Machine Learning with the i.MX RT1060

Markus Levy
Director of AI and Machine Learning Technologies

Juan Carlos Pacheco
Systems Engineer

June 2019 | Session #AMF-MBL-T3645

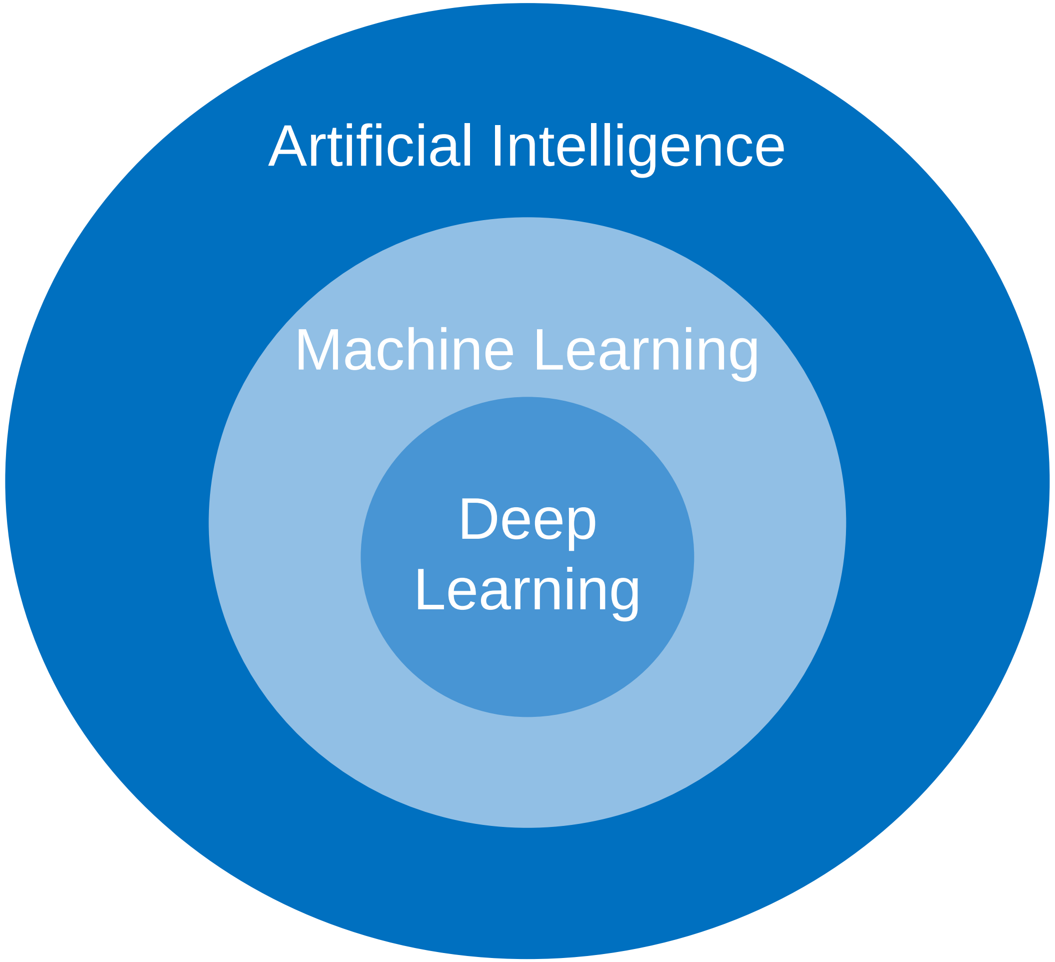
SECURE CONNECTIONS FOR A SMARTER WORLD

Agenda

- Artificial Intelligence/Machine Learning
- eIQ
- eIQ on i.MXRT
- Hands-On
- Q&A and Wrap-up

Artificial Intelligence and Machine Learning

Artificial Intelligence, Machine Learning, and Deep Learning



Artificial Intelligence

- The very broad concept of using machines to do "smart" things and act intelligently like a human

Machine Learning

- One of many ways to implement AI
- The concept that if you give machines a lot of data, they can learn how to do smart things on their own without having to be explicitly programmed to do that action.
- Self learning and self improving

Deep Learning

- One of many ways to implement machine learning
- Uses Neural Networks to do the learning that ML requires
- Automatically determines most relevant data aspects to analyze instead of having to be explicitly told
- Needs lots and lots of data

Embedded Machine Learning Applications

- Image/Object Recognition
- Voice Recognition
- Anomaly Detection
- Smart Wearables
- Intelligent Factories
- Medical
- Augmented Reality

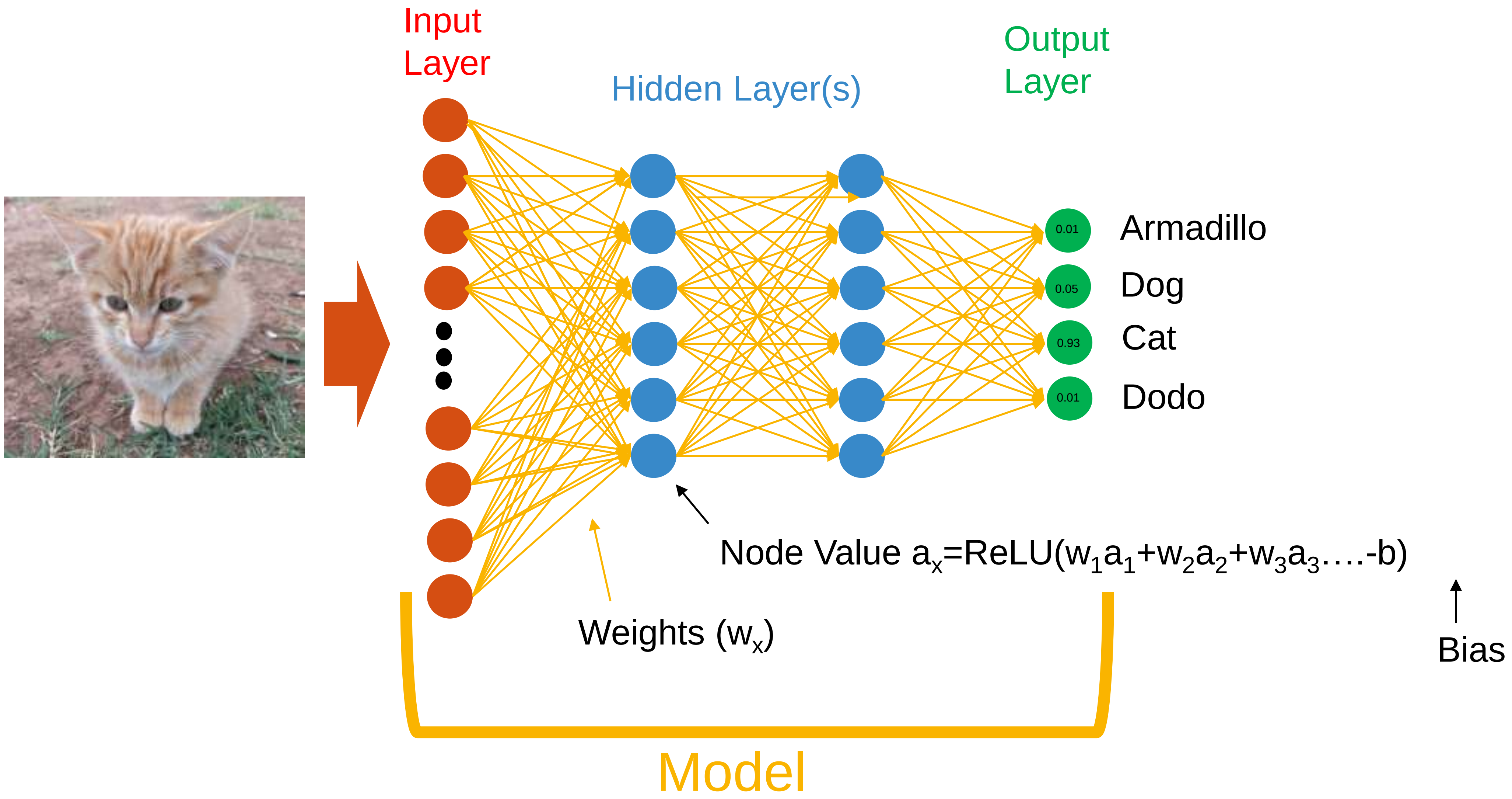
Focused ML Applications for Edge and MCU

- Image based classification
  - Peek hole: Recognize if it is in family set
  - Smart appliances: Recognizes food, inventory monitor
  - Education and hobbyist: OpenMV
- Voice based keyword spotting
  - Always-on voice triggering
  - Audio quality improvement
- Motor control and motion control:
  - Improves accuracy for multiple algorithms.
- Anomaly detection based on time-series: Condition monitoring
  - Monitoring of motor driven systems: lift, pump, wheels, etc for damage and malfunctions.
  - Fall down detect
  - ECG monitoring: Early warning of heart decease risk

