Drive for better vision

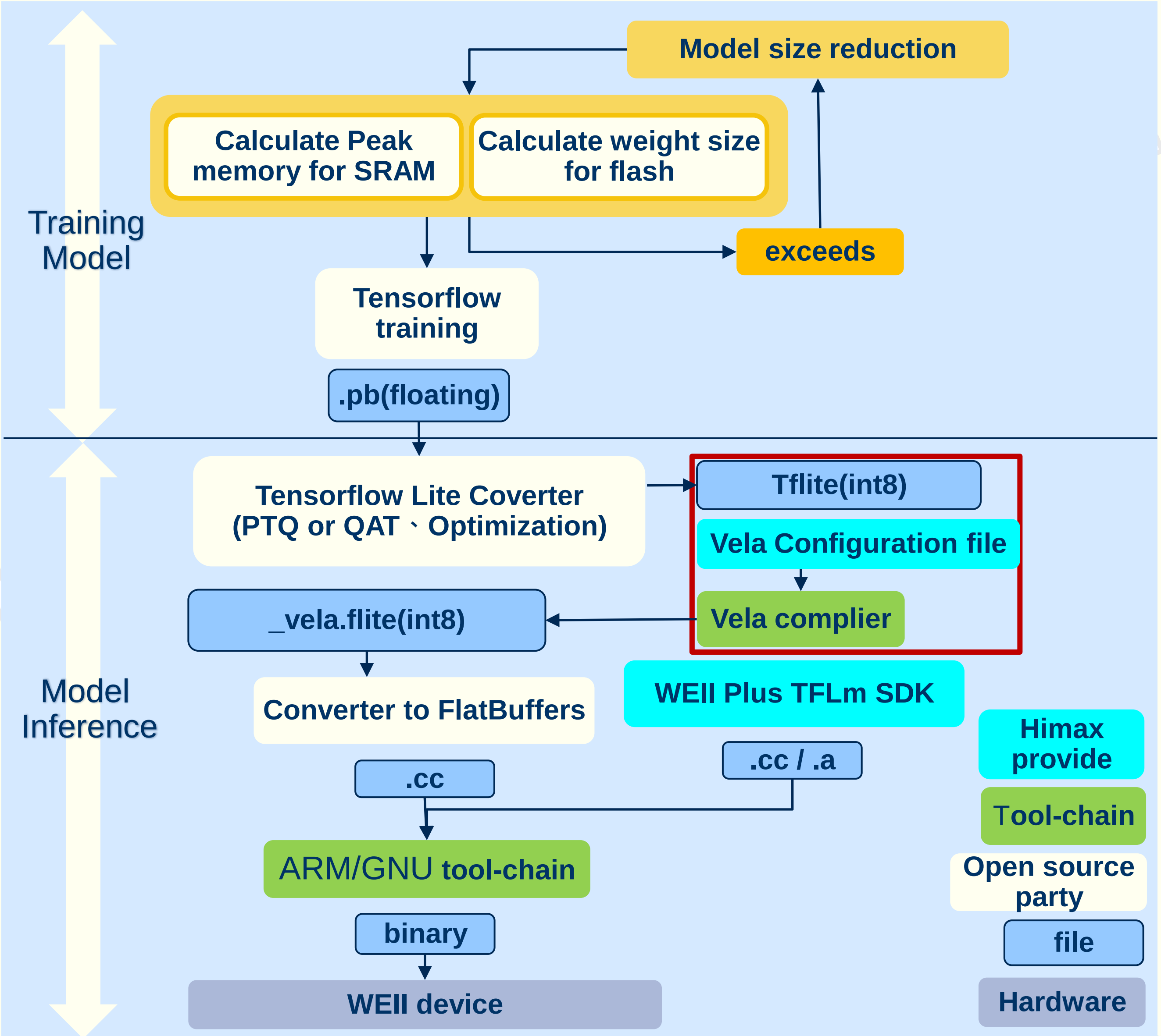
Vela Complier Deploy NN model on WE-II tutorial

Himax Technologies, Inc
奇景光電股份有限公司

Model Deployment Tutorial of Himax WE-II

- Himax WE-II has limited memory and computational power. Various optimizations can be applied to models so that they can be run within these constraints. The limited memory conditions are as follows:
  - Flash ≤ 7M
  - SRAM ≤ 1.8M
- It's recommended that you consider model optimization during your application development process. This document outlines some best practices for optimizing TensorFlow models for deployment to Himax WE-II device.
- TensorFlow Lite for Microcontrollers(TFLμ,TFLm) is a port of TensorFlow Lite(TFlite) aimed at microcontrollers and other devices with only kilobytes of memory.

