TENSAI FLOW COMPILER USER GUIDE

Version 1.1

SEPTEMBER 29, 2020

ETA COMPUTE
310 N Westlake Blvd, Suite #110, Westlake Village CA 91362

Table of Contents

Introduction

TENSAI Compiler is a tool developed at Eta Compute to aid the conversion of Neural Networks to C code. With the help of this tool a user can generate C code, from a saved model file generated through popular Machine Learning Frameworks, that can be compiled and run on Eta Products like ECM3532.

With the current release of TENSAI Compiler, the supported Machine Learning framework is limited to Tensorflow Lite, also referred to as TFLite. Figure 1 shows the workflow using the TENSAI Compiler for generation of C code.

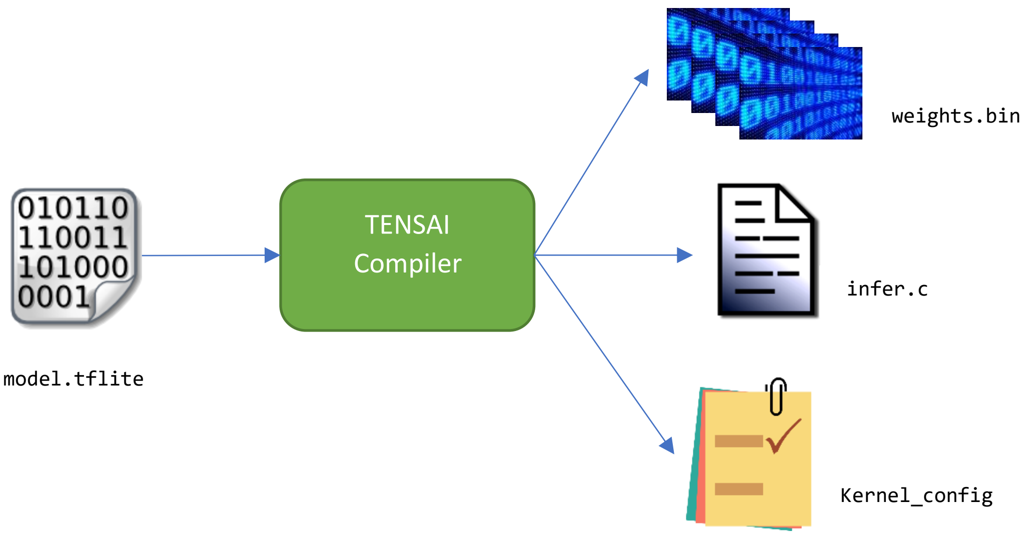


Figure 1. TENSAI Compiler Workflow

A user starts with a model file converted to TFLite format, quantized to uint8 using TF1.15, which is the format known to the compiler. The compiler will then process this model file and output a set of `.bin` files for the weights, an `infer.c` file corresponding the Neural Network(NN) model and a `Kernel_config` file enabling the appropriate executor functions. These can be used in the Project Generator to simplify the writing of NN infer function in the app. This can then be compiled after updating the main.c file by integrating the incoming data from an appropriate sensor and deployed on ECM3532.

NOTE: The tool works with the input data for up to 3 dimensions data in HWC format (or 4 dimensions, NHWC format, if the N=1).

NOTE: The user is expected to use dynamic allocation for the input and output buffers. They should also free these buffers when appropriate in the `main()` function.

The compiler will automatically handle scheduling of certain layers on DSP, utilizing the full potential of the hardware. Currently, a limited set of NN kernels(operations) are supported in the compiler, both for M3 and DSP targets.

List of Supported Kernels (M3)

The kernels that are currently supported on M3 are as below:

- Convolution 2D
- Depthwise Convolution 2D
- AveragePool 2D
- Maxpool 2D
- Reshape
- Fully connected
- Logistic (Sigmoid)
- Concatenation (height dimension only)
- Add (arrays as both operands only)

List of Supported Kernels (DSP)

Some kernels, with input, output and weight size constraints can run on DSP. The list id briefed below:

- Convolution 2D
    - Filter Size: 3x3
        - Stride: 1, 2
        - Weights size <= 8 kilobytes
        - 1 kilobyte <= Input size <= 12 kilobytes
        - Input height >= 3, Input width >= 3
    - Filter Size: 1x1
        - Weights size <= 15 kilobytes
        - Input Channel + Output Channel <= 6 kilobytes
        - Input Channel <= 2 kilobytes
        - Input Width is even.
- Depthwise Convolution 2D
    - Filter Size: 3x3
        - Stride: 1, 2
        - Weights size <= 8 kilobytes
        - 1 kilobyte <= Input size (per channel) <= 12 kilobytes
        - Input height >= 3, Input width >= 3
        - Depth Multiplier = 1

Additionally, the activation functions like relu and relu6 are supported if they are fused in the TFlite file.