Machine Learning Models

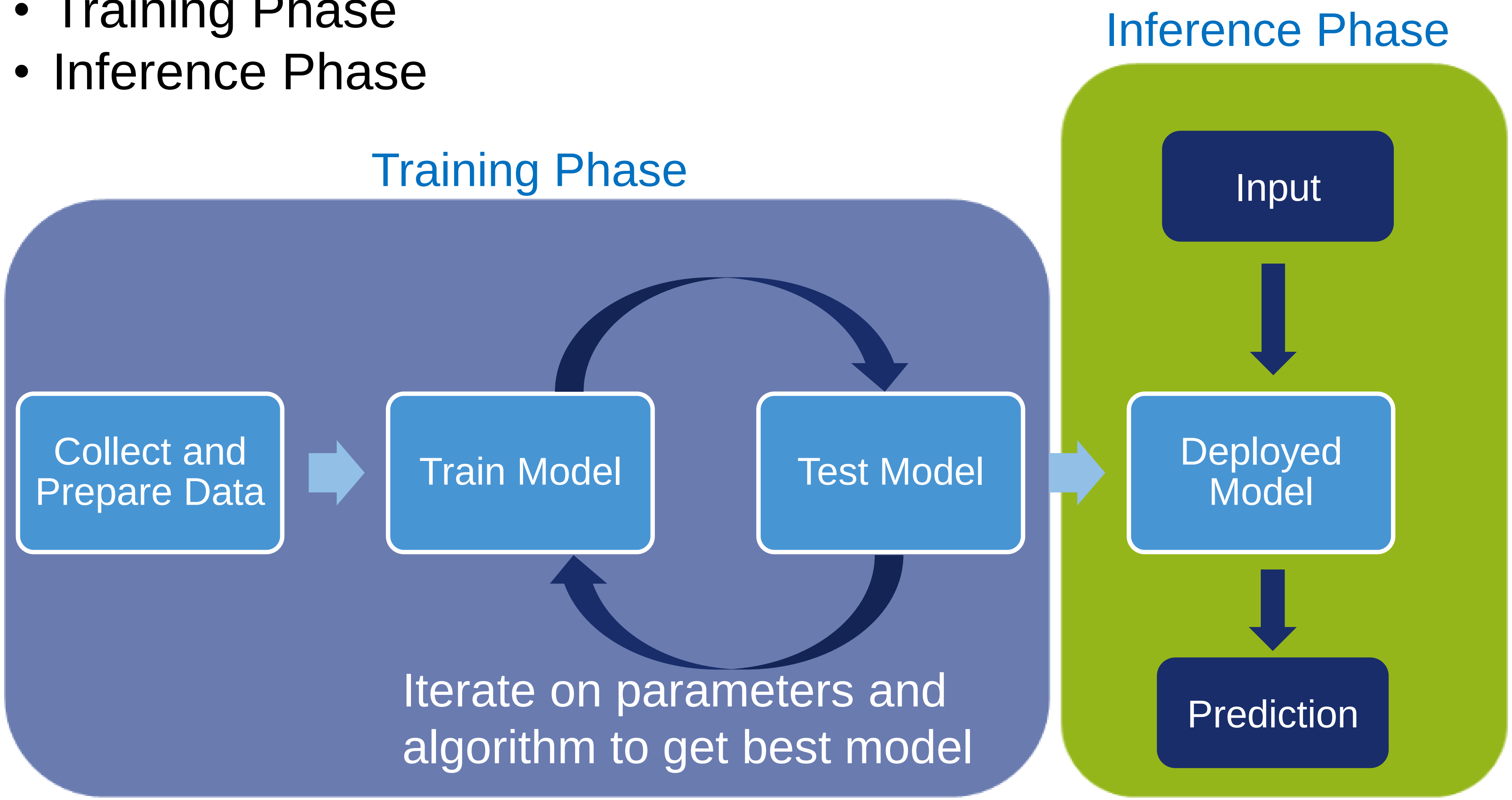
- Models are a mathematical representation of a real-world process
  - ie image recognition, speech recognition, etc
- Essentially a model is a extremely complicated math function that gives a “smart” output value for a given input

Very Simplified Neural Network Model



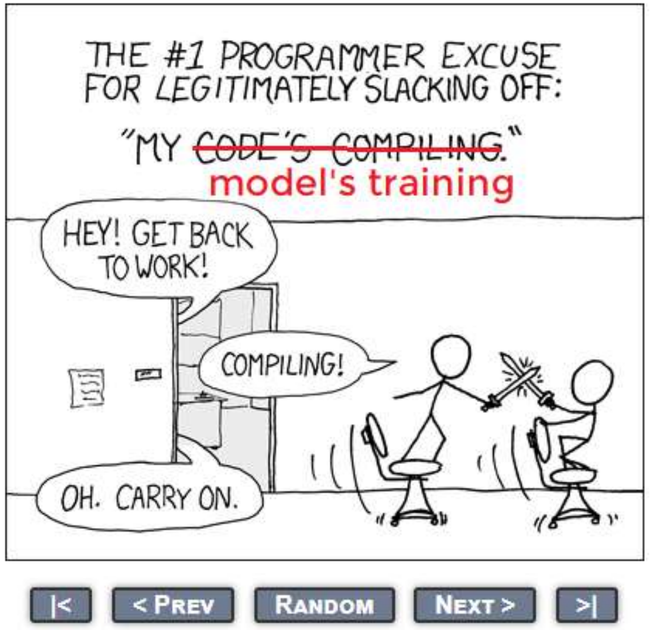
Machine Learning Process

- Training Phase
- Inference Phase



Training Phase

- Training a model is very compute and time intensive
- Involves trying many different weights and biases until get acceptable results over the entire training data
- Difficult to determine what the optimal values are
- Training usually done on CPUs, GPUs, or on cloud
- Due to randomness in the values that are tried, this can result in slightly different weights and biases even if training data is the same
- Could use data taken from the “edge” and upload into cloud for training



PERMANENT LINK TO THIS COMIC: HTTPS://XKCD.COM/303/

Inference Phase

- Inference is using a model to perform evaluations on new data
- Inference time depends on framework and model
- Two possibilities (using image detect as an example):

1) Upload an image to cloud and evaluate on cloud platform
   - Requires network bandwidth.
   - Latency issues
   - Cloud compute costs
2) Evaluate image on embedded system itself: Edge Computing
   - Faster response time and throughput
   - Lower Power
   - Don’t need internet connectivity
   - Increased privacy and security

How Are Models Designed? – Model Frameworks

- A framework provides proven APIs and utilities to design, analyze, train, test, validate and deploy models.
- Each framework has their own APIs and methodologies
- Allows developers to focus on overall logic of model, instead of the details of how to implement algorithms or link layers together

```python
with tf.variable_scope('conv1') as scope:
  kernel = _variable_with_weight_decay('weights',
                                       shape=[5, 5, 3, 64],
                                       stddev=5e-2,
                                       wd=None)
  conv = tf.nn.conv2d(images, kernel, [1, 1, 1, 1], padding='SAME')
  biases = _variable_on_cpu('biases', [64], tf.constant_initializer(0.0))
  pre_activation = tf.nn.bias_add(conv, biases)
  conv1 = tf.nn.relu(pre_activation, name=scope.name)
  _activation_summary(conv1)

# pool1
pool1 = tf.nn.max_pool(conv1, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1],
                       padding='SAME', name='pool1')
# norm1
norm1 = tf.nn.lrn(pool1, 4, bias=1.0, alpha=0.001 / 9.0, beta=0.75,
                  name='norm1')

# conv2
with tf.variable_scope('conv2') as scope:
  kernel = _variable_with_weight_decay('weights',
                                       shape=[5, 5, 64, 64],
                                       stddev=5e-2,
                                       wd=None)
  conv = tf.nn.conv2d(norm1, kernel, [1, 1, 1, 1], padding='SAME')
  biases = _variable_on_cpu('biases', [64], tf.constant_initializer(0.1))
  pre_activation = tf.nn.bias_add(conv, biases)
  conv2 = tf.nn.relu(pre_activation, name=scope.name)
  _activation_summary(conv2)
```

CIFAR-10 Model in TensorFlow Framework

Model Frameworks

- There are several popular model frameworks in use today.
- This is a constantly changing list as new software is released:
  - TensorFlow – Google framework
  - Keras – higher level API, usually built on top of TensorFlow
  - Caffe2 – Facebook framework
  - PyTorch – Facebook framework

- Python is used for most ML frameworks
  - Interact, build, and train via Python scripts
- New breakthroughs constantly and favorite framework du jour can change quickly

Machine Learning Accuracy

Model Accuracy Continued

Models are not perfect. Especially when scaling down models to fit on embedded systems

Model	Million MACs	Million Parameters	Top-1 Accuracy	Top-5 Accuracy
MobileNet_v1_1.0_224	569	4.24	70.9	89.9
MobileNet_v1_1.0_192	418	4.24	70.0	89.2
MobileNet_v1_1.0_160	291	4.24	68.0	87.7
MobileNet_v1_1.0_128	186	4.24	65.2	85.8
MobileNet_v1_0.75_224	317	2.59	68.4	88.2
MobileNet_v1_0.75_192	233	2.59	67.2	87.3
MobileNet_v1_0.75_160	162	2.59	65.3	86.0
MobileNet_v1_0.75_128	104	2.59	62.1	83.9
MobileNet_v1_0.50_224	150	1.34	63.3	84.9
MobileNet_v1_0.50_192	110	1.34	61.7	83.6
MobileNet_v1_0.50_160	77	1.34	59.1	81.9
MobileNet_v1_0.50_128	49	1.34	56.3	79.4
MobileNet_v1_0.25_224	41	0.47	49.8	74.2
MobileNet_v1_0.25_192	34	0.47	47.7	72.3
MobileNet_v1_0.25_160	21	0.47	45.5	70.3
MobileNet_v1_0.25_128	14	0.47	41.5	66.3

Things That Affect Model Accuracy

- Quality of input training data
- Quantity of input training data
- Model Structure and Training Method
- Efficiency of model conversion for running on embedded system
  - Quantization and Pruning
- Quality of input test data

Quantization and Pruning

- Quantization is transforming 32-bit floating point weights into 8-bit fixed point weights
  - Reduces model size by 4x
  - Fixed point math much quicker than floating point
  - Usually results in little loss of accuracy
  - Uses min/max of floating point values and maps them to a 0-255 value
- Pruning is removing unused or low importance weights and biases from a neural network
  - Recommended to retrain model after pruning

Enablement for Machine Learning

NXP Broad-based Machine Learning Solutions and Support (Available Today!)

eIQ™ ML Enablement

- eIQ (edge intelligence) for general-purpose edge AI/ML inference enablement
- i.MX 8 family (GA w/ 4.19 release), i.MX RT1050/1060 (GA w/ 2.6 release)

DIY

Third Party SW and HW

- Coral Dev Board
- i.MX 8M Development Kit for Amazon® Alexa Voice Service w/ DSP Concepts
- Au-Zone Development Tools

Short List Here

EdgeScale™ Solution

- Secure deployment of applications (incl. AI/ML) through docker containers
- Layerscape devices now; adding i.MX

Think Docker

Turnkey Solutions

- AVS Solution (Alexa Voice Services) - i.MX RT106A (part# SLN-ALEXA-IOT) Link
- Coming soon for broad market - Anomaly detection and facial recognition solutions based on i.MX RT, i.MX 8M Mini

Fully Tested

eIQ

Edge Intelligence

eIQ – Collection of Libraries and Development Tools for Building ML Apps Targeting NXP MCUs and App Processors

Deploying open-source inference engines

Integration and optimization of neural net (NN) inference engines (Arm NN, Arm CMSIS-NN, OpenCV, TFLite, ONNX, etc.)

