TENSORFLOW LITE DELEGATES ON ARM-BASED DEVICES

Pavel Macenauer
Senior Software Engineer, NXP

MARCH 2021

SECURE CONNECTIONS FOR A SMARTER WORLD

PUBLIC

TF Lite Model Deployment

Delegates Overview

Partitioning and Implementing a Custom Delegate

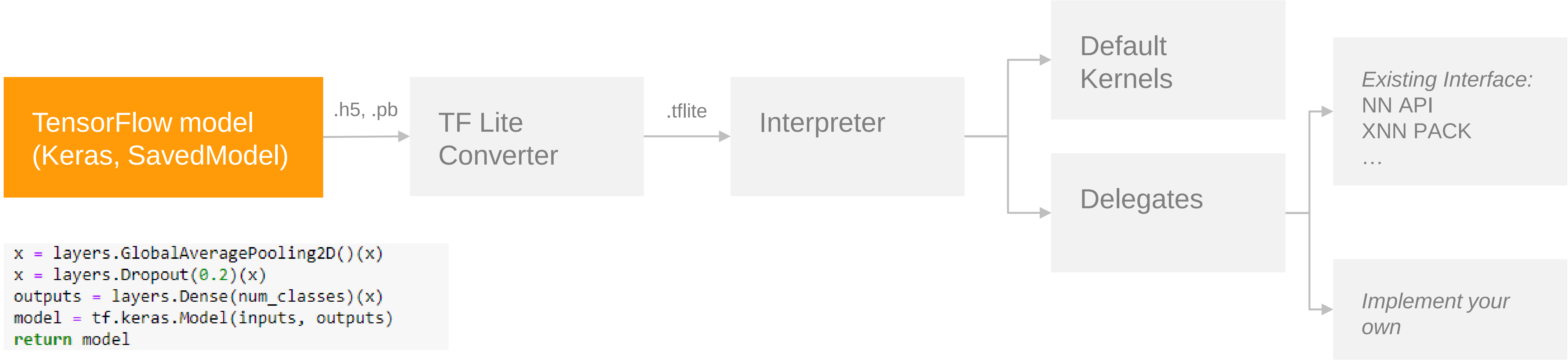
Benchmarking and Corectness

Delegates More Thoroughly

- NNAPI Delegate
- ARM NN Delegate

Python and C++ Code Samples

TF LITE MODEL DEPLOYMENT




```
x = layers.GlobalAveragePooling2D()(x)
x = layers.Dropout(0.2)(x)
outputs = layers.Dense(num_classes)(x)
model = tf.keras.Model(inputs, outputs)
return model
```

- Develop a model in TensorFlow or load from *.h5 (Keras) / *.pb (SavedModel/checkpoint)

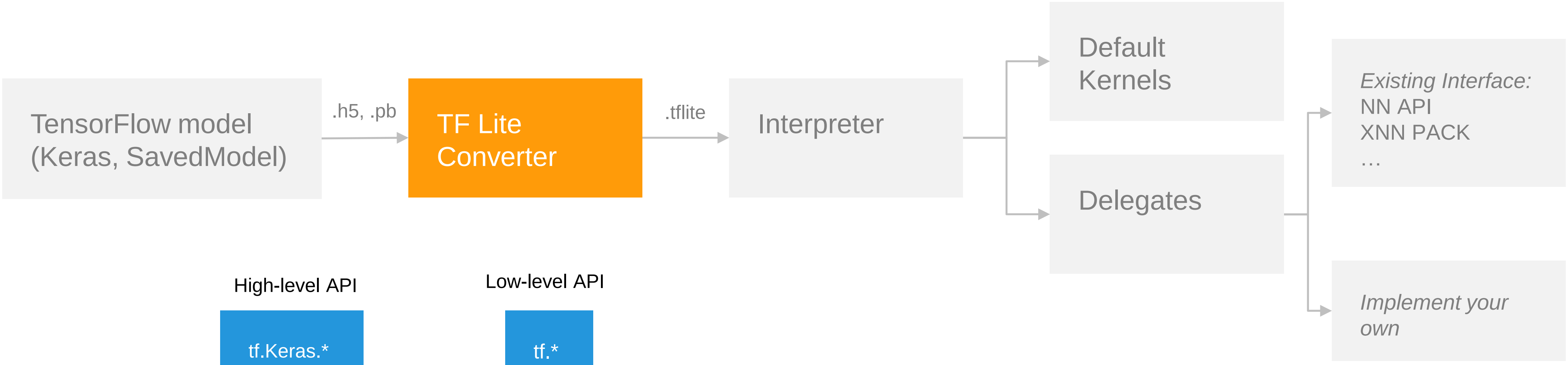
TensorFlow is a popular machine learning opensource framework developed by Google

