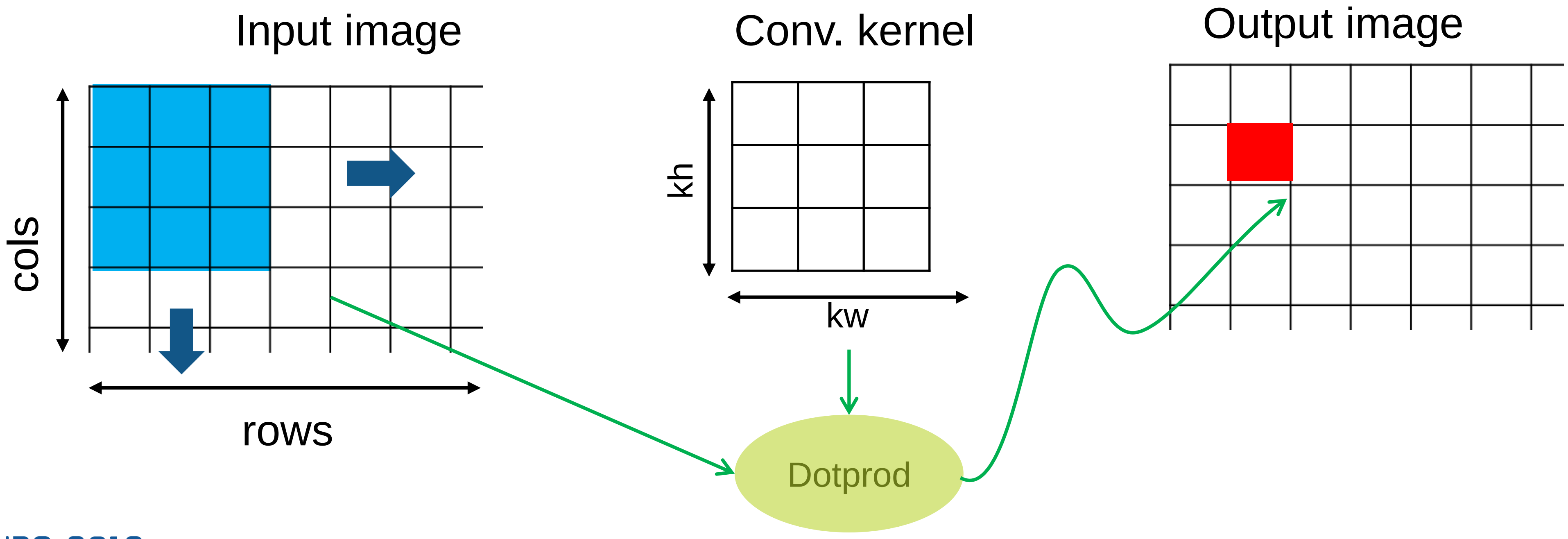


Buffer minimization
Optimized memory

Performance
1. Optimized Libraries
2. Network Optimizations
3. **Coding Patterns**

Coding Patterns: Stencil Kernels

- Automatically applied for image processing functions (e.g. imfilter, imerode, imdilate, conv2, …)
- Manually apply using *gpucoder.stencilKernel()*