End-to-end examples demonstrating customer use-cases (e.g. camera → inference engine)

Support for emerging neural net compilers (e.g. GLOW)

Suite of classical ML algorithms such as support vector machine (SVM) and random forest

Integrated into Yocto Linux BSP and MCUXpresso SDK

No separate SDK or release to download

- iMX: New layer meta-imx-machinelearning in Yocto
- MCU: Integrated in MCUXpresso SDK middleware

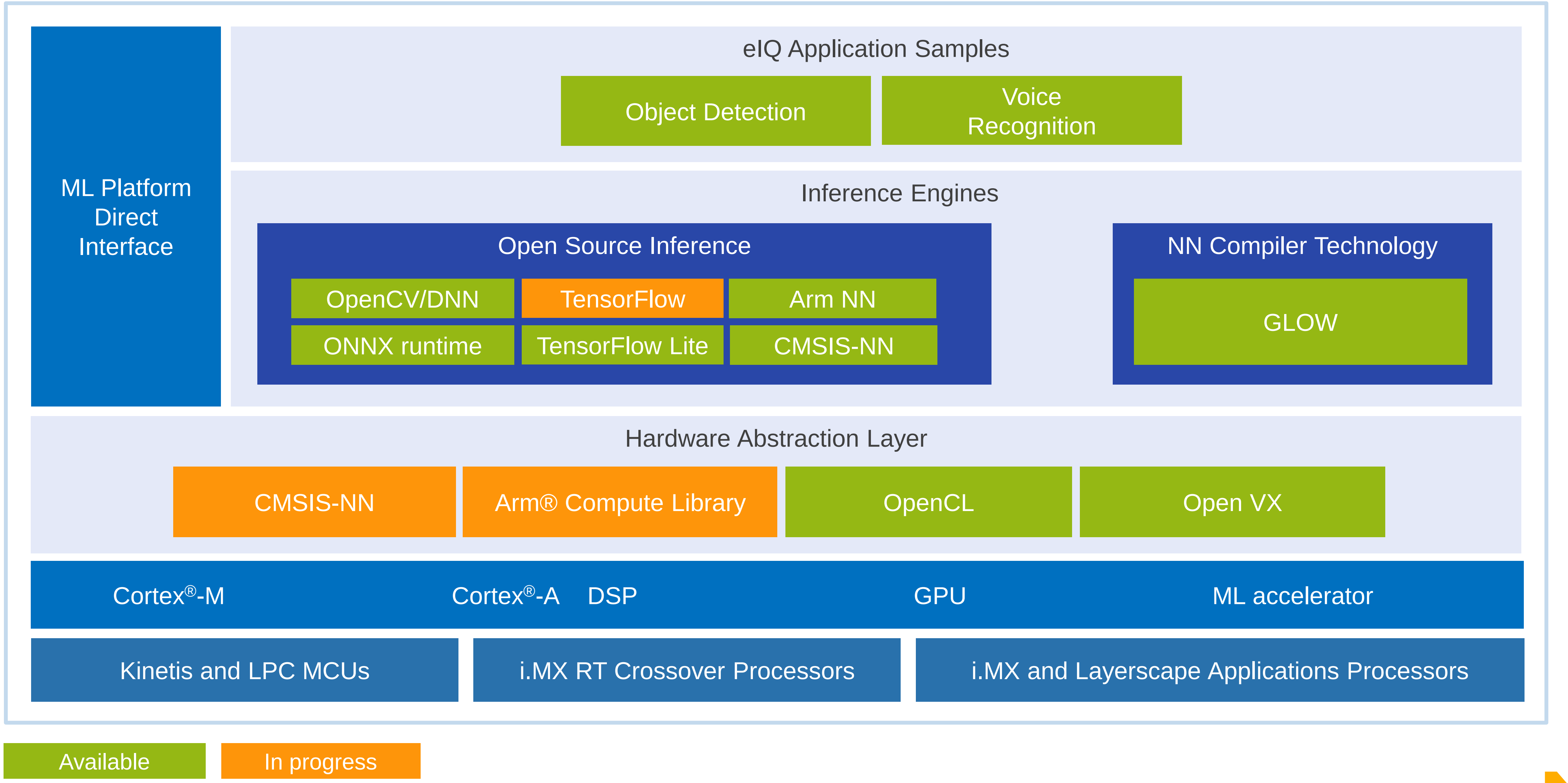
Supporting materials for ease of use

Documentation: eIQ White Paper, Release Notes, eIQ User's Guide, Demo User's Guide

Guidelines for importing pretrained models based on popular NN frameworks (e.g. TensorFlow, Caffe)

Training collateral for CAS, DFAEs and customers (e.g. lectures, hands-on, video)

eIQ-Core Machine Learning Software Development Environment



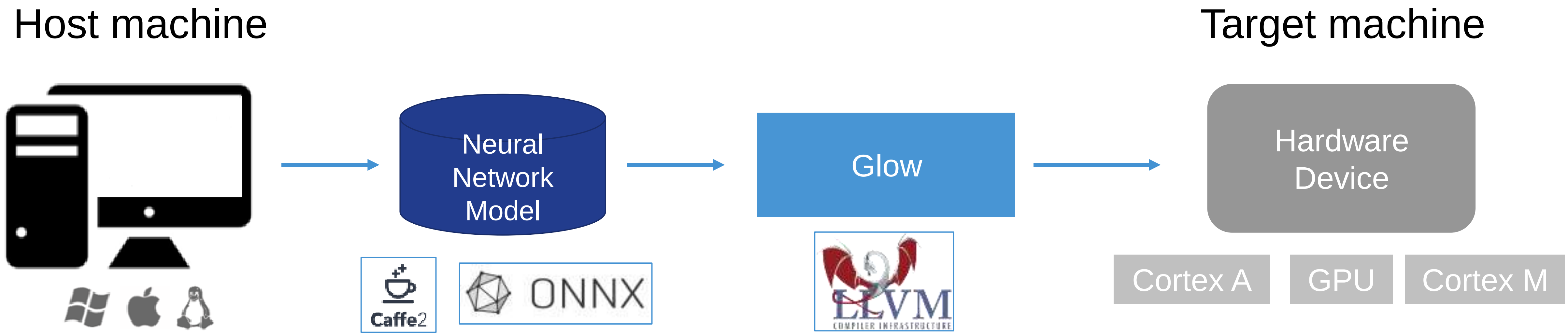
NXP eIQ –Inference Engines & Libraries

eIQ Deployment NN Models

Embedded Compute Engines

	Cortex-M		DSP	Cortex-A				GPU	
	CMSIS-NN / TensorFlow Lite	GLOW	TensorFlow Lite	Arm NN / Arm Compute Library	ONNX Runtime	TensorFlow Lite	OpenCV / OpenCL	Arm NN	Arm Compute Library
i.MX 8QM	*	*		NOW	May ’19	May ’19	NOW	July ’19	July ’19
i.MX 8QXP	*	*		NOW	*	*	NOW	July ’19	July ’19
i.MX 8M Quad	*	*		NOW	*	*	NOW	July ’19	July ’19
i.MX 8M Mini	*	*		NOW	*	*	NOW		
i.MX 6 and 7	* (only some models)	* (only some models)		*	*	*	*		
LS1, LS2, LX2	---	---		*	*	*	*		
i.MX RT600	TBD	TBD		---	---	---	---		
i.MX RT1050/1060	NOW	May ’19		---	---	---	---		

Glow Overview [NN Compiler]