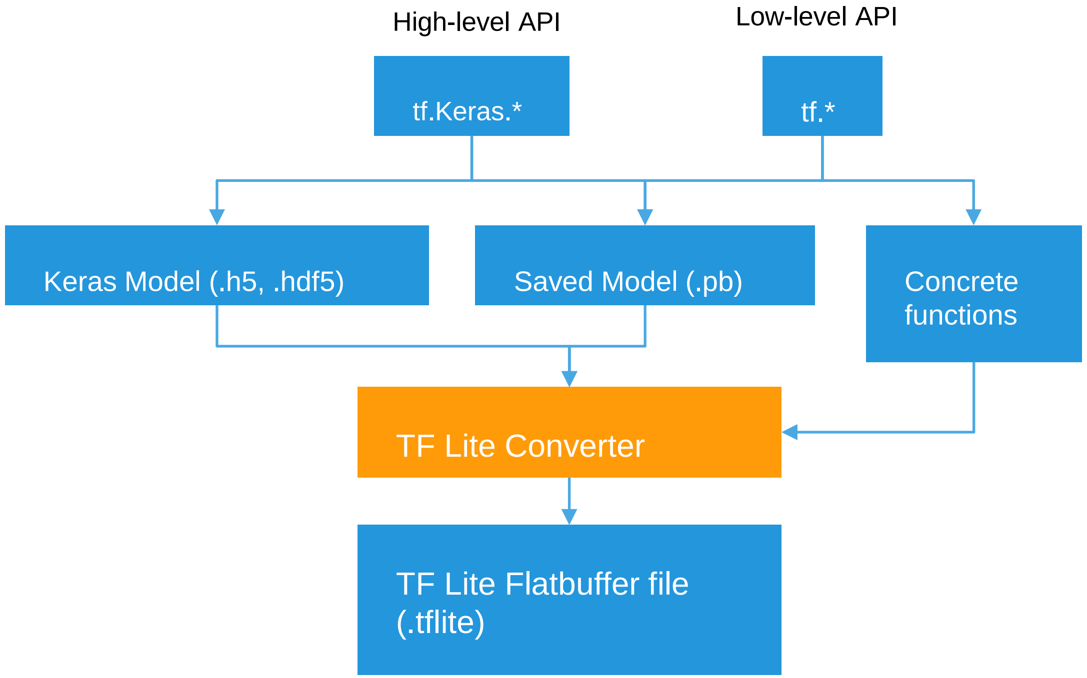
TF Lite is a module targeted mostly for inference on IoT / embedded devices

(for Microcontrollers there is the new TF Micro, which we will not cover)

TensorFlow

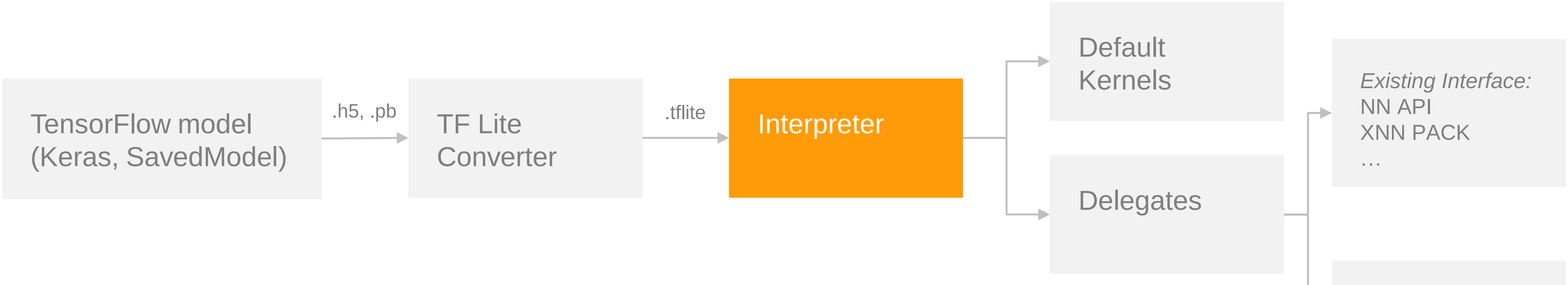
TF LITE MODEL DEPLOYMENT





- Convert to Flatbuffer (serialized) format
- Can be converted to uint8 (quantized) or kept in float32
- Enables different quantization methods
  - Float16, uint8, hybrid uint8/uint16