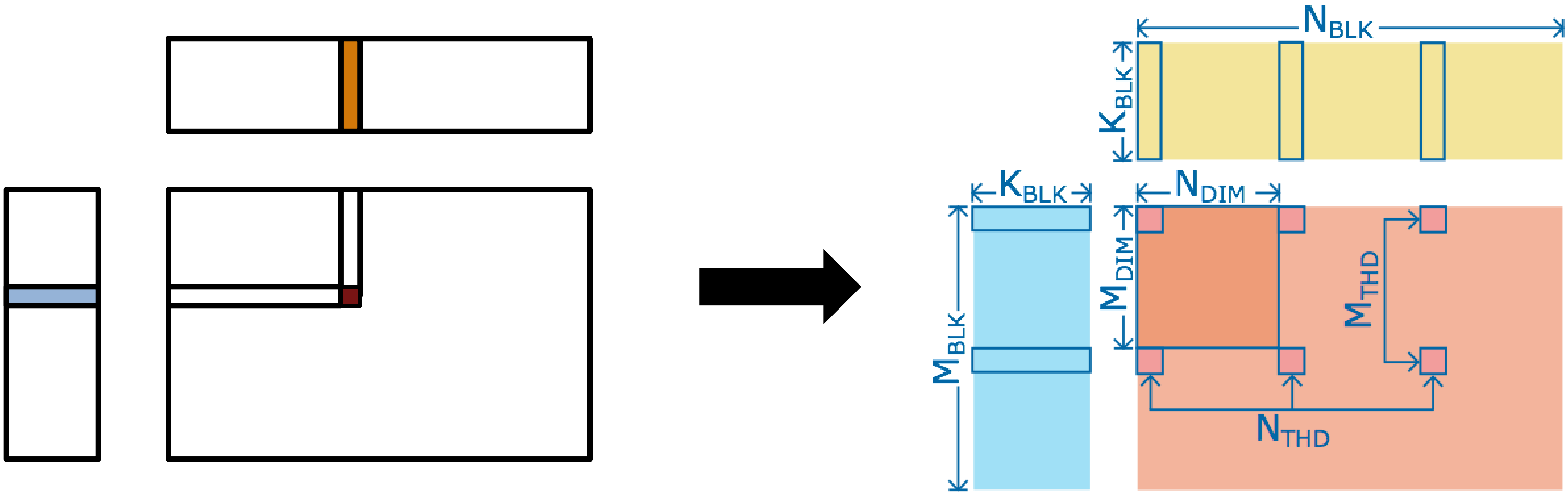


Coding Patterns: Matrix-Matrix Kernels

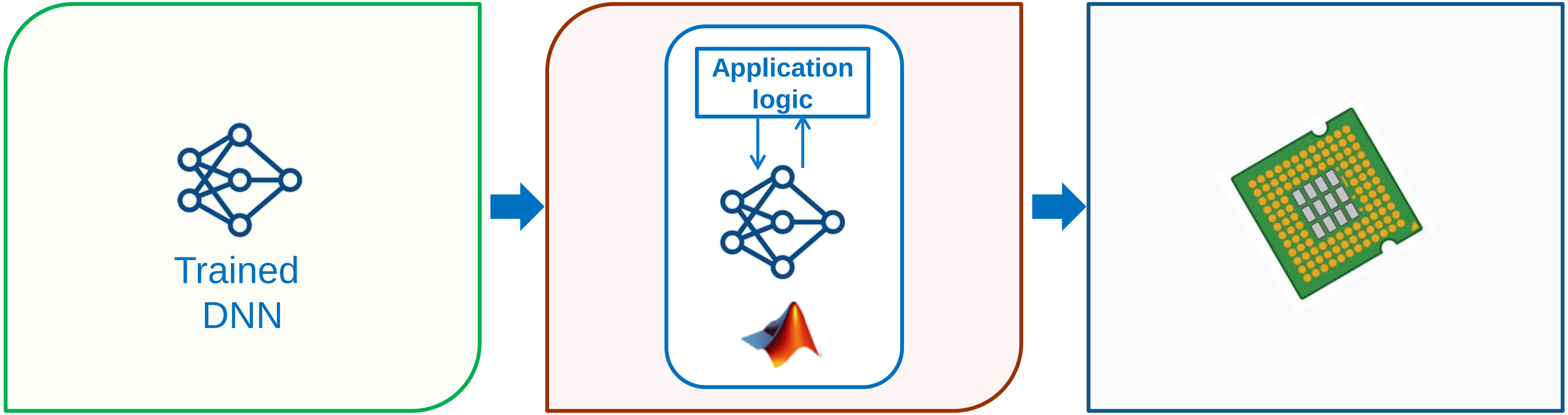
Performance

1. Optimized Libraries
2. Network Optimizations
3. **Coding Patterns**

- Automatically applied for many MATLAB functions (e.g. matchFeatures SAD, SSD, pdist, …)
- Manually apply using *gpucoder.matrixMatrixKernel()*