Model design & training

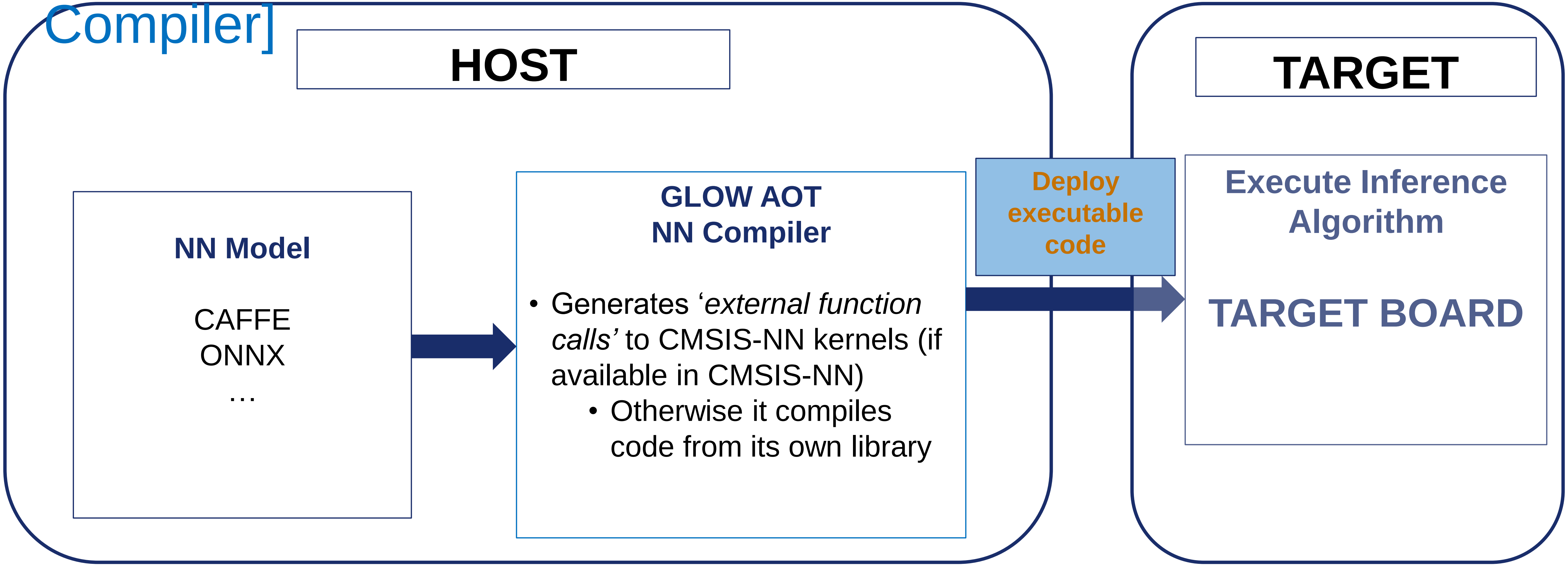
PC or Cloud

Pre-trained model
Standard formats

Model optimization
Model compression
Model compilation

Inference

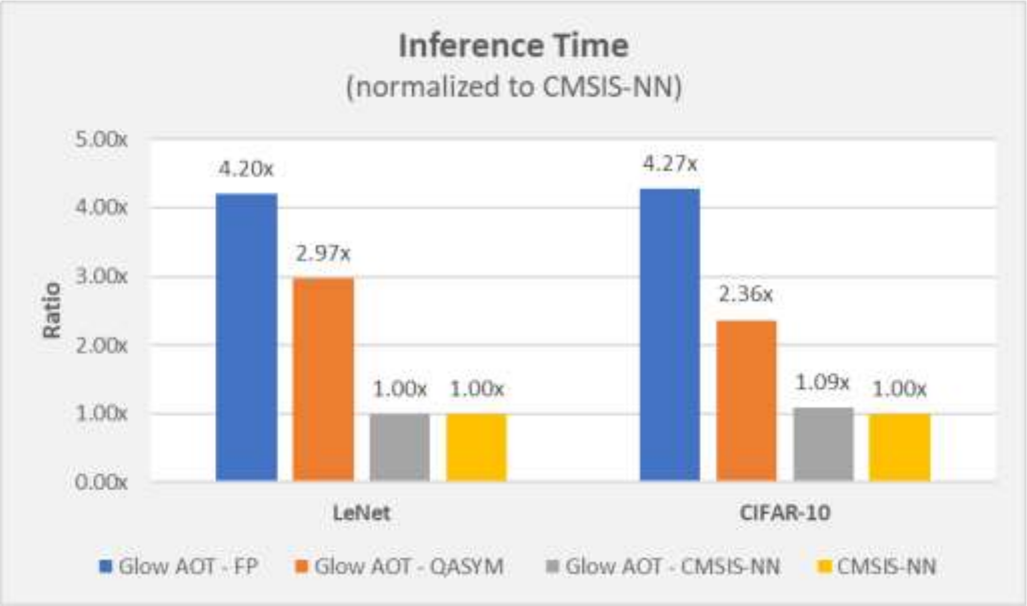
GLOW – AOT [NN Compiler]

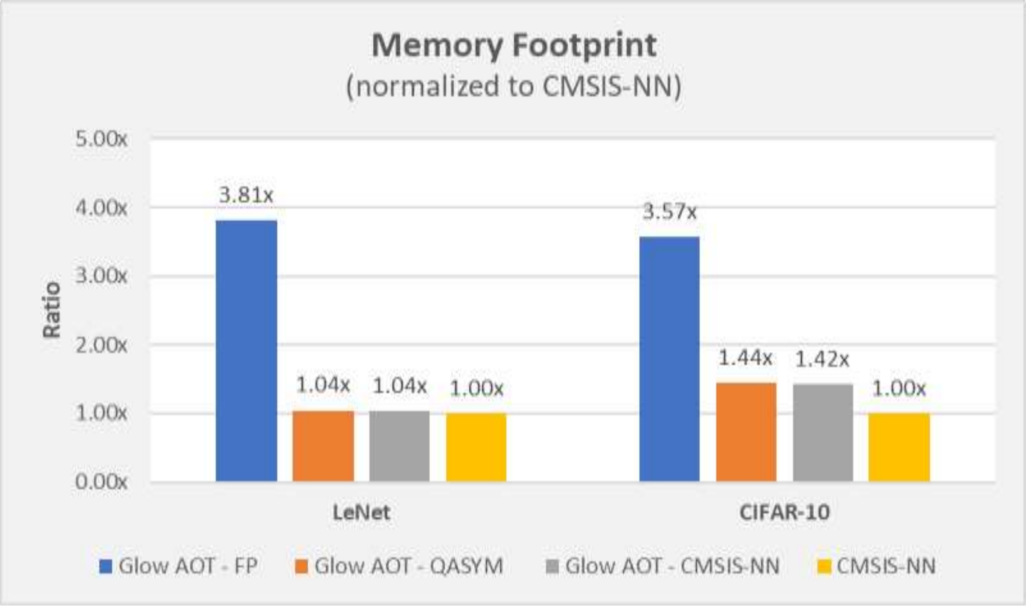


AOT (Ahead Of Time)

GLOW Benefits Revealed

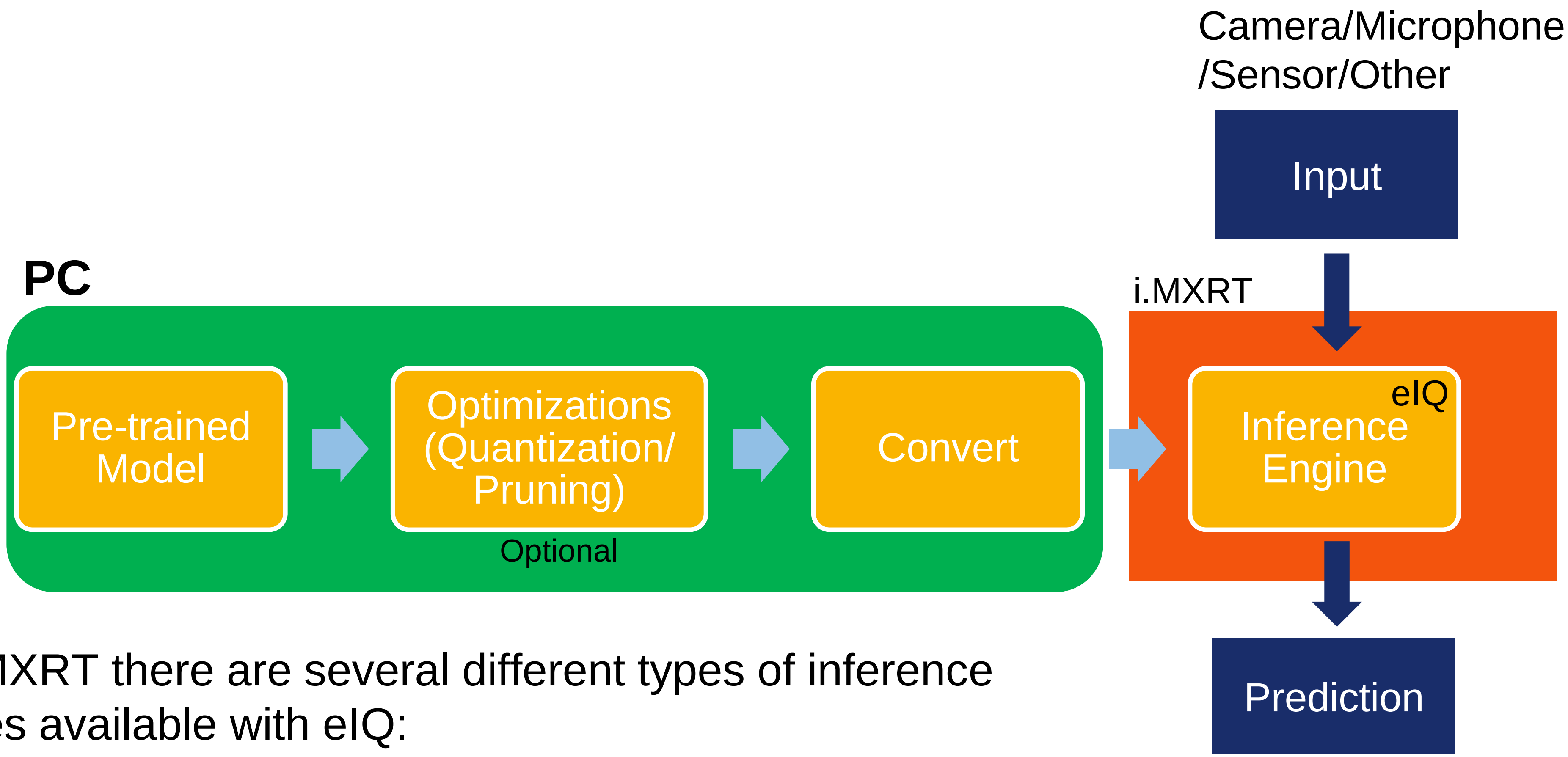
- On par with CMSIS-NN
- But more flexibility





eIQ on i.MXRT

eIQ for iMXRT



For i.MXRT there are several different types of inference engines available with eIQ:

- TensorFlow Lite – Used for TensorFlow model frameworks
- CMSIS-NN – Can be used for several different model frameworks
- Glow – Machine Learning compiler (Coming Soon)

eIQ TensorFlow

TensorFlow Lite Inference Engine

- Developed by Google
- Uses tflite_convert utility (provided by TensorFlow) to convert a TensorFlow model to a .tflite binary
- Load the .tflite binary into embedded system and use TensorFlow Lite inference engine running on i.MXRT to run model

- Only can be used for TensorFlow models
- Tensorflow Lite supports a subset of Tensorflow operators
  - Depending on model, conversion may not be possible or require custom implementation
  - https://www.tensorflow.org/lite/guide/ops_compatibility

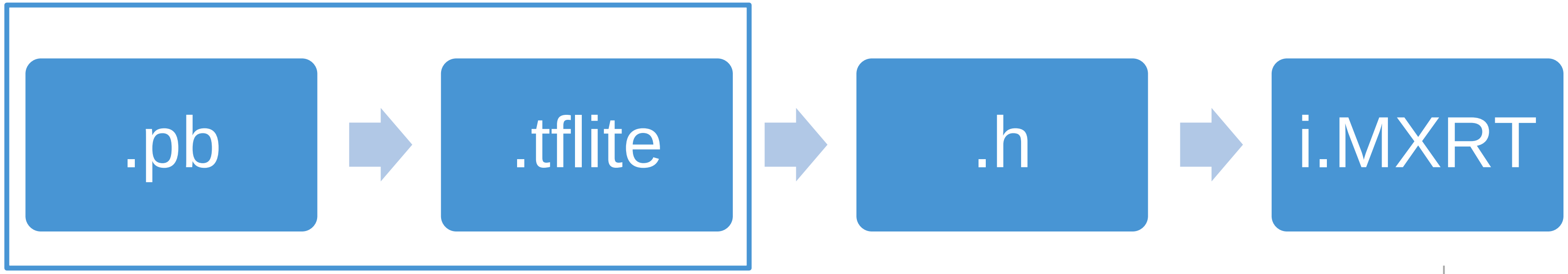
TensorFlow Lite Conversion Process (1 of 3)

Step 1: Convert TensorFlow .pb model file to .tflite file with the **tflite_convert** utility

```
tflite_convert \
    --graph_def_file=retrained_graph.pb \
    --output_file=retrained_graph.tflite \
    --input_shape=1,128,128,3 \
    --input_array=input \
    --output_array=final_result \
    --inference_type=FLOAT \
    --input_data_type=FLOAT
```

This model takes in 128x128 image with 3 color channels (RGB)

Get first and last layer names via tf_get_labels.py or using Netron

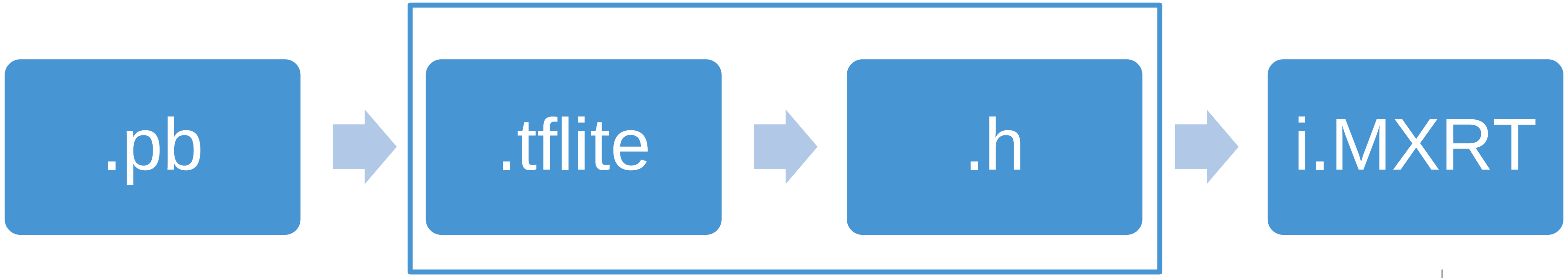


TensorFlow Lite Conversion Process (2 of 3)

Step 2: Convert .tflite file to a binary array with xxd

```
xxd -i retrained_graph.tflite > retrained_graph.h
```

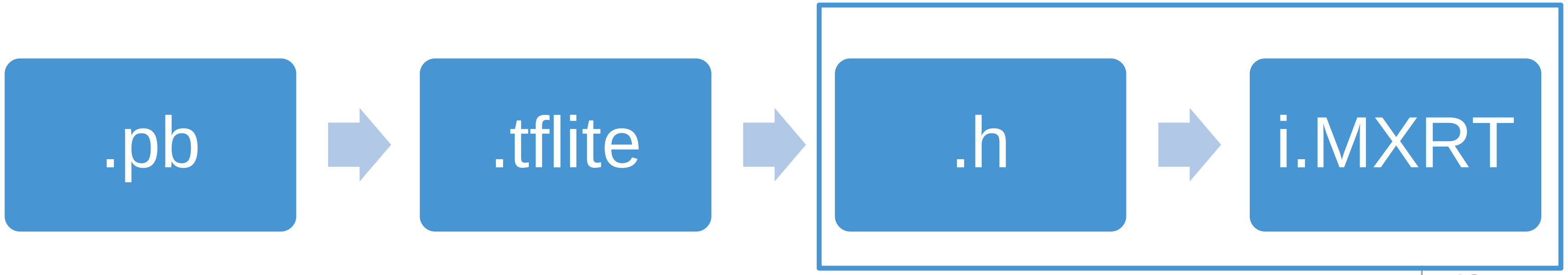
Will also need to change generated array from “unsigned char” to “const char”



TensorFlow Lite Conversion Process (3 of 3)

Step 3: Import array into eIQ project and use TensorFlow Lite API to load model at runtime

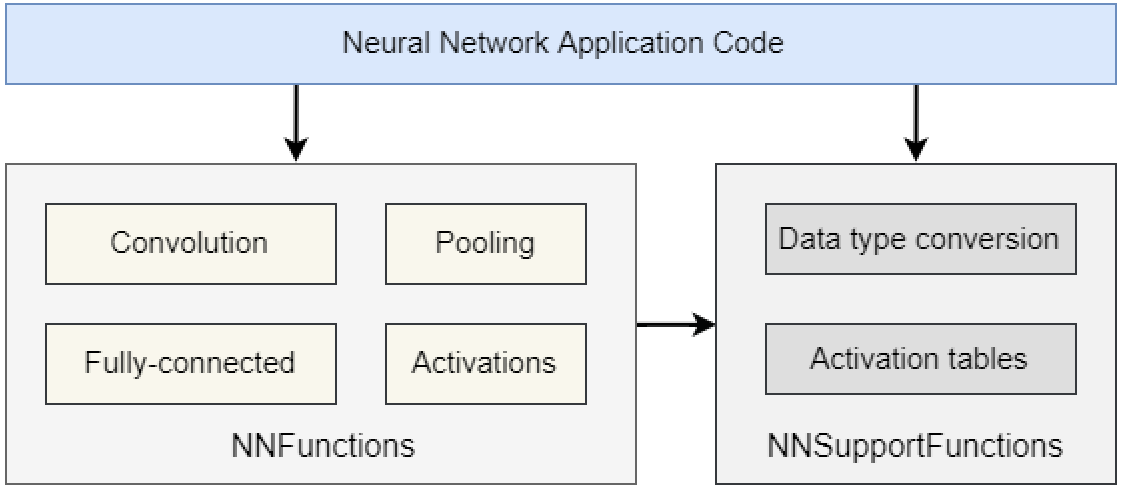
```
#include “retrained_graph.h”

model = tflite::FlatBufferModel::BuildFromBuffer(retrained_graph, retrained_graph_len);
```



eIQ CMSIS-NN

CMSIS-NN Inference Engine

- Developed by ARM
- API to implement common model layers such as convolution, fully-connected, pooling, activation, etc., efficiently at a low level
- Conversion scripts (provided by ARM) to convert Caffe models into CMSIS-NN API calls.
- CMSIS-NN could also be used to optimize the implementation of other inference engines
- Using “Release” high optimization compile settings significantly reduces inference time