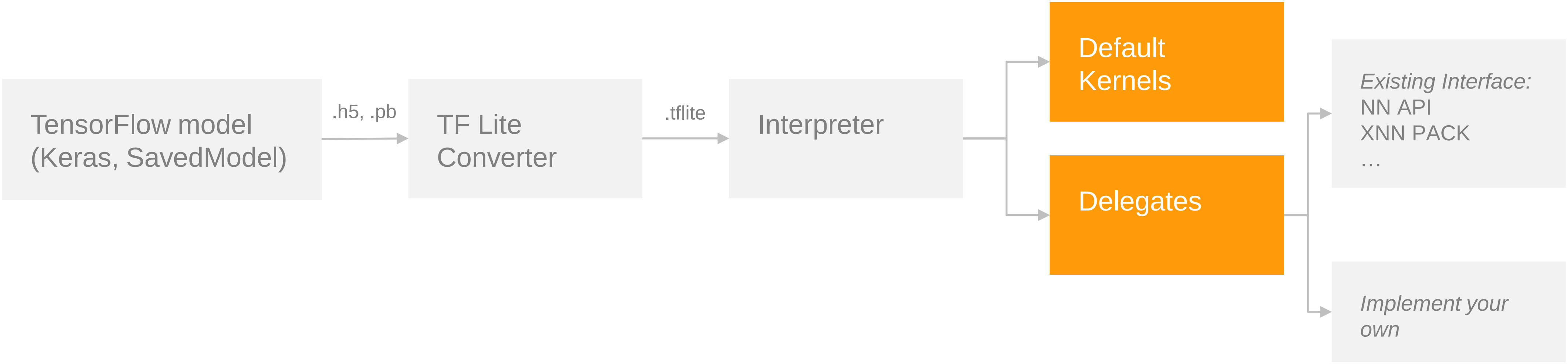
TF LITE MODEL DEPLOYMENT




```
tf.lite.Interpreter(
    model_path=None, model_content=None, experimental_delegates=None,
    num_threads=None
)
```

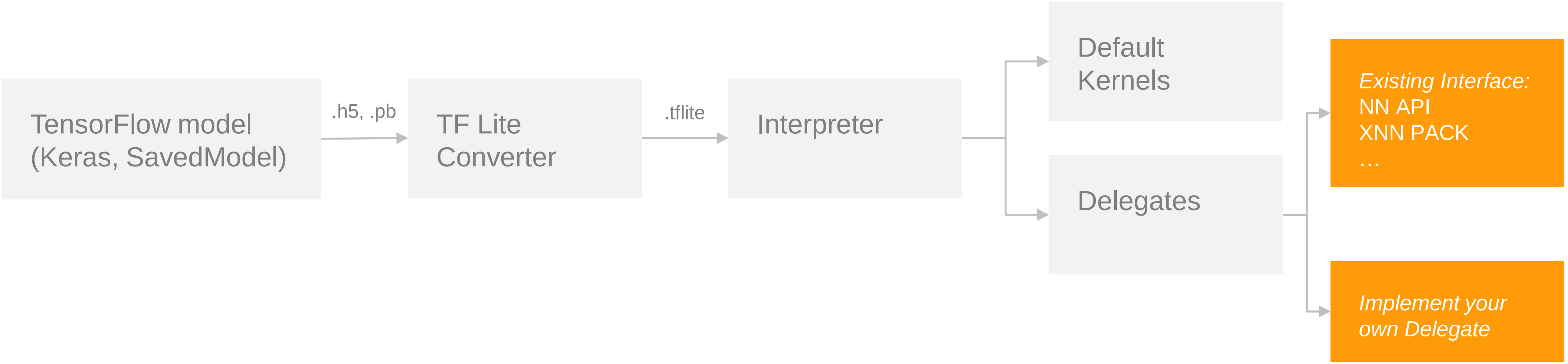
- Runs the model on the device (by default on the CPU)
- `experimental_delegates` enables `TfLiteDelegate` API
    - To implement a custom delegate see also `SimpleDelegate` API, which is a wrapper

TF LITE MODEL DEPLOYMENT



- Default kernels run on the CPU and they are optimized for Arm NEON
- Delegates are able to offload execution to a different device (GPU, NPU, DSP, …)

TF LITE MODEL DEPLOYMENT



- Examples of existing delegates are NNAPI (Android), XNN Pack, GPU (OpenCL), Hexagon DSP, CoreML, …
  - NNAPI defines an interface, implementation is found on the device
  - NXP i.MX8 microprocessors use NN API delegate to offload execution to the GPU or the NPU depending on what is available
  - XNN Pack Delegate is an alternative to the default CPU kernels
- A custom delegate can be provided – examples are **VX Delegate**, **Arm NN Delegate**, …
  - https://github.com/ARM-software/armnn (under delegate)
  - https://github.com/VeriSilicon/TIM-VX (in development)

WHY TO USE A DELEGATE AND WHAT ARE THE DOWNSIDES?