Model Deployment Workflow of Himax WE-II

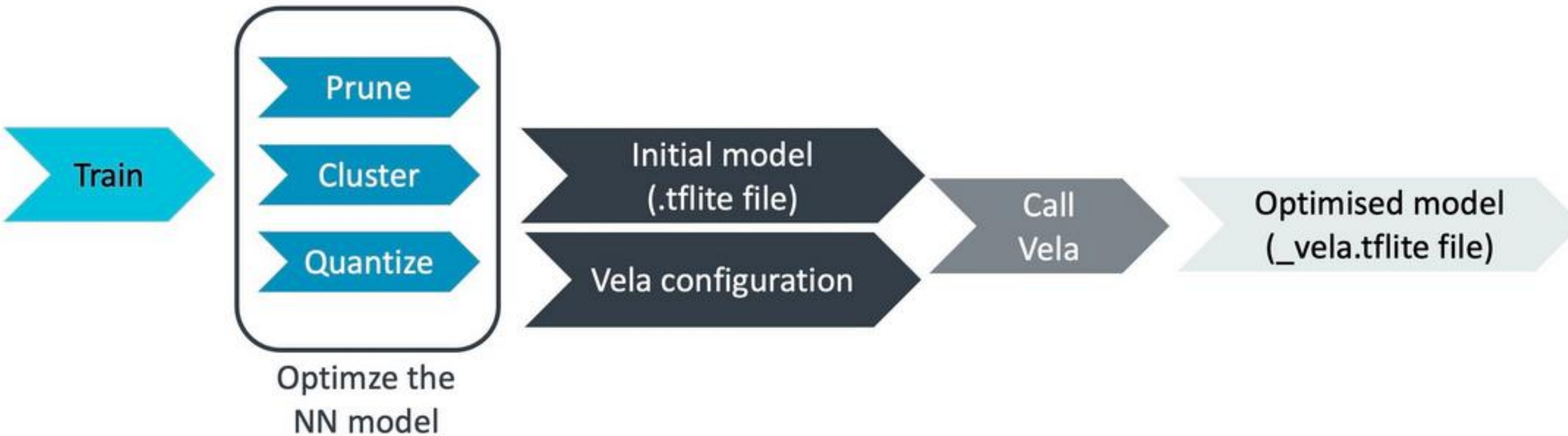


Tensorflow Lite Coverter

- Optimize a ML model for fast inference on Ethos-U microNPU
  - The purpose of Tensorflow Lite Coverter is to convert the model to standard FlatBuffers format with TensorFlow Lite converter.
    - Quantization
      - Post-training quantization (PTQ)
      - Quantization-aware training (QAT)

Vela Complier

- To deploy your neural network (NN) model on an embedded system containing an Ethos-U NPU , the first step you need to do is use open-source Python tool Vela to compile your prepared model.
    - `pip3 install ethos-u-vela`