CMSIS-NN – Efficient NN Kernels for Cortex-M CPUs

Convolution

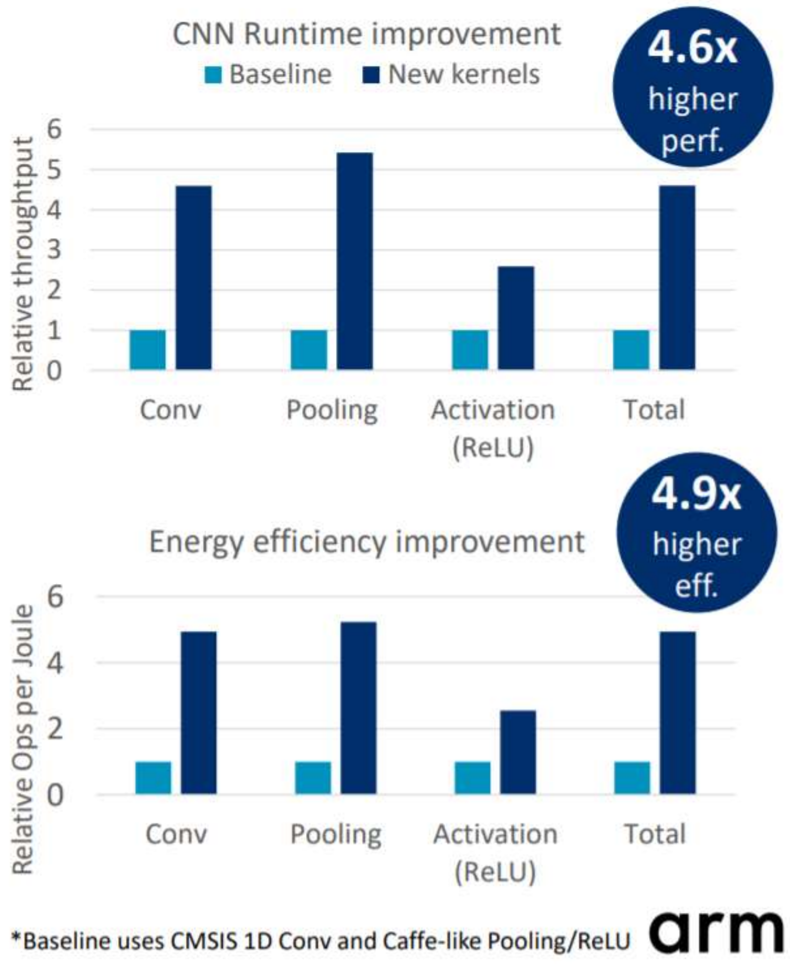
- Boost compute density with GEMM based implementation
- Reduce data movement overhead with depth-first data layout
- Interleave data movement and compute to minimize memory footprint

Pooling

- Improve performance by splitting pooling into x-y directions
- Improve memory access and footprint with in-situ updates

Activation

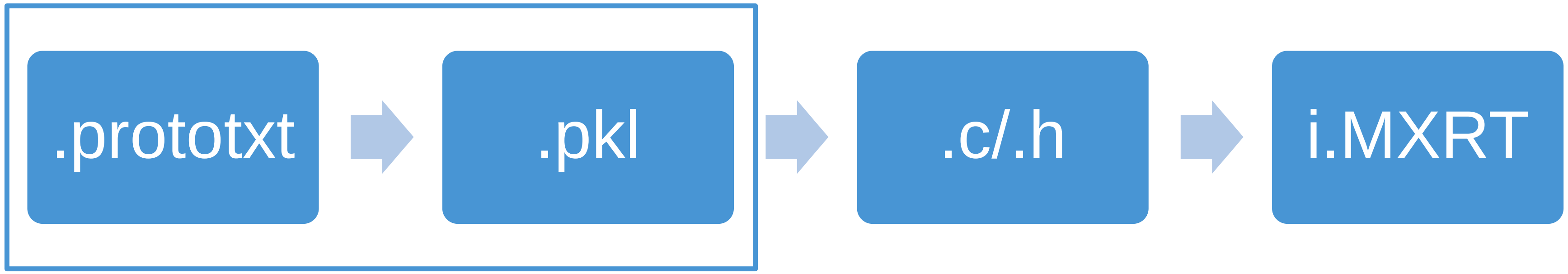
- ReLU: Improve parallelism by branch-free implementation
- Sigmoid/Tanh: fast table-lookup instead of exponent computation



CMSIS-NN Conversion Process for Caffe Model (1 of 3)

Step 1: Quantize a Caffe model with nn_quantizer.py script and put into pickle format:

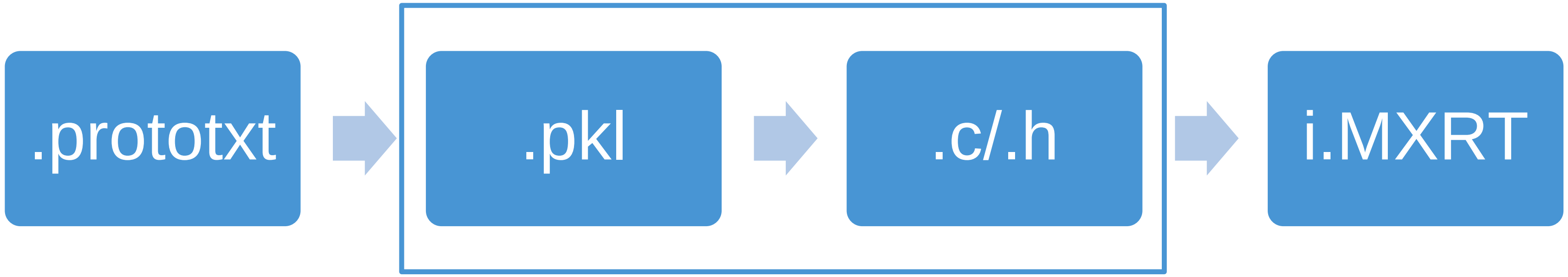
```
python nn_quantizer.py \
    --model cifar10_m7_train_test.prototxt \
    --weights cifar10_m7_iter_300000.caffemodel.h5 \
    --save cifar10_m7.pkl
```



CMSIS-NN Conversion Process for Caffe Model (2 of 3)

Step 2: Convert model to CMSIS-NN code with code_gen.py script:

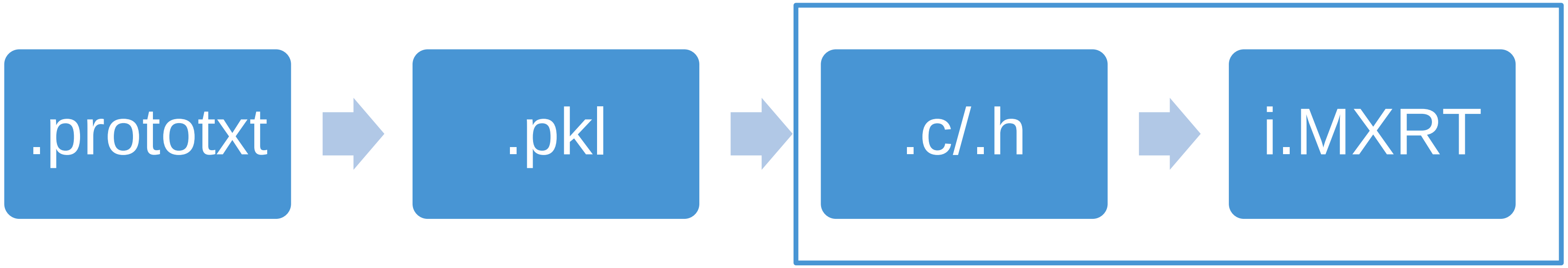
```
python code_gen.py \
    --model cifar10_m7.pkl \
    --out_dir m7_code
```



CMSIS-NN Conversion Process for Caffe Model (3 of 3)

Step 3: Import weights and parameter files into eIQ project, and copy in the generated CMSIS-NN code into project:

```
#include "parameter.h"    //Parameters for model
#include "weights.h"      //Weights for model

arm_convolve_HWC_q7_RGB(img_buffer2, CONV1_IM_DIM,
                        CONV1_IM_CH, conv1_wt, CONV1_OUT_CH …….
```