PROS

- Specialized hardware provides better performance and/or power consumption
  - Hardware such as a GPU/NPU/DSP if available
  - A lot of hardware fuses operations (activations, pooling layers, FC, …)
  - Possible to free CPU for other tasks

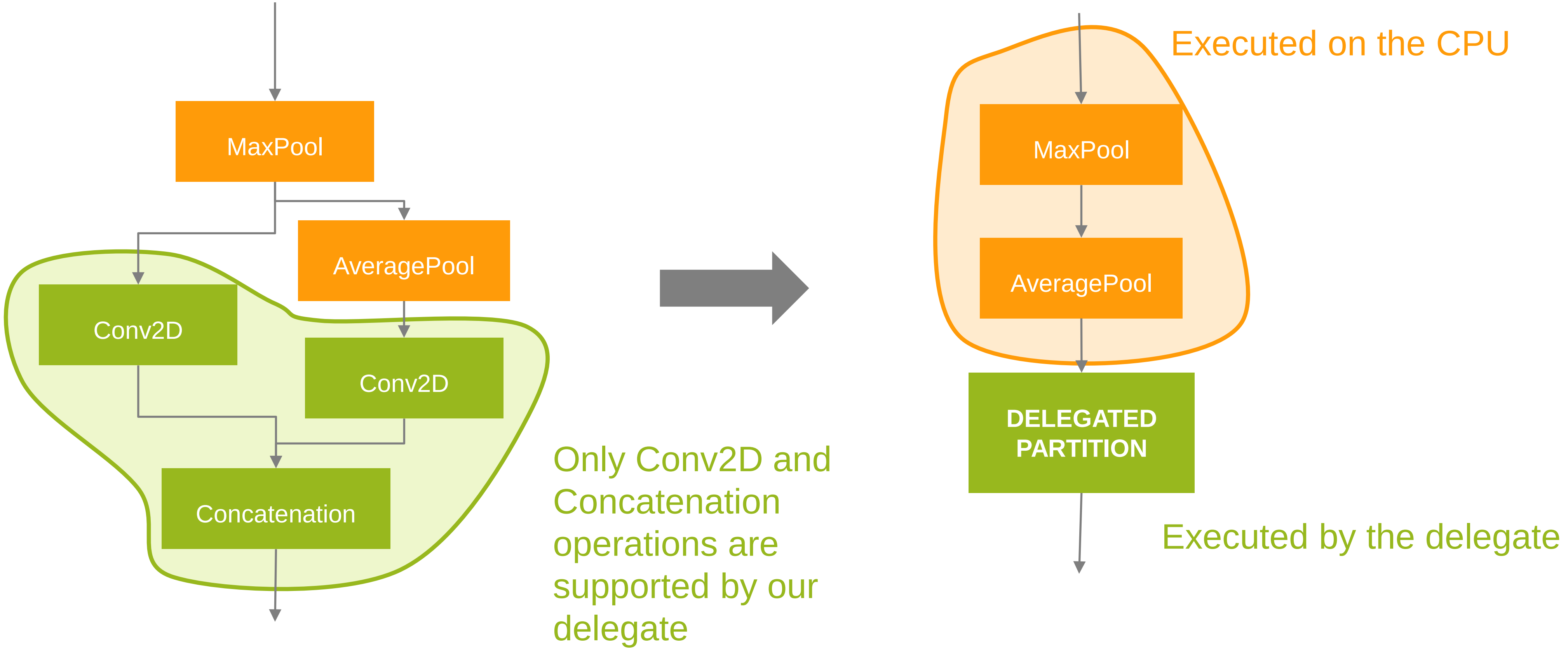
CONS

- Unsupported ops run on the CPU and cause performance degradation due to additional tensor copies