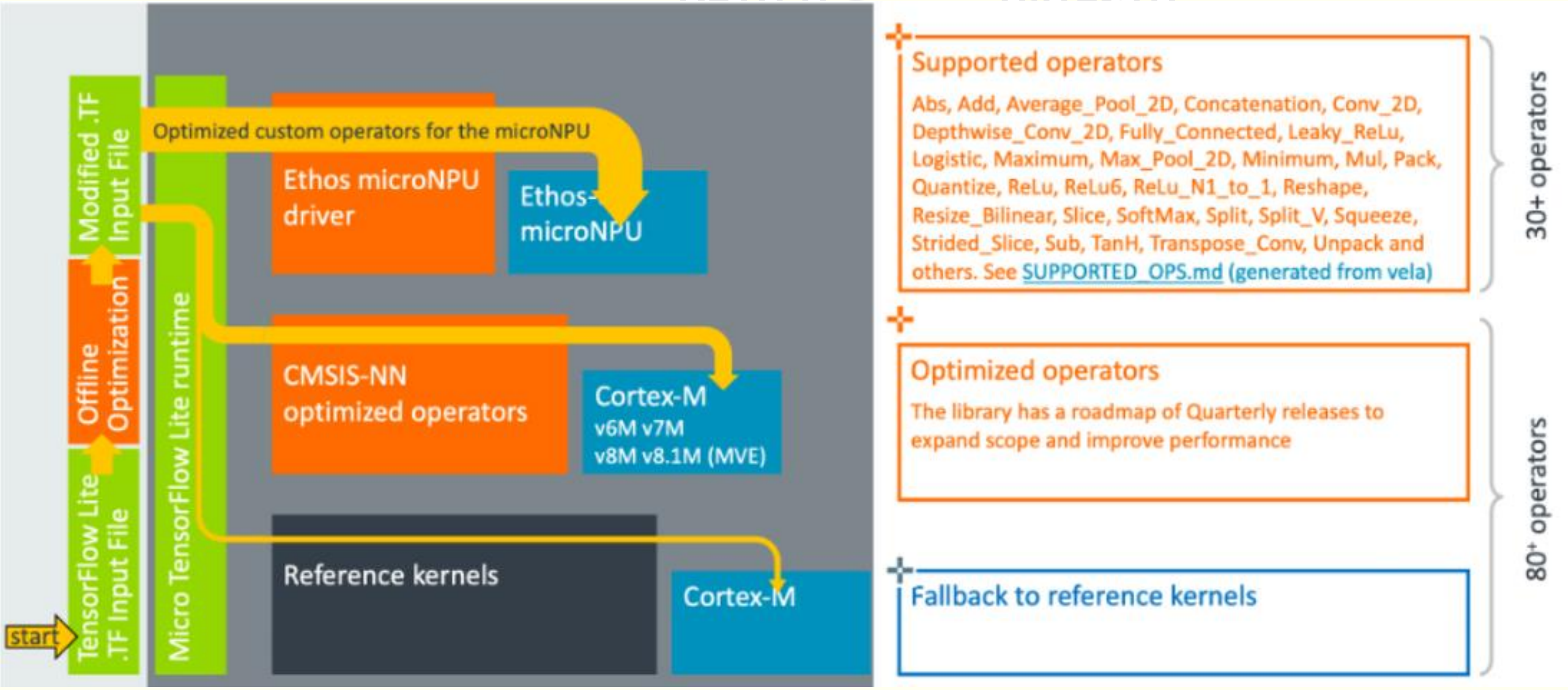
Generic workflow

Vela Complier : command-line interfaces (CLI)

- Vela provides users with lots of command-line interfaces (CLI) to configure each specific calling process. The verbose and detailed description can be found in "Vela Options" session.
  - If you do not specify these parameters additionally, it runs under the internal default values which are version-specific. Refer to each version's "Vela Options" documentation to find out the default value of each parameter.

Vela Complier : Vela's support operators

- After compilation, TensorFlow Lite custom operators for those parts of the model that can be accelerated by the Ethos-U micro NPU. Parts of the model that cannot be accelerated are left unchanged and will run on the CPU using an appropriate kernel. `vela --supported-ops-report` or latest version SUPPORTED_OPS.md



Vela Complier : Vela's support operators

- Tflite must be quantized to either 8-bit or 16-bit (signed).Arm ML Zoo and TensorFlow Hub can offer you a wide variety of ML models.
- Check Ethos-U micro NPU accelerated falls back operators of your model.
  - `vela yolo_coco.tflite --show-cpu-operations`

Himax WE-II Vela Configuration file

- The Vela configuration file is a Python ConfigParser .ini file format.Consists of 2 sections, System Configuration and Memory Mode.
  - WEII Vela Configuration file: himax_vela.ini

Vela Complier : Generate _vela.tflite

- `Vela yolo_coco.tflite --config himax_vela.ini --accelerator-config ethos-u55-64  --system-config My_Sys_Cfg --memory-mode My_Mem_Mode_Parent --output-dir “.\custom_directory“`
- After using the above command to call Vela seen previously, you will obtain the optimized output model under your specified directory "./output". The output file is in _vela.tflite format. Meanwhile, your computer’s console window will present a log of the Vela compilation process.

