**Documentation - TensorFlow Lite MetaWareNN Delegate**

**Purpose:**

The purpose of this document is to give an overview of MetaWareNN Delegate implementation and to explain workflow from the TensorFlow Lite framework.

**Overview:**

Machine Learning models are being loaded using the ML frameworks. The execution of the models is happening in the host environment (x86 / ARM platform). In order to enable graph execution in other platforms/devices, these ML frameworks added specific implementations to convert ML models to graph Intermediate Representation (IR) and then use the IR to execute the graph in expected devices(DSP, etc).

In TFLite, delegates like Hexagon, NNAPI, etc have its specific implementation to build graphs from the TFLite model corresponding to each of the delegates. We are trying to have MetaWareNN IR common between ONNXRuntime, TFLite & GLOW frameworks.

**TFLite - MetaWareNN Repository:**

* Cloned TensorFlow repository (Tag [v2.3.1](https://github.com/foss-for-synopsys-dwc-arc-processors/synopsys-tensorflow/tree/v2.3.1/)) and created a [metawarenn\_dev](https://github.com/foss-for-synopsys-dwc-arc-processors/synopsys-tensorflow/tree/metawarenn_dev) branch to incorporate MetaWareNN delegate related code changes
* Added BUILD files along with MetaWareNN source files and linked it to tensorflow lite shared object file to infer the ML model with MetaWareNN Delegate - [Initial commit](https://github.com/foss-for-synopsys-dwc-arc-processors/synopsys-tensorflow/commit/c9adc22623700be576901256c75f43b4568867a4)
* Added code changes to generate high level MetaWareNN Graph, apply graph transformations(passes) & generate low level MetaWareNN Graph and further generate Inference Engine & Execution Context using [metawarenn\_lib](https://github.com/SowmyaDhanapal/metawarenn_lib/tree/onnx_conversion) in onnx\_conversion branch
* **FLAGS to Control Code Flow**
  + By default, *INFERENCE\_ENGINE* is disabled which will create output ONNXProto directly from MWNNGraph and dumps the ONNX model to *op\_onnx\_models*
  + Enable *INFERENCE\_ENGINE* flag to convert MWNNGraph to ExecutableGraph and cache it to EXEC\_DUMPS path and then create Inference Engine & Execution Context and finally dumps the output ONNX model in *op\_onnx\_models*
  + Enable *INVOKE\_NNAC* flag to generate a MetaWareNN Graph proto from the low level MetaWareNN Graph and serialize it to a binary file. EVConvert python module which is integrated in MetaWareNN library will generate a Caffe prototxt & Caffemodel from MetaWareNN binary proto file, and finally evgencnn executable will generate a EV binary using the Caffe files
    - [Note] : INVOKE\_NNAC flag is Outdated and not tested after MWNNGraph update to ONNX format

**Trigger MetaWareNN Delegate:**

* [Create](https://github.com/foss-for-synopsys-dwc-arc-processors/synopsys-tensorflow/blob/f0093077b3f51bb9bd9d40a5aa3d3d83ae029f13/tensorflow/lite/delegates/MetaWareNN/inference/inference_metawarenn.cpp#L38) the MetaWareNN delegate from the top level [inference script](https://github.com/foss-for-synopsys-dwc-arc-processors/synopsys-tensorflow/blob/metawarenn_dev/tensorflow/lite/delegates/MetaWareNN/inference/inference_metawarenn.cpp)
* [Modifies](https://github.com/foss-for-synopsys-dwc-arc-processors/synopsys-tensorflow/blob/08fa5dd4e2706366c828514857a1e4a0f473a157/tensorflow/lite/delegates/MetaWareNN/inference/inference_metawarenn.cpp#L26) the input graph based on the supported nodes in MetaWareNN delegate to NodeSubsets(subgraphs) and update it with MetaWareNN delegate information using TFLite interpreter
* Execute the subgraphs using [invoke call](https://github.com/foss-for-synopsys-dwc-arc-processors/synopsys-tensorflow/blob/08fa5dd4e2706366c828514857a1e4a0f473a157/tensorflow/lite/delegates/MetaWareNN/inference/inference_metawarenn.cpp#L49) through interpreter

**Build subgraph from TFLite model and its execution:**

[MetaWareNN\_delegate.cc](https://github.com/foss-for-synopsys-dwc-arc-processors/synopsys-tensorflow/blob/metawarenn_dev/tensorflow/lite/delegates/MetaWareNN/MetaWareNN_delegate.cc) - TFLiteMetaWareNNDelegate creation and triggers partition of graph to NodeSubsets based on the supported nodes list and MetaWareNN kernel registration