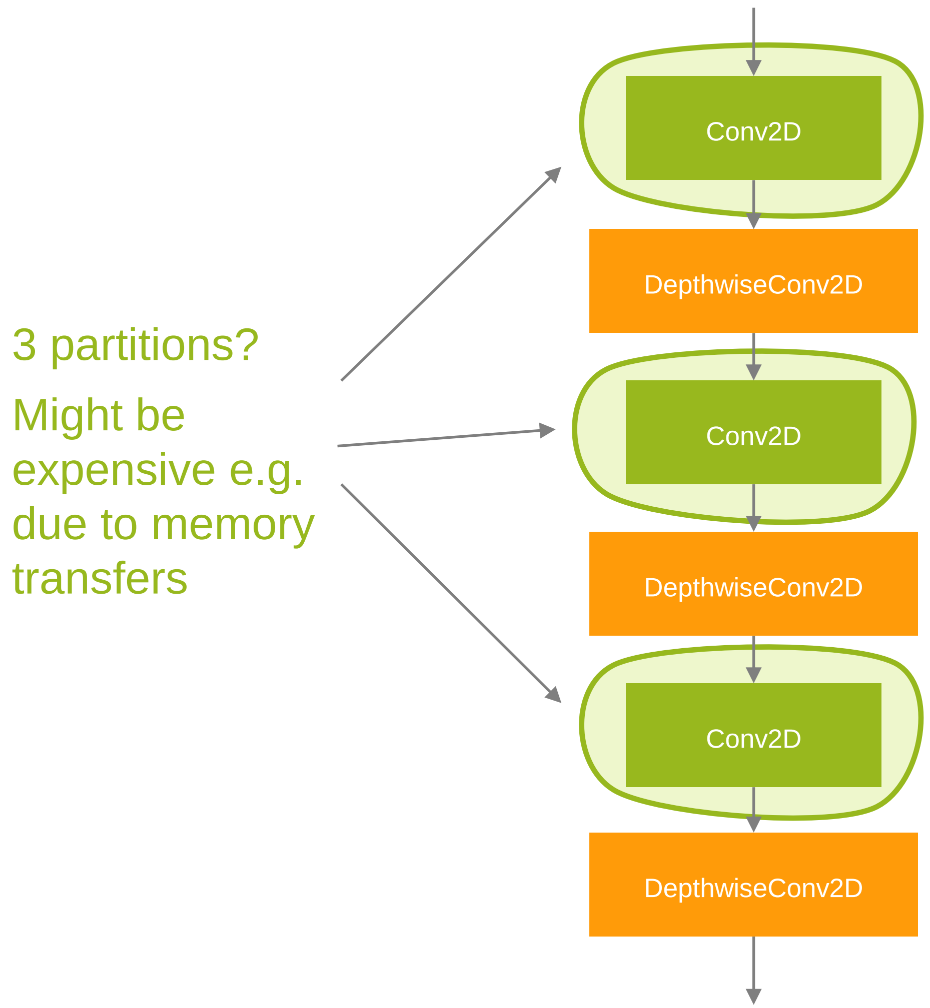
- Memory transfer can become a bottleneck especially when graph is partitioned a lot

DELEGATE PARTITIONING

- Graph is partitioned based on op support



SWITCHING BETWEEN PARTITIONS



The preferred method to implement a Delegate is using *SimpleDelegate API*:

SimpleDelegateInterface

- Capabilities of the Delegate (options), op support, factory class

SimpleDelegateKernelInterface

- Logic for initializing, preparing and running the delegated partitions

COMMON PARAMETERS *(SimpleDelegateInterface)*

num_threads: int (default=1)

- The number of threads to use for running the inference on CPU.

max_delegated_partitions: int (default=0, i.e. no limit)

- The maximum number of partitions that will be delegated.
- Currently supported by the **GPU**, **Hexagon**, **CoreML** and **NNAPI** delegate.

min_nodes_per_partition: int (default=delegate's own choice)

- The minimal number of TFLite graph nodes of a partition that needs to be reached to be delegated. A negative value or 0 means to use the default choice of each delegate.
- This option is currently supported by the **Hexagon** and **CoreML** delegate.

… and other Delegate specific options

benchmark_model

- Simple tool to evaluate performance and memory
  - average inference latency
  - initialization overhead
  - memory footprint

```
./benchmark_model  \
--graph=mobilenet_v1_1.0_224_quant.tflite \
--use_nnapi=true
```

```
STARTING!
…
The input model file size (MB): 4.27635
Initialized session in 3.84ms.
Running benchmark for at least 1 iterations and at least 0.5 seconds
but terminate if exceeding 150 seconds.
count=1 curr=6036807

Running benchmark for at least 50 iterations and at least 1 seconds but
terminate if exceeding 150 seconds.
count=361 first=2795 curr=2704 min=2653 max=2849 avg=2693.12 std=21

Inference timings in us: Init: 3840, First inference: 6036807, Warmup
(avg): 6.03681e+06, Inference (avg): 2693.12

Note: as the benchmark tool itself affects memory footprint, the
following is only APPROXIMATE to the actual memory footprint of the
model at runtime. Take the information at your discretion.
Peak memory footprint (MB): init=2.75391 overall=29.0273
```