Vela Complier : Generate _vela.tflite log

- Please see Vela Performance Estimation Summary for a detailed explanation.

```
Network summary for yolo_coco
Accelerator configuration                    Ethos_U55_64
System configuration                           My_Sys_Cfg
Memory mode                            My_Mem_Mode_Parent
Accelerator clock                                     400 MHz
Design peak SRAM bandwidth                           3.20 GB/s
Design peak Off-chip Flash bandwidth                 0.05 GB/s

Total SRAM used                                    300.97 KiB
Total Off-chip Flash used                          441.17 KiB

CPU operators = 0 (0.0%)
NPU operators = 114 (100.0%)

Average SRAM bandwidth                               0.57 GB/s
Input   SRAM bandwidth                               3.53 MB/batch
Weight  SRAM bandwidth                               1.23 MB/batch
Output  SRAM bandwidth                               2.40 MB/batch
Total   SRAM bandwidth                               7.22 MB/batch
Total   SRAM bandwidth              per input        7.22 MB/inference (batch size 1)

Average Off-chip Flash bandwidth                     0.03 GB/s
Input   Off-chip Flash bandwidth                     0.00 MB/batch
Weight  Off-chip Flash bandwidth                     0.36 MB/batch
Output  Off-chip Flash bandwidth                     0.00 MB/batch
Total   Off-chip Flash bandwidth                     0.36 MB/batch
Total   Off-chip Flash bandwidth    per input        0.36 MB/inference (batch size 1)

Neural network macs                              27225400 MACs/batch
Network Tops/s                                       0.00 Tops/s

NPU cycles                                        5030211 cycles/batch
SRAM Access cycles                                 752672 cycles/batch
DRAM Access cycles                                      0 cycles/batch
On-chip Flash Access cycles                             0 cycles/batch
Off-chip Flash Access cycles                            0 cycles/batch
Total cycles                                      5030211 cycles/batch

Batch Inference time                    12.58 ms,   79.52 inferences/s (batch size 1)
```

Model memory usage

Converter to FlatBuffers

- Different operation system has different convert method.
  - Linux
    - Input following command.
    - `$ xxd –i yolo_coco_vela.tflite yolo_fastest_model_vela.cc`
  - Windows
    - Download package from NLUUG.
    - Place .flite file at same directory as xxd.exe.
    - cd to xxd.exe location.
    - Input following command.
    - `xxd.exe –i yolo_coco_vela.tflite > yolo_fastest_model_vela.cc`

Model Inference: WEII TFLm SDK

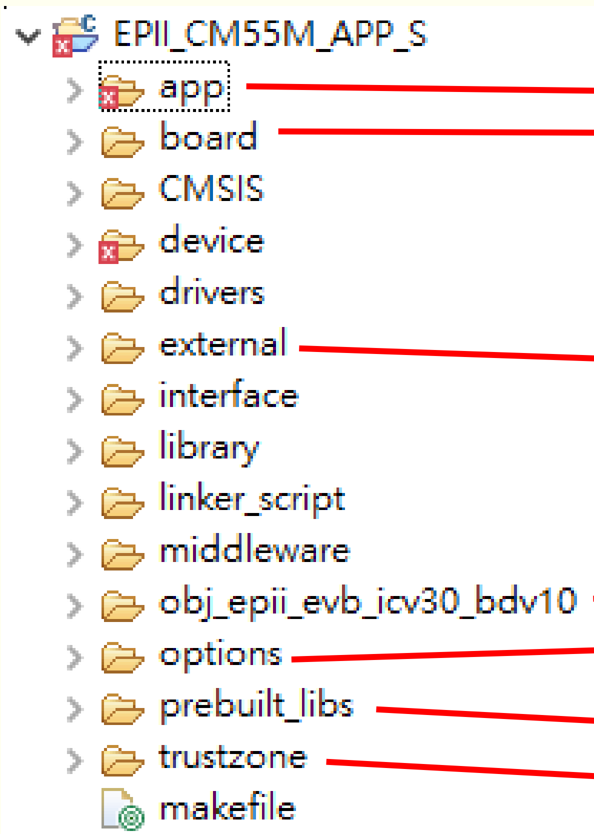
- The purpose is to introduce the WEI TFLm SDK that needs to be used.
  - C byte array with .cc file also needs to be added to this file and add the required related c code .