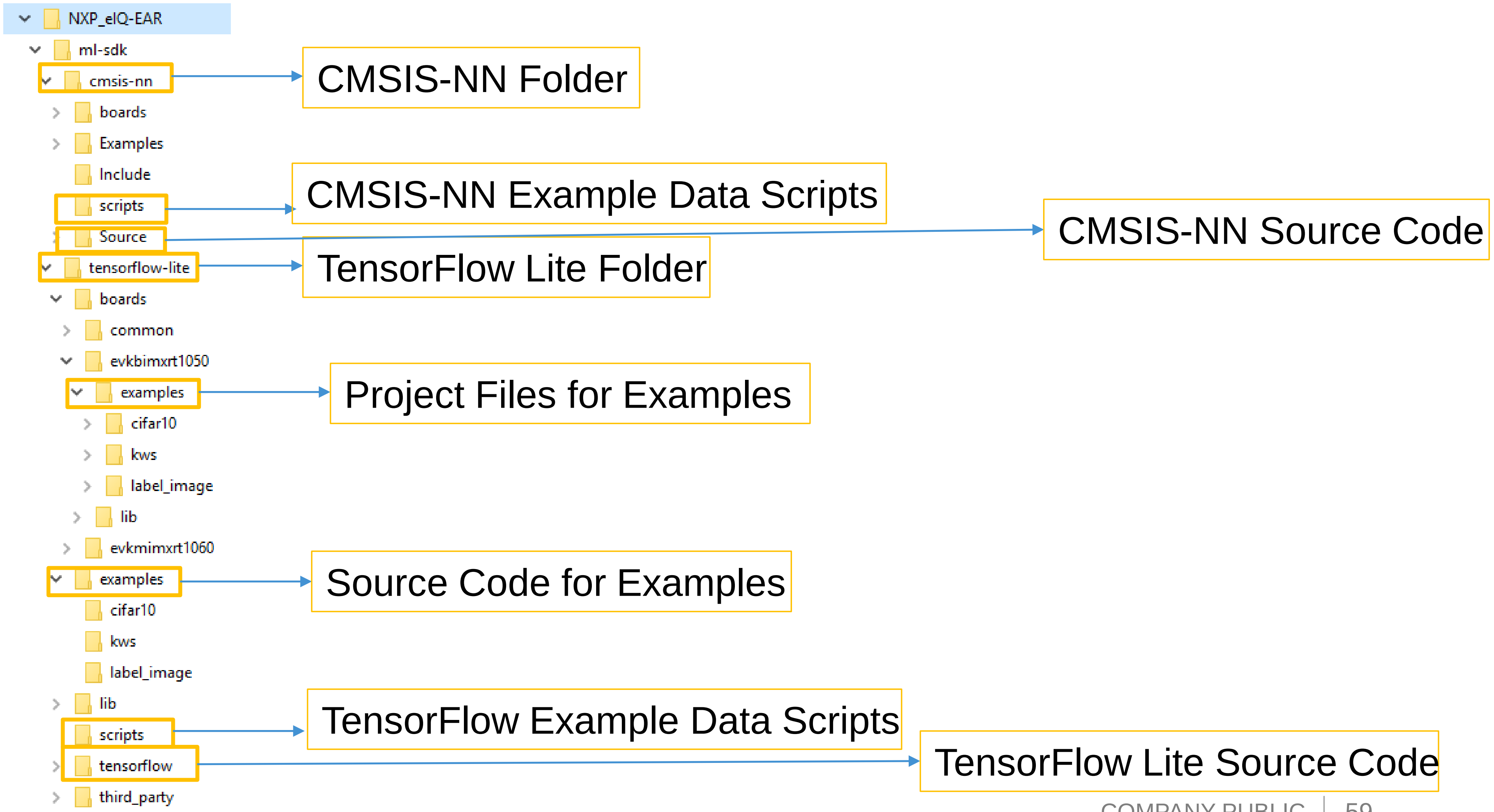
eIQ i.MXRT Examples

eIQ Examples

Three ML application examples available:

	CIFAR-10	Keyword Spotting (KWS)	Label Image
Description	Classifies 32x32 image into one of 10 categories using the CIFAR-10 dataset	Detects specific keywords from pre-recorded audio sample	Classifies an image into one of 1000 categories
TensorFlow Lite Example	✓	✓	✓
CMSIS-NN Example	✓	✓	

eIQ Folder Structure



TensorFlow Lite “Label Image” Example Walkthrough

- Written in C++. Support for MCUXpresso IDE and IAR

- Include image, model, and labels

```
#include "stopwatch_image.h"                    //Image to analyze
#include "mobilenet_v1_0.25_128_quant_model.h"  //Model
#include "labels.h"                             //Categories to label image
```

- Load tflite converted model with BuildFromBuffer()

```
void RunInference(Settings* s) {
  std::unique_ptr<tflite::FlatBufferModel> model;
  std::unique_ptr<tflite::Interpreter> interpreter;
  model = tflite::FlatBufferModel::BuildFromBuffer(mobilenet_model, mobilenet_model_len);  //Load model
```

- Load image and set to input tensor

```
  uint8_t* in = read_bmp(stopwatch_bmp, stopwatch_bmp_len, &image_width, &image_height,
                         &image_channels, s);

  int input = interpreter->inputs()[0];
```

```
      resize<float>(interpreter->typed_tensor<float>(input), in, image_height,
                    image_width, image_channels, wanted_height, wanted_width,
                    wanted_channels, s);
```

TensorFlow Lite “Label Image” Example Walkthrough

- Run inference with Invoke()

```
auto start_time = GetTimeInUS();
for (int i = 0; i < s->loop_count; i++) {
  if (interpreter->Invoke() != kTfLiteOk) {
    LOG(FATAL) << "Failed to invoke tflite!\r\n";
  }
}
auto end_time = GetTimeInUS();
LOG(INFO) << "Average time: " << (end_time - start_time) / 1000 << " ms\r\n";
```

- Get results from output tensor

```
int output = interpreter->outputs()[0];
TfLiteIntArray* output_dims = interpreter->tensor(output)->dims;
/* Assume output dims to be something like (1, 1, ... , size) */
auto output_size = output_dims->data[output_dims->size - 1];
switch (interpreter->tensor(output)->type) {
  case kTfLiteFloat32:
    get_top_n<float>(interpreter->typed_output_tensor<float>(0), output_size,
                     s->number_of_results, threshold, &top_results, true);
```

```
if (ReadLabels(labels_txt, &labels, &label_count) != kTfLiteOk)
  return;

LOG(INFO) << "Detected:\r\n";
for (const auto& result : top_results) {
  const float confidence = result.first;
  const int index = result.second;
  LOG(INFO) << "  " << labels[index] << " (" << (int)(confidence * 100) << "% confidence)\r\n";
```

CMSIS-NN “CIFAR-10” Example Walkthrough

- Written in C
- Include image data, weights, and parameters for model

```
#include "inputs.h"       //Image data
#include "parameter.h"    //Parameters for model
#include "weights.h"      //Weights for model
```

- Load image data

```
    uint8_t image_data[32 * 32 * 3] = SHIP_IMG_DATA;
```

CMSIS-NN “CIFAR-10” Example Walkthrough

Call CMSIS-NN APIs to execute model layers

```
    /* conv1 img_buffer2 -> img_buffer1 */
    arm_convolve_HWC_q7_RGB(img_buffer2, CONV1_IM_DIM, CONV1_IM_CH, conv1_wt, CONV1_OUT_CH, CONV1_KER_DIM, CONV1_PADDING,
                            CONV1_STRIDE, conv1_bias, CONV1_BIAS_LSHIFT, CONV1_OUT_RSHIFT, img_buffer1, CONV1_OUT_DIM,
                            (q15_t *) col_buffer, NULL);

    arm_relu_q7(img_buffer1, CONV1_OUT_DIM * CONV1_OUT_DIM * CONV1_OUT_CH);

    /* pool1 img_buffer1 -> img_buffer2 */
    arm_maxpool_q7_HWC(img_buffer1, CONV1_OUT_DIM, CONV1_OUT_CH, POOL1_KER_DIM,
                       POOL1_PADDING, POOL1_STRIDE, POOL1_OUT_DIM, NULL, img_buffer2);

    /* conv2 img_buffer2 -> img_buffer1 */
    arm_convolve_HWC_q7_fast(img_buffer2, CONV2_IM_DIM, CONV2_IM_CH, conv2_wt, CONV2_OUT_CH, CONV2_KER_DIM,
                             CONV2_PADDING, CONV2_STRIDE, conv2_bias, CONV2_BIAS_LSHIFT, CONV2_OUT_RSHIFT, img_buffer1,
                             CONV2_OUT_DIM, (q15_t *) col_buffer, NULL);

    arm_relu_q7(img_buffer1, CONV2_OUT_DIM * CONV2_OUT_DIM * CONV2_OUT_CH);

    /* pool2 img_buffer1 -> img_buffer2 */
    arm_maxpool_q7_HWC(img_buffer1, CONV2_OUT_DIM, CONV2_OUT_CH, POOL2_KER_DIM,
                       POOL2_PADDING, POOL2_STRIDE, POOL2_OUT_DIM, col_buffer, img_buffer2);

    /* conv3 img_buffer2 -> img_buffer1 */
    arm_convolve_HWC_q7_fast(img_buffer2, CONV3_IM_DIM, CONV3_IM_CH, conv3_wt, CONV3_OUT_CH, CONV3_KER_DIM,
                             CONV3_PADDING, CONV3_STRIDE, conv3_bias, CONV3_BIAS_LSHIFT, CONV3_OUT_RSHIFT, img_buffer1,
                             CONV3_OUT_DIM, (q15_t *) col_buffer, NULL);

    arm_relu_q7(img_buffer1, CONV3_OUT_DIM * CONV3_OUT_DIM * CONV3_OUT_CH);

    /* pool3 img_buffer-> img_buffer2 */
    arm_maxpool_q7_HWC(img_buffer1, CONV3_OUT_DIM, CONV3_OUT_CH, POOL3_KER_DIM,
                       POOL3_PADDING, POOL3_STRIDE, POOL3_OUT_DIM, col_buffer, img_buffer2);

    arm_fully_connected_q7_opt(img_buffer2, ip1_wt, IP1_DIM, IP1_OUT, IP1_BIAS_LSHIFT, IP1_OUT_RSHIFT, ip1_bias,
                               output_data, (q15_t *) img_buffer1);

    arm_softmax_q7(output_data, 10, output_data);
}
```

CMSIS-NN “CIFAR-10” Example Walkthrough

Get Results

```
/* Get the object class with the highest confidence value */
arm_max_q7(output_data, 10, &max_value, &max_index);
PRINTF("Predicted class: %s \r\n", labels[max_index]);
```

Inference Times

- Benchmarking ongoing and optimizations still under development. Numbers subject to change.
- Inference time heavily dependent on the particular model
  - Input data does not affect inference time
- Each eIQ example reports inference time

- CIFAR-10 example in IAR with Release compile settings
  - CMSIS-NN: 23ms
  - TensorFlow Lite: 72ms

Memory Requirements

- Non-volatile memory (Flash/HyperFlash) stores the model, inference engine, and input data
- Volatile memory (SRAM/SDRAM) stores the intermediate products of the model layers
  - Amount required depends on a lot of factors like the amount, size, and type of the layers.