CORRECTNESS

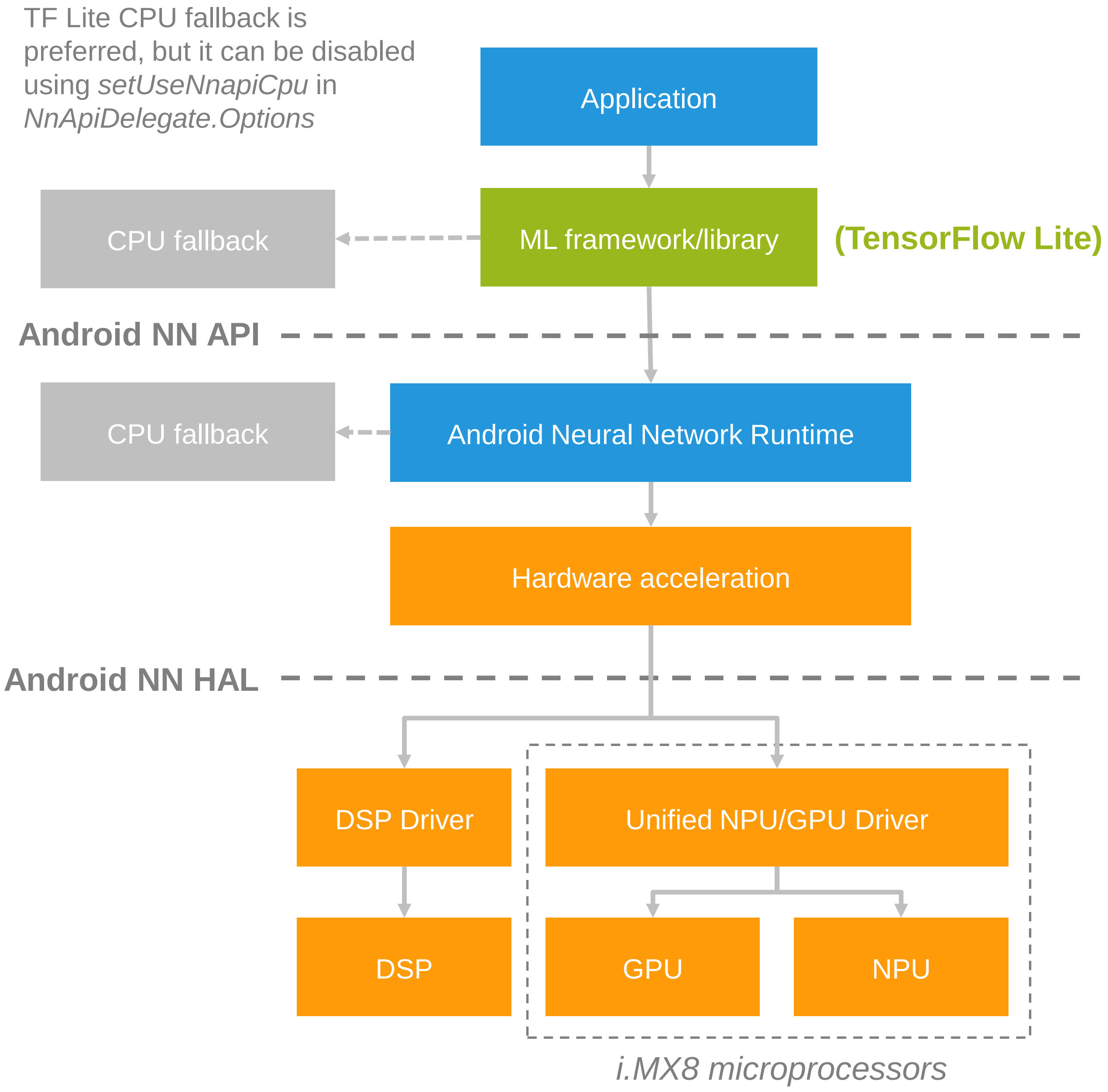
inference_diff

- Simple tool to evaluate correctness

```
num_runs: 50
process_metrics {
    inference_profiler_metrics {
        …
        output_errors {
            max_value: 0.000999001
            min_value: 0
            avg_value: 1.5448780194240654 8e-05
            std_deviation: 0.00029687365
        }
    }
}
```

NN API DELEGATE

- Android C API designed to run machine learning operations on Android devices
- Limited to *float16*, *float32*, *int8* and *uint8*
- Supports acceleration on a GPU, an NPU or a DSP depending on the target device