yolo_coco_vela.cc

```
extern const unsigned char yolo_fastest_tflite[];
extern const int yolo_fastest_tflite_len;
const unsigned char yolo_fastest_tflite[] __attribute__((section(".tflite_model"), aligned(16))) = {
  0x20, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x33, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x12, 0x00, 0x1c, 0x00, 0x18, 0x00, 0x14, 0x00, 0x10, 0x00,
  0x0c, 0x00, 0x08, 0x00, 0x00, 0x00, 0x04, 0x00, 0x12, 0x00, 0x00, 0x00,
```

```
  0x00, 0x00, 0x00, 0x20, 0x07, 0x00, 0x00, 0x00, 0x65, 0x74, 0x68, 0x61,
  0x73, 0x2d, 0x75, 0x00
};
const int yolo_fastest_tflite_len = 452992;
```

Model Inference: WEII TFLm SDK



Scenario APP(Example code)

Board Configurations

External Device (CIS)

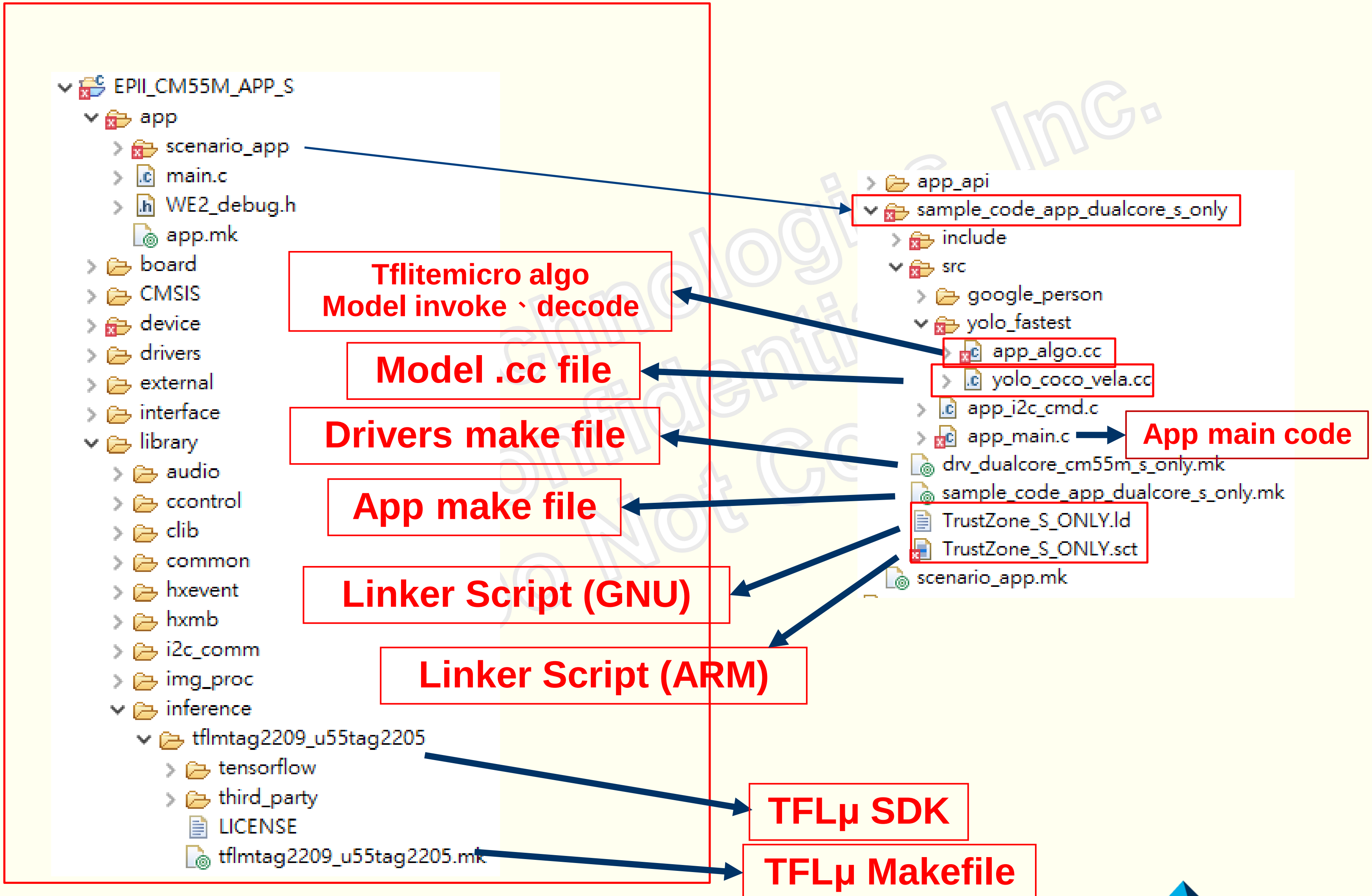
Out Folder

Build File and Build Rule

Prebuilt Lib

ARM TrustZone API

Model Inference: WEII TFLm SDK



Model Inference: GNU tool-chain

- The purpose of GNU tool-chain describe how to build binary code (see WE2_SDK_User_Guide_v05dc_BYD.pdf)and glance at the model invoke inference code.
- About GNU tool-chain:
  - GNU tool-chain enable users to efficiently build, debug, profile and optimize their embedded software applications.