If the model contains any layer that is not supported by the compiler, it will generate the error message locating the layer's index and printing the layer type.

Pre-requisites

The current tool runs only on Linux Ubuntu environment. Other than this, there are no special requirements in terms of software to be installed.

Just run the `tensaiflow_compile` binary from `TFSW/Tools/Tensai_compiler/bin`.

Familiarity with the ECM3532 Platform Software is assumed. If unfamiliar, then please refer to the file: `TFSW/Docs/Reference_Manual/ECM3532_Platform_Software_Guide.pdf`. This should help the user in understanding how to create the application folder. For the sake of simplicity, a demo app is available at: `TFSW/Applications/compiler_tool_demo`.

How to Run

Open any command line utility like `terminal` in Ubuntu based system and change directory to `platform_sw/TFSW/Tools/Tensai_compiler/bin`. As an alternative, the path to this folder can be added to the environment variable PATH in the `.bashrc` file of the user.

This tool takes in 4 command line arguments (other than `--help`) as shown in Figure .

`-i` <or> `--tflite_file` takes in a path to TFLite file that will be processed.

`-o` <or> `--out_path` takes in a path to directory where the output infer.c will be saved.

`-w` <or> `--weights_path` takes in a path to directory where the bin files for weights will be stored.

`--cifar10_dsp` is a flag, which if provided along with a specific model, then the compiler will generate the `infer.c` code which utilizes the fused DSP kernels. (Check Cifar10_wide: Fused DSP for more details)

```
~/platform_sw/TFSW/Tools/Tensai_compiler/bin$ ./tensaiflow_compile --help
Usage: tensaiflow_compile [OPTIONS]

Options:
  -i, --tflite_file FILE        Tflite file to be converted.  [required]
  -o, --out_path DIRECTORY      Path to output c code file in.  [required]
  -w, --weights_path DIRECTORY  Path to store weights file/s in.  [required]
  --cifar10_dsp                 Enable DSP code generation with fused
                                kernels, limited to cifar10_wide model from
                                model_zoo only.
  --help                        Show this message and exit.
```

Figure 2. tensiflow_compile help options

Model Zoo Examples

Along with the compiler, a model zoo targeting various NN applications using publicly available datasets is also provided at: `platform_sw/TFSW/Tools/Tensai_compiler/model_zoo`. This folder will be periodically updated with new examples based on customer feedback and use case coverage.

Currently, there are two models of image recognition types available in the model zoo in cifar10_wide and person_detection folders. Each folder contains a TFLite file and an executable to generate the input and expected output c arrays from any image of user's choice.

Additionally, demo scripts for testing these two models are available at:

`platform_sw/TFSW/Applications/compiler_tool_demo`

Cifar10_wide

Cifar10_wide is the classical example of image recognition on tiny images into 10 classes: 'airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck'. The model is pre-trained and converted to TFLite for ease of use. This is a tiny model with just 49 kilobytes of quantized weights.

The executable binary in this folder is used to generate the input c array and expected output c array which can be used in the test app to verify if the compiler has produced the source code that an get the expected output.

Just run the compiler to generate the `infer.c` and weights `.bin` files by running the compiler as shown in Figure . Note that we are leveraging the demo app directory for quick building and deployment. In case one wants to start from a new application, please check Setting up (New) Inference Application.

```
~/platform_sw/TFSW/Tools/Tensai_compiler/bin$ ./tensaiflow_compile \
> --tflite_file ../model_zoo/cifar10_wide/cifar10_wide.tflite \
> --out_path ../../../Applications/compiler_tool_demo/src/ \
> --weights_path ../../../Applications/compiler_tool_demo/include/

Total size for model weights (kB):        49
Maximum size for data buffers (kB):       16
```

Figure 3. Cifar10_wide demo using full options, M3 only

This example can then be run using the provided canned image in the demo app. Just go to the demo app folder at: `platform_sw/TFSW/Applications/compiler_tool_demo` and run the `menuconfig` to load the config file appropriate for this model as below:

```
make loadconfig \
CONFIG=../../../Platform/ECM3532/M3/configs/compiler_tool_cifar10_defconfig
```

Then, just compile the example using the `cmake` build system.