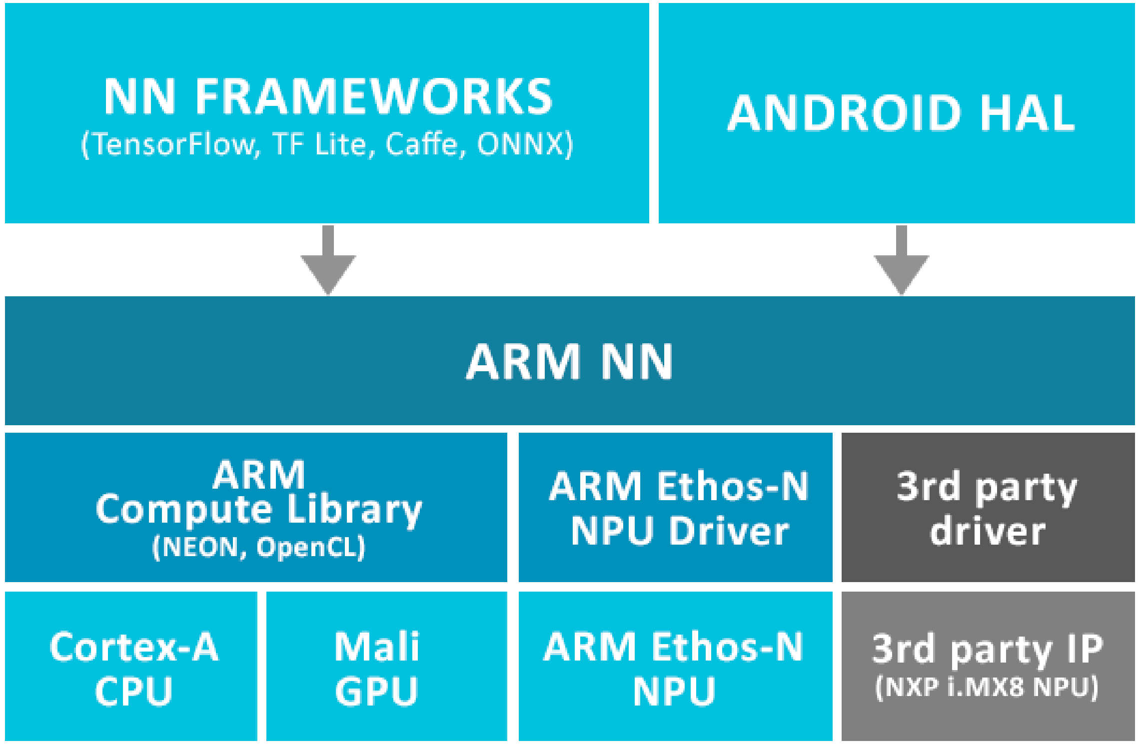


WHAT IS ARM NN?

- A **middleware inference engine for machine learning on the edge donated mid-2018 to Linaro AI initiative**
  - Single API integrating popular high-level ML frameworks (TensorFlow, TF Lite, Caffe, ONNX – MXNet, PyTorch)
  - Connects high-level ML frameworks to compute engines, drivers, HW through Arm NN backends
  - Optimized for ARM and NXP hardware
    - Cortex-A CPUs, Mali GPUs, Ethos-N NPUs
    - i.MX8 microprocessors (Cortex-A CPUs + GPU/NPU for acceleration)

https://www.mlplatform.org

https://github.com/ARM-software/armnn



ARM NN DELEGATE

- Released originally in 20.11, but much more mature in 21.02 (https://github.com/ARM-software/armnn/releases/tag/v21.02)
- For installation, build, integration see the docs: https://arm-software.github.io/armnn/21.02/

- Allows to offload execution to all the backends supported by Arm NN
  - Arm Compute Library (Arm NEON – Cortex-A CPU, OpenCL – GPU/Mali)
  - Arm Ethos-N NPU
  - Custom backends such as NXP's GPU/NPU (VSI NPU)
- Allows to replace Arm NN parser front-end with TF Lite

- Builds as a dynamic/shared library (libarmnnDelegate.so) which is linked by the target application
- Builds standalone or together with Arm NN (*CMake*)
- Requires prebuilt TF Lite as the only dependency

PYTHON API ARM NN EXAMPLE

- Import any Python module such as numpy
- tflite_runtime (Python wrapper) is required

```
import numpy as np
import tflite_runtime.interpreter as tflite

# Load TFLite model and allocate tensors
armnn_delegate = tflite.load_delegate(library="/usr/lib/libarmnnDelegate.so",
                                      options={"backends": "VsiNpu, CpuAcc, CpuRef",
                                               "logging-severity": "info"})

# Delegates/Executes all operations supported by ArmNN to/with ArmNN
interpreter = tflite.Interpreter(model_path="mobilenet_v1_1.0_224_quant.tflite",
                                 experimental_delegates=[armnn_delegate])


# Now we may allocate input, output tensors, and run inference
interpreter.allocate_tensors()
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
input_shape = input_details[0]['shape']
input_data = np.array(np.random.random_sample(input_shape), dtype=np.uint8)
interpreter.set_tensor(input_details[0]['index'], input_data)
interpreter.invoke()

# Print out result
output_data = interpreter.get_tensor(output_details[0]['index'])
```