    - You can use the GNU tool-chain for free.

Model Inference: GNU tool-chain

- Tool-chain & IC package select
  - EPII_CM55M_APP_S/makefile

```
# IC package select : LQFP128/WLCSP65
##
IC_PACKAGE_SEL = WLCSP65
```

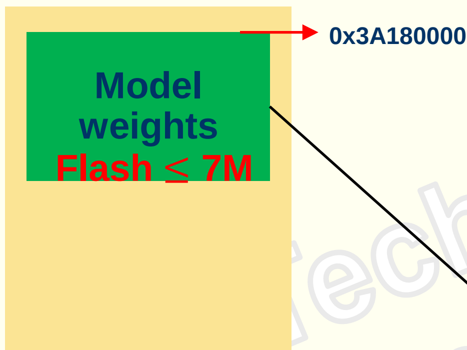
```
##
# Set toolchain
# arm, gnu
##
TOOLCHAIN ?= gnu
```

    - Tool chain can choose arm or gnu depend on your environment.
- After trained model convert into C byte array with .cc filename extension. User can replace the original .cc file or only the array information with below location of WE-II FW.
  - EPII_CM55M_APP_S/app/scenario_app/sample_code_app_dualcore_s_only/src/yolo_fastest/yolo_coco_vela.cc

Model Inference: GNU tool-chain

- Model weights FLASH address

FLASH(0x3A00_0000)



TrustZone_S_ONLY.ld

```
#include "WE2_device_addr.h"

MEMORY
{
  /* Define each memory region */
  CM55M_S_APP_ROM (rx) : ORIGIN = 0x10000000, LENGTH = 0x4
  CM55M_S_APP_DATA (rwx) : ORIGIN = 0x30000000, LENGTH = 0
  CM55M_S_SRAM (rwx) : ORIGIN = BOOT2NDLOADER_BASE, LENGTH
  CM55M_S_APP_FLASH1 (rx) : ORIGIN = 0x3A180000, LENGTH =
}
```