If needed, one can generate the test input and expected output by running the auxiliary script in the `cifar10_wide` folder, for the custom inputs as shown in Figure 4. Include this file in `main.c` instead of header file with canned data.

```
~/platform_sw/TFSW/Tools/Tensai_compiler/model_zoo/cifar10_wide$ \
> ./generate_cifar10_wide_input \
> --tflite_file cifar10_wide.tflite \
> --image ~/Downloads/deer.jpeg \
> --out_file ../../../../Applications/compiler_tool_demo/include/input_data.h

[[ 22   6  81  61 132  91   0 117  23  37]]
Inferred:  deer
```

Figure 4. Generating example for cifar10_wide

Fused DSP

As mentioned earlier, the compiler can produce a c code that is highly optimized by fusing kernels to run on the DSP. However, for alpha_plus release of the compiler, it is only limited to the cifar10_wide model. This will be available for generic models in one of the future releases. To generate a c file where the first two convolution and average pool layers are fused to run on DSP, use the `--cifar10_dsp` flag in the compiler as shown in Figure 5.

```
~/platform_sw/TFSW/Tools/Tensai_compiler/bin$ ./tensaiflow_compile \
> --tflite_file ../model_zoo/cifar10_wide/cifar10_wide.tflite \
> --out_path ../../../Applications/compiler_tool_demo/src/ \
> --weights_path ../../../Applications/compiler_tool_demo/include/ \
> --cifar10_dsp

Total size for model weights (kB):       49
Maximum size for data buffers (kB):      16
```

Figure 5. DSP flag on compiler for cifar10_wide

All the other steps for generating inputs and building the app are same.

Person Detection

Person detection example is based on Google's visual wake words model for microcontrollers. This model is based on MobilenetV1 model with a depth multiplier of 0.25 and input size of 96x96, which just detects presence of a person in an image. This model has 206 kilobytes of weights.

The original trained and quantized model published by the Tensorflow Lite team is in available in a c array, but for convenience, it is converted to TFLite file and provided as a part of model zoo at:

`Tensai_compiler/model_zoo/person_detection/person_detection.tflite`

Generate the `infer.c` and weights `.bin` files by using the compiler as shown in Figure 6.

```
~/platform_sw/TFSW/Tools/Tensai_compiler/bin$ ./tensaiflow_compile \
> --tflite_file ../model_zoo/person_detection/person_detection.tflite \
> --out_path ../../../Applications/compiler_tool_demo/src/ \
> --weights_path ../../../Applications/compiler_tool_demo/include/

Total size for model weights (kB):       206
Maximum size for data buffers (kB):      54
```

Figure 6. person_detection compiler command

This example can then be run using the provided canned image in the demo app. Just go to the demo app folder at: `platform_sw/TFSW/Applications/compiler_tool_demo/include` and run the `menuconfig` to load the config file appropriate for this model as below:

```
make loadconfig \
CONFIG=../../../Platform/ECM3532/M3/configs/compiler_tool_
person_det_defconfig
```

Then, just compile the example using the `cmake` build system.

Setting up (New) Inference Application

The platform software guide will enable any user with creating the new application for inference, however, one needs to update this app so that it can build and run.

Konfig File

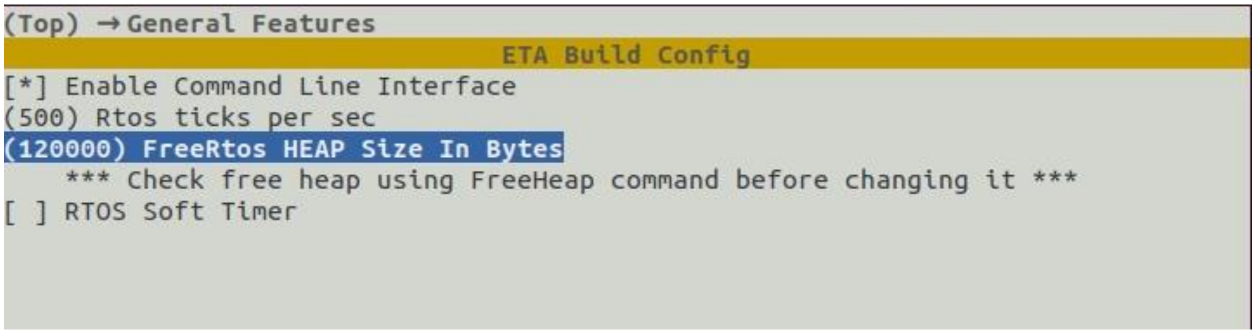
Each new app will have a `Konfig` file to configure the project options. When starting from a new app project, one needs to add the below line:

```
orsource "Kernel_config"
```

Doing this, will enable the executor functions that are used in the `infer.c` file. `Kernel_config` is generated by the tool, so one does not need to worry about writing this themselves.

Heap Size

In a new app, one setting that may need to be changed is the FreeRTOS Heap Size. This can be changed using the `menuconfig` option under General Features as shown in Figure 7.



```
(Top) → General Features
                              ETA Build Config
[*] Enable Command Line Interface
(500) Rtos ticks per sec
(120000) FreeRtos HEAP Size In Bytes
    *** Check free heap using FreeHeap command before changing it ***
[ ] RTOS Soft Timer
```

Figure 7. FreeRTOS Heap Size setting

In general, the heap size should be about 50 kilobytes more than the size of Data Buffer reported by the tool when generating the `infer.c` file.