PYTHON API ARM NN EXAMPLE

- Load the dynamic delegate
- Set delegate options – most importantly choose all the backends

```
import numpy as np
import tflite_runtime.interpreter as tflite

# Load TFLite model and allocate tensors
armnn_delegate = tflite.load_delegate(library="/usr/lib/libarmnnDelegate.so",
                                      options={"backends": "VsiNpu, CpuAcc, CpuRef",
                                               "logging-severity": "info"})

# Delegates/Executes all operations supported by ArmNN to/with ArmNN
interpreter = tflite.Interpreter(model_path="mobilenet_v1_1.0_224_quant.tflite",
                                 experimental_delegates=[armnn_delegate])


# Now we may allocate input, output tensors, and run inference
interpreter.allocate_tensors()
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
input_shape = input_details[0]['shape']
input_data = np.array(np.random.random_sample(input_shape), dtype=np.uint8)
interpreter.set_tensor(input_details[0]['index'], input_data)
interpreter.invoke()

# Print out result
output_data = interpreter.get_tensor(output_details[0]['index'])
```

PYTHON API ARM NN EXAMPLE

- Load TF Lite model
- Link the Arm NN delegate instance

```
import numpy as np
import tflite_runtime.interpreter as tflite

# Load TFLite model and allocate tensors
armnn_delegate = tflite.load_delegate(library="/usr/lib/libarmnnDelegate.so",
                                      options={"backends": "VsiNpu, CpuAcc, CpuRef",
                                               "logging-severity": "info"})

# Delegates/Executes all operations supported by ArmNN to/with ArmNN
interpreter = tflite.Interpreter(model_path="mobilenet_v1_1.0_224_quant.tflite",
                                 experimental_delegates=[armnn_delegate])


# Now we may allocate input, output tensors, and run inference
interpreter.allocate_tensors()
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
input_shape = input_details[0]['shape']
input_data = np.array(np.random.random_sample(input_shape), dtype=np.uint8)
interpreter.set_tensor(input_details[0]['index'], input_data)
interpreter.invoke()

# Print out result
output_data = interpreter.get_tensor(output_details[0]['index'])
```

PYTHON API ARM NN EXAMPLE

- Set inputs, process outputs, run inference

```
import numpy as np
import tflite_runtime.interpreter as tflite

# Load TFLite model and allocate tensors
armnn_delegate = tflite.load_delegate(library="/usr/lib/libarmnnDelegate.so",
                                      options={"backends": "VsiNpu, CpuAcc, CpuRef",
                                               "logging-severity": "info"})

# Delegates/Executes all operations supported by ArmNN to/with ArmNN
interpreter = tflite.Interpreter(model_path="mobilenet_v1_1.0_224_quant.tflite",
                                 experimental_delegates=[armnn_delegate])


# Allocate input, output tensors, and run inference
interpreter.allocate_tensors()
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
input_shape = input_details[0]['shape']
input_data = np.array(np.random.random_sample(input_shape), dtype=np.uint8)
interpreter.set_tensor(input_details[0]['index'], input_data)
interpreter.invoke()

# Print out result
output_data = interpreter.get_tensor(output_details[0]['index'])
```

C++ API ARMNN EXAMPLE

- C++ API is slightly more verbose than Python

```
#include "armnn_delegate.hpp"
...
std::vector<armnn::BackendId> backends = {armnn::Compute::VsiNpu,
                                          armnn::Compute::CpuAcc,
                                          armnn::Compute::CpuRef };

// Create the ArmNN Delegate
armnnDelegate::DelegateOptions delegateOptions(backends);

std::unique_ptr<TfLiteDelegate, decltype(&armnnDelegate::TfLiteArmnnDelegateDelete)>
        theArmnnDelegate(armnnDelegate::TfLiteArmnnDelegateCreate(delegateOptions),
                         armnnDelegate::TfLiteArmnnDelegateDelete);

// Instruct the Interpreter to use the armnnDelegate
interpreter->ModifyGraphWithDelegate(theArmnnDelegate.get());

// Now you may allocate input, output tensors, and run inference
interpreter->AllocateTensors()
...
```

LINKS TO BEGIN WITH TF LITE

TensorFlow Lite
https://www.tensorflow.org/lite

Arm NN 21.02 with TF Lite Delegate
https://github.com/ARM-software/armnn

TensorFlow fork for NXP i.MX8 microprocessors:
https://source.codeaurora.org/external/imx/tensorflow-imx

NN API delegate and VSI NPU backend for Arm NN for NXP i.MX8 microprocessors
https://source.codeaurora.org/external/imx/nn-imx/

SECURE CONNECTIONS
FOR A SMARTER WORLD