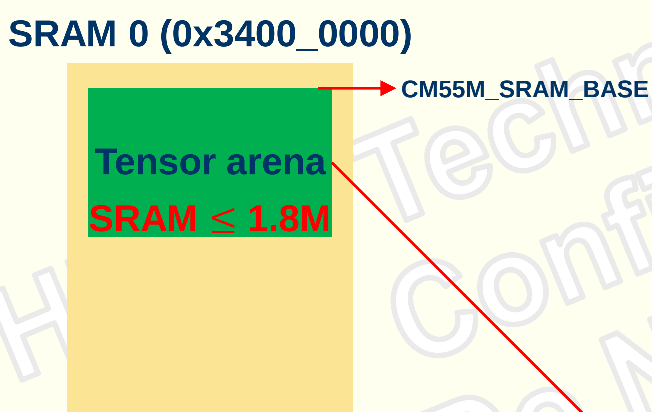
```
#ifdef FLASH_AS_SRAM
#if defined (TFLITE_ALGO_ENABLED)
    .model : ALIGN(4)
    {
        * (.tflite_model)
    } > CM55M_S_APP_FLASH1
#endif
#endif
```

yolo_coco_vela.cc

```
extern const unsigned char yolo_fastest_tflite[];
extern const int yolo_fastest_tflite_len;
const unsigned char yolo_fastest_tflite[] __attribute__((section(".tflite_model"), aligned(16))) = {
  0x20, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x33, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x12, 0x00, 0x1c, 0x00, 0x18, 0x00, 0x14, 0x00, 0x10, 0x00,
```

Model Inference: GNU tool-chain

- Linker script
  - /app/scenario_app/sample_app/TFLM_NoTrustZone.sct
  - Tensor arena SRAM address



TrustZone_S_ONLY.ld

```
#include "WE2_device_addr.h"

MEMORY
{
  /* Define each memory region */
  CM55M_S_APP_ROM (rx) : ORIGIN = 0x10000000, LENGTH = 0x4
  CM55M_S_APP_DATA (rwx) : ORIGIN = 0x30000000, LENGTH = 0
  CM55M_S_SRAM (rwx) : ORIGIN = BOOT2NDLOADER_BASE, LENGTH
  CM55M_S_APP_FLASH1 (rx) : ORIGIN = 0x3A180000, LENGTH =
}
```

```
    .algo : ALIGN(0x100)
    {
      * (.bss.tensor_arena)
    } > CM55M_S_SRAM
```

Model Inference: WEII TFLm SDK

- User can reference the sample code and modify according to different model's output data or additional computation.
- Following will briefly describe the content and modify part of app_algo.cc
- ./app/scenario_app/sample_code_app_singlecore_s_only/src/app_algo.cc
  - TFLm library

```
#include "tensorflow/lite/micro/micro_mutable_op_resolver.h"
#include "tensorflow/lite/micro/micro_interpreter.h"
#include "tensorflow/lite/schema/schema_generated.h"
#include "tensorflow/lite/c/common.h"
#include "tensorflow/lite/micro/micro_error_reporter.h"
```

  - `micro_interpreter.h` contains code for loading and running model.

Model Inference: GNU tool-chain

- The YOLO fastest model requires Peak memory(KB)+Tail usage of tensor arena to store all runtime resources. Based on experience, we set the security range to within 320KB.

```
constexpr int tensor_arena_size = 320*1024;
```

```
uint8_t tensor_arena[tensor_arena_size] __attribute__ ((section(".bss.tensor_arena"), aligned(16)));
```

- Tail usage contains model operator API other need usage. If you want to Minimize tensor arena consumption, you must use this function to output the actual Tail usage after the API function "interpreter->AllocateTensors()", We actually used Bytes in the model.

```
interpreter->arena_used_bytes();
```

Model Inference: GNU tool-chain

- `int32_t app_algo_init()`
  - The initial function which set up basic API which is needed by trained model's algorithm.
    1. Loading trained model