Deep Learning Workflow in MATLAB



Trained DNN

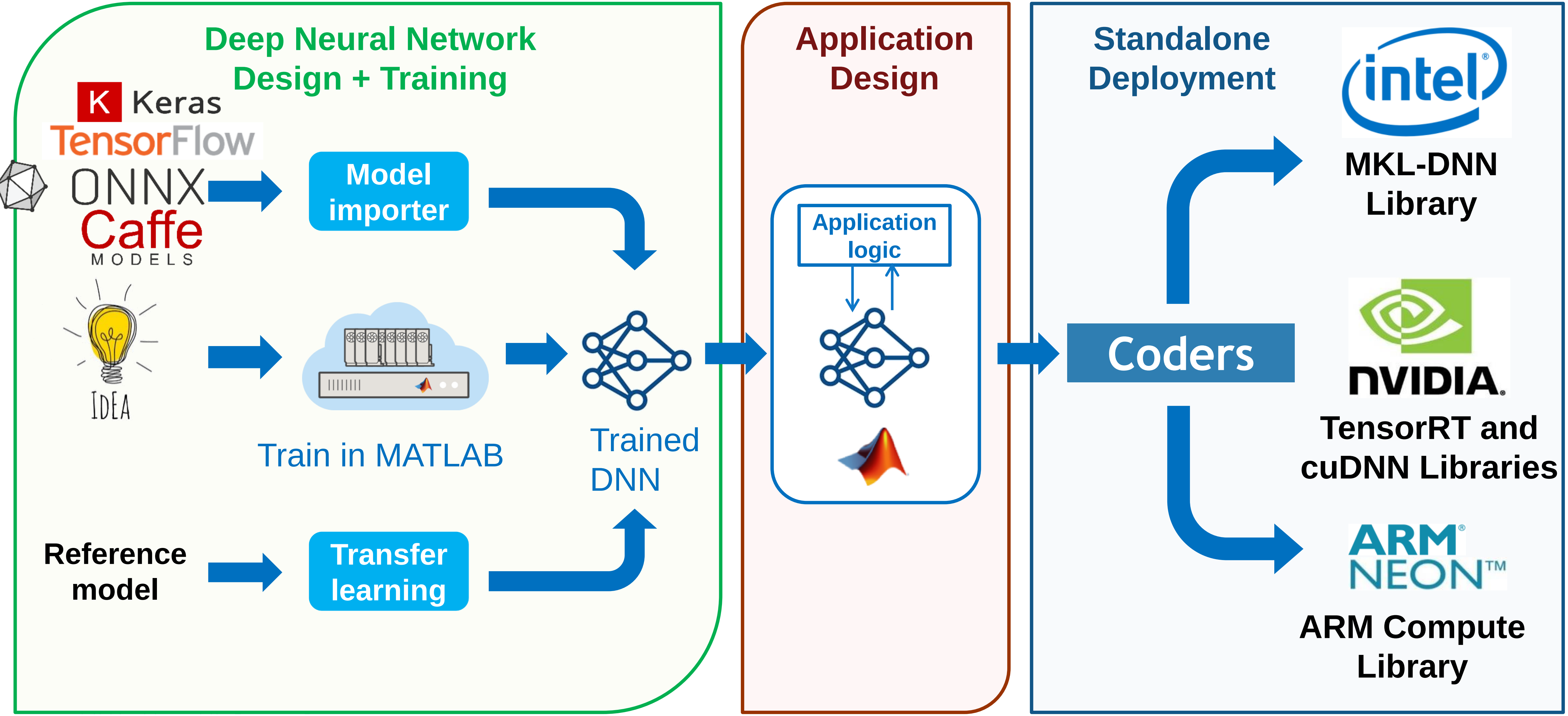
Application logic

Deep Neural Network Design + Training

Application Design

Standalone Deployment

Deep Learning Workflow in MATLAB



Call to action

- Visit the Deep Learning Booth!
- Related upcoming talks:
  - AI Techniques for Signals, Time-series, and Text Data
  - Sensor Fusion and Tracking for Autonomous Systems
  - Deploying Deep Neural Networks to Embedded GPUs and CPUs