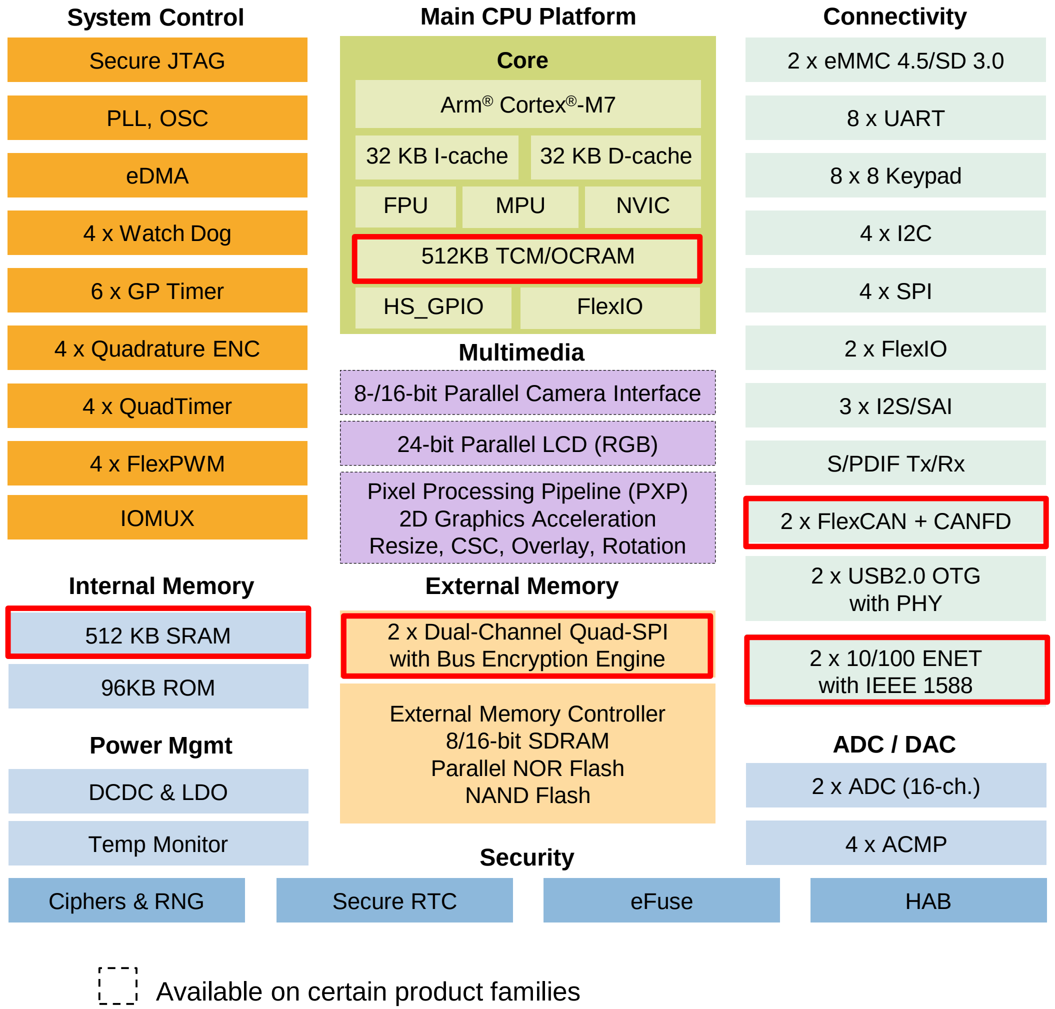
Benchmarking ongoing and optimizations still under development. Numbers subject to change:

- **CMSIS-NN with CIFAR-10:** 110KB Flash, 50KB RAM
- **TensorFlow Lite with CIFAR-10:** 600KB Flash (92KB for model, 450KB for inference engine), 320KB RAM
- **TensorFlow Lite with Label Image:** 1.5MB Flash (450KB for model, 450KB for inference engine, 450KB for input photo), 2.5MB RAM

eIQ Hands-On with Label Image

NXP 32-bit Arm-based MCUs – High Performance

i.MX RT1060: Block Diagram



Specifications

- Package: MAPBGA196 | 10x10mm^2, 0.65mm pitch (130 GPIOs)
- Temp / Qual: -40 to 105°C (Tj) Industrial / 0 to 95°C (Tj) Consumer

High Performance Real Time system

- Cortex-M7 up to 600MHz , 50% faster than any other existing M7 products
- 20ns interrupt latency, a TRUE Real time processor
- 512KB SRAM + 512KB TCM/OCRAM

Rich Peripheral

- Motor Control: Flex PWM X 4, Quad Timer X 4, ENC X 4
- 2x USB, 2x SDIO, 2x CAN + 1x CANFD, 2x ENET with 1588, 8xUART, 4x SPI, 4X I2C
- 8/16-bit CSI interface and 8/16/24-bit LCD interface
- 2x Qual-SPI interface, with Bus Encryption Engine
- Audio interface: 3x SAI/ SPDIF RX & TX/ 1x ESAI

Security

- TRNG&PRNG(NIST SP 800-90 Certified)
- 128-AES cryptography
- Bus Encryption Engine: Protect QSPI Flash Content

Ease of Use

- MCUXpresso with SDK
- FreeRTOS
- Comprehensive ecosystem

Low BOM Cost

- Competitive Price
- Fully integrated PMIC with DC-DC
- Low cost package, 10x10 BGA with 0.65mm Pitch
- SDRAM interface

Transfer Learning and Inference Lab

- Can take a pre-existing model and train it on new input
  - Allows much quicker training on an already known good model
  - Need to ensure model type is a good match for the type of data being retrained for
    - Some models better at image recognition. Others at speech.

- Lab will re-train a Mobilenet model built with TensorFlow to categorize 5 different flower types in images
  - This can be used then for any types of images that customer is interested in

- Skip Section 2 as all programs have already been installed on lab computers

Wrap-Up and Q&A

Agenda

- Artificial Intelligence/Machine Learning
- eIQ
- eIQ on i.MXRT
- Hands-On
- Q&A and Wrap-up

Further Reading

- NXP eIQ
- TensorFlow Lite
- CMSIS-NN

Machine Learning Courses:

- Video series on Neural Network basics
- ARM Embedded Machine Learning for Dummies
- Google TensorFlow Lab
- Google Machine Learning Crash Course
- Google Image Classification Practica

Git Repos

- TensorFlow Lite
  - https://github.com/tensorflow/tensorflow/tree/v1.13.1/tensorflow/lite

- CMSIS-NN
  - https://github.com/ARM-software/CMSIS_5/tree/master/CMSIS/NN
  - CIFAR-10: https://github.com/ARM-software/ML-examples/tree/master/cmsisnn-cifar10
  - KWS: https://github.com/ARM-software/ML-KWS-for-MCU

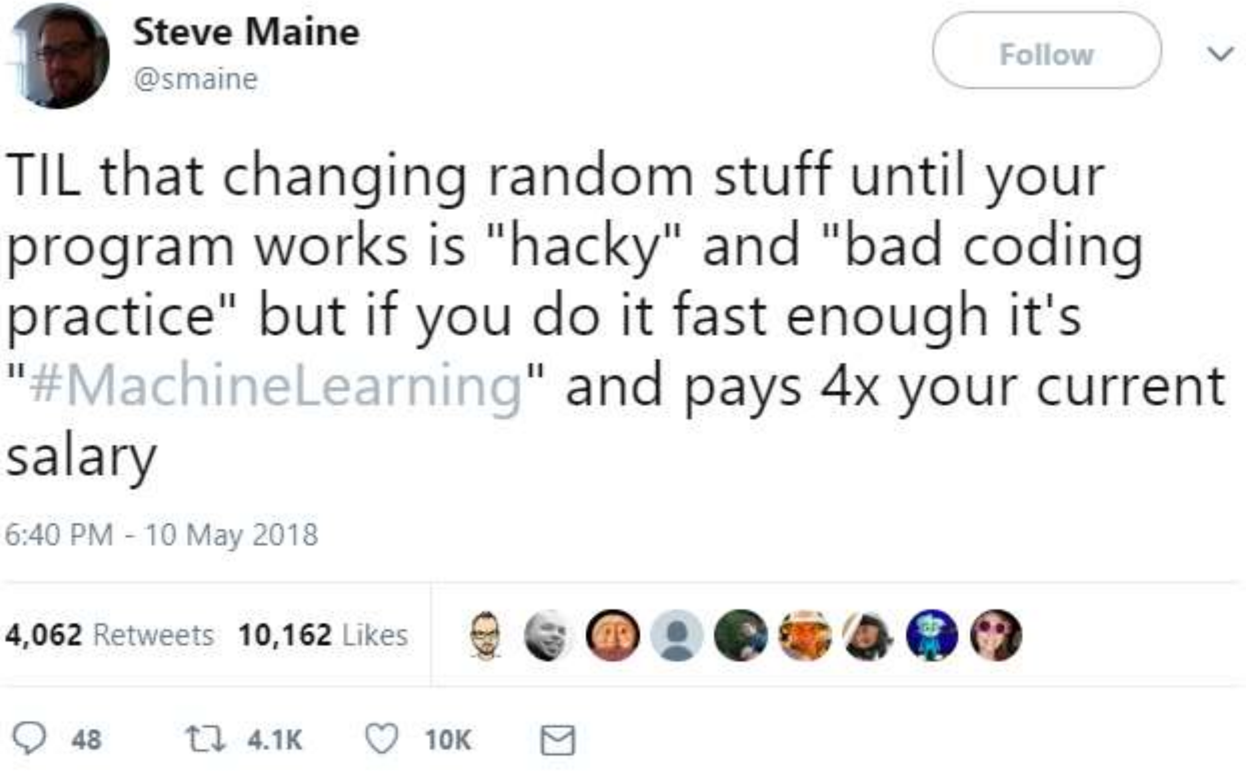
- Glow
  - https://github.com/pytorch/glow

NXP eIQ Resources

- eIQ for iMX and RT on MCUXpresso SDK Builder
  - https://mcuxpresso.nxp.com

- Available for i.MXRT1050 and i.MXRT1060

Questions?


Steve Maine
@smaine

Follow

TIL that changing random stuff until your program works is "hacky" and "bad coding practice" but if you do it fast enough it's "#MachineLearning" and pays 4x your current salary

6:40 PM - 10 May 2018

4,062 Retweets 10,162 Likes

48 4.1K 10K

SECURE CONNECTIONS
FOR A SMARTER WORLD