[MetaWareNN\_delegate\_kernel.cc](https://github.com/foss-for-synopsys-dwc-arc-processors/synopsys-tensorflow/blob/metawarenn_dev/tensorflow/lite/delegates/MetaWareNN/MetaWareNN_delegate_kernel.cc) - Implementation of Init, Prepare, Invoke for each NodeSubset in the partitioned graph which is triggered from common framework code flow

[builders](https://github.com/foss-for-synopsys-dwc-arc-processors/synopsys-tensorflow/tree/metawarenn_dev/tensorflow/lite/delegates/MetaWareNN/builders) directory - Contains Build file & model builder source file to generate the multiple levels of MetaWareNN Graph structure from TFLite NodeSubset. Also has metawarenn library submodule which contains all files related to MetaWareNN Graph handling

[model\_builder.cc](https://github.com/foss-for-synopsys-dwc-arc-processors/synopsys-tensorflow/blob/metawarenn_dev/tensorflow/lite/delegates/MetaWareNN/builders/model_builder.cc) - Implementation for creation of MetaWareNN High Level Graph Representation from TFLite SubGraph Nodes using structures like MWNNGraph, MWNNNode, MWNNTensor, etc. Creates Low Level MetaWareNN graph structure by registering and invoking basic metawarenn graph optimization passes.

If INVOKE\_NNAC is enabled, serialize it to a binary file using MetaWareNN GraphProto and invokes nnac python script which takes the MetaWareNN serialized binary and invokes [EVConvert](https://github.com/SowmyaDhanapal/metawarenn_lib/tree/metawarenn_dev/mwnnconvert/evconvert) python module to generate Caffemodel and Caffe Prototxt which is used by evgencnn executable to generate EV binary.

[inference/inference\_metawarenn.cpp](https://github.com/foss-for-synopsys-dwc-arc-processors/synopsys-tensorflow/blob/metawarenn_dev/tensorflow/lite/delegates/MetaWareNN/inference/inference_metawarenn.cpp) - Inference script which creates the MetaWareNN Delegate and invokes the model inference for float and quantized models based on options.

[inference/test\_regression\_tflite.py](https://github.com/foss-for-synopsys-dwc-arc-processors/synopsys-tensorflow/blob/metawarenn_dev/tensorflow/lite/delegates/MetaWareNN/inference/test_regression_tflite.py) - Python script to run inference for all the TFLite models in [models.txt](https://github.com/foss-for-synopsys-dwc-arc-processors/synopsys-tensorflow/blob/metawarenn_dev/tensorflow/lite/delegates/MetaWareNN/inference/models.txt) using inference\_metawarenn.cpp executable with float CLI argument

[inference/test\_regression\_quantized\_tflite.py](https://github.com/foss-for-synopsys-dwc-arc-processors/synopsys-tensorflow/blob/metawarenn_dev/tensorflow/lite/delegates/MetaWareNN/inference/test_regression_quantized_tflite.py) - Python script to run inference for all the Quantized TFLite models in [quantized\_models.txt](https://github.com/foss-for-synopsys-dwc-arc-processors/synopsys-tensorflow/blob/metawarenn_dev/tensorflow/lite/delegates/MetaWareNN/inference/quantized_models.txt) using inference\_metawarenn.cpp executable with uint8\_t CLI argument

**MetaWareNN Library Documentation:**

Refer to [this document](https://github.com/SowmyaDhanapal/metawarenn_lib/blob/onnx_conversion/docs/Documentation-MetaWareNNLibrary-Structures.docx) for the details on MetaWareNN Library Class Structures, Optimizers, Serializations & Shared memory handlings.

**Clone the Repository:**

* *git clone --recursive* [*https://github.com/foss-for-synopsys-dwc-arc-processors/synopsys-tensorflow.git*](https://github.com/foss-for-synopsys-dwc-arc-processors/synopsys-tensorflow.git)
* *cd synopsys-tensorflow*
* *git checkout metawarenn\_dev*
* *Refer this* [*README.md*](https://github.com/foss-for-synopsys-dwc-arc-processors/synopsys-tensorflow/blob/metawarenn_dev/tensorflow/lite/delegates/MetaWareNN/README.md) *file to get the detailed steps to compile and run the inference for sample models*

**Flow Chart:**

The workflow of the MetaWareNN Delegate in TFLite has been depicted in the following diagram,