       ```
       model = ::tflite::GetModel(yolo_fastest_tflite);
       ```

    2. Adding operation resolver. (Depend on trained model algorithm)

       ```
       static tflite::MicroMutableOpResolver<1> micro_op_resolver;
       micro_op_resolver.AddEthosU();
       ```

    3. Build an interpreter to run the model with.

       ```
       static tflite::MicroInterpreter static_interpreter(model,
       micro_op_resolver,tensor_arena, tensor_arena_size,error_reporter);
       ```

    4. Allocate runtime resources.

       ```
       interpreter->AllocateTensors()
       ```

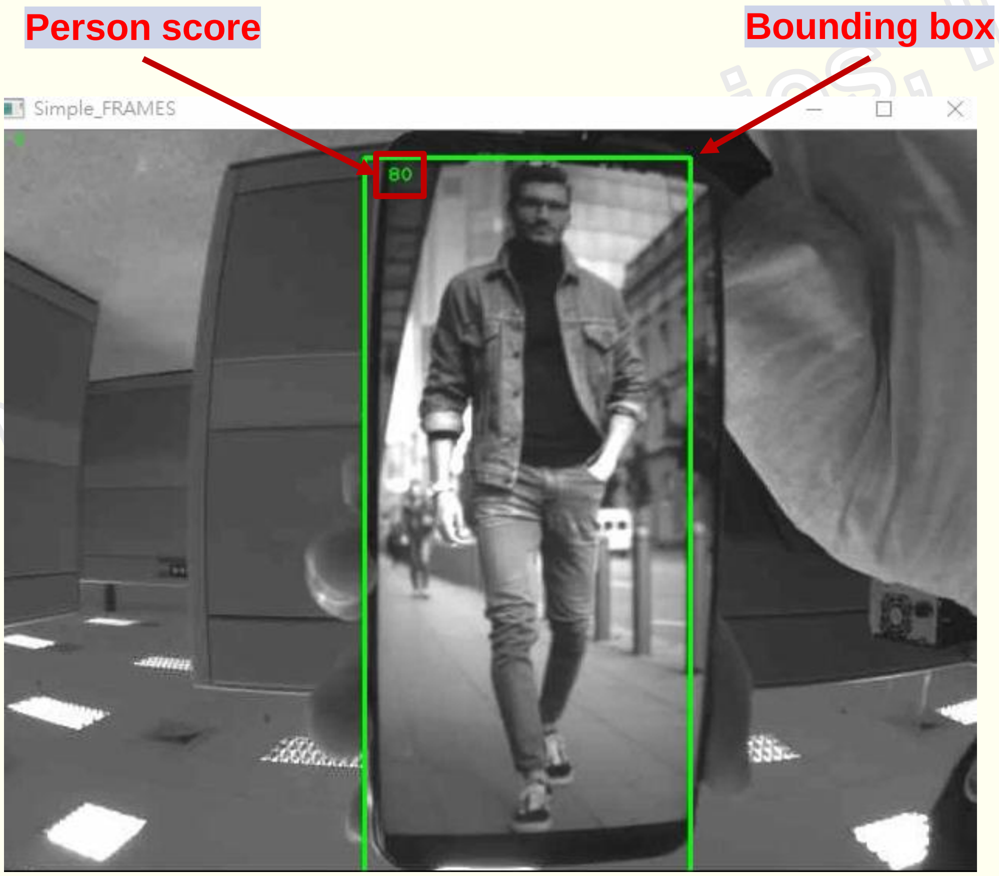
Model Inference: GNU tool-chain

- `static network model_inference(tflite::MicroInterpreter* static_interpreter, uint8_t* input_img)`
  - The main function for microcontroller to operate the trained model invoke.
    1. Load input image and rescale if needed.
       ```
       TfLiteStatus invoke_status = static_interpreter->Invoke();
       ```
    2. Run the model inference.

- `static void model_inference_post_processing(network* net, ALGO_RESULT *algoresult)`
  - Get output result data after decode proceed. User can build your computation based on the result of output data.

Model Inference result

- YOLO fastest person detection result



Drive for better vision