TENSAI FLOW QUICK START GUIDE

Tensai Flow 1.0 Beta Release

Oct 26, 2021

ETA COMPUTE
340 N Westlake Blvd. Suite 115, Westlake Village, California 91362

Table of Contents

1. Setting up Development Environment

Extract and copy TENSAI_FLOW SDK into Ubuntu machine.
To install prerequisites for using TENSAI_FLOW release run install.sh script inside TENSAI_FLOW/Tools/install/ folder

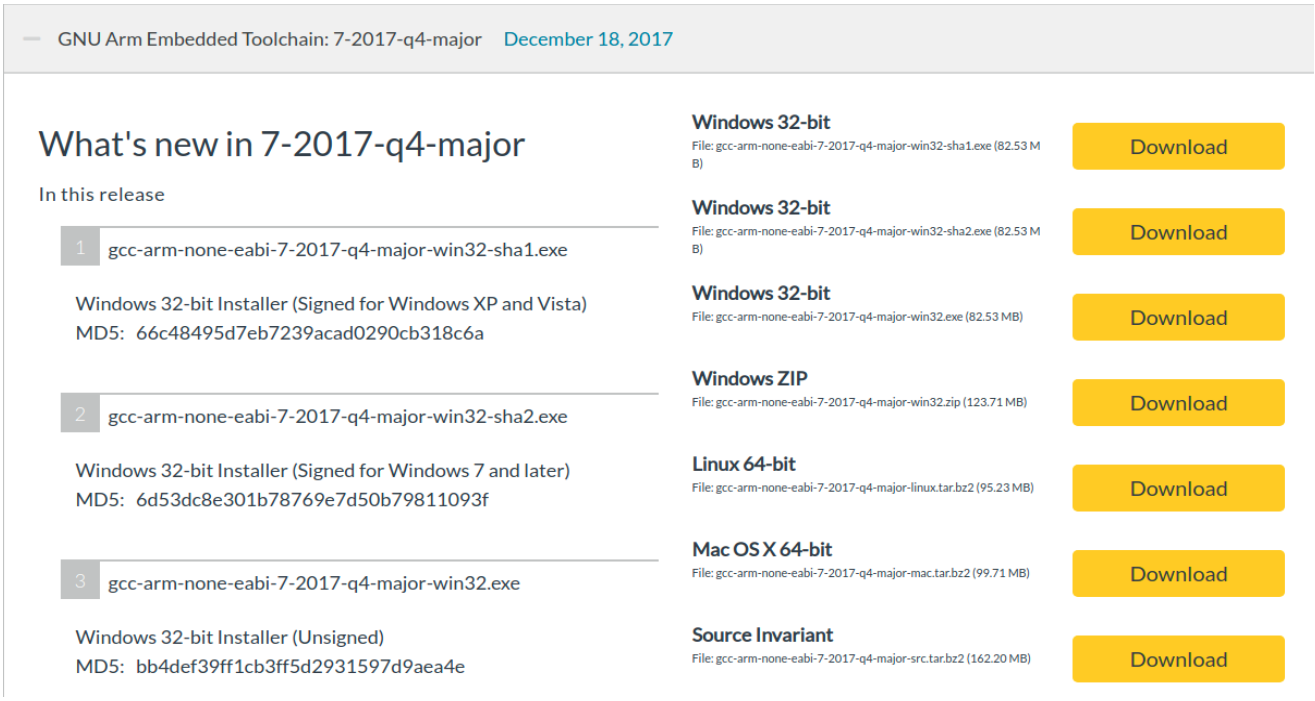
```
sh install.sh
```

In Ubuntu version 18.04, the `echo` is utility is located only at `/bin` so run the below command to make a softlink under `/usr/bin`

```
sudo ln -s /bin/echo /usr/bin/echo
```

1.1 arm gcc tool chain

Build is tested with arm gcc cross compiler tool chain version gcc-arm-none-eabi-7-2017-q4-major, which can be downloaded from below mentioned link.
https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu- toolchain/gnu-rm/downloads
Scroll down the page to 'GNU Arm Embedded Toolchain: 7-2017-q4-major' and download the Linux 64-bit version as shown below.



Extract the downloaded tarball to `/opt`:

```
sudo tar -xjf gcc-arm-none-eabi-7-2017-q4-major-linux.tar.bz2 -C /opt/
```

Edit the ~/.bashrc file. Add below lines at the end of bashrc so that the PATH variable includes the directory to the compiler

```
export PATH=/opt/gcc-arm-none-eabi-7-2017-q4-major/bin:$PATH
```

Restart the terminal after this for the changes to take effect

2. Build and Run Applications using the command line.

Before building the firmware binary (section 2.4) for "compiler-test-app" application, we need to generate an infer.c file and two additional files, "input_data.h" and "data_io.c". "input_data.h" contains sample input and expected output for testing inferencing from stored images. "data_io.c" has functions for reading input images from the camera and it will also have a post processing function. Refer to sections 2.1, 2.2 and 2.3 for generating and storing these files in appropriate paths. Refer to section 2.7 for menuconfig option selection for doing inference from either stored image or from camera captured image.
Follow the "Example" section in the compiler user guide to compile tflite file to infer.c file
`TENSAI_FLOW/Tools/Tensai_compiler/doc/TensaiCompiler_UserGuide.pdf`

2.1 Generating c model files from compiler tools for Fashion MNIST model

```
/* generate infer.c for Fashion MNIST model */
<Tools/Tensai_compiler/bin> ./tensaicc \
--model=../model_zoo/fashion_mnist_model/model.tflite \
--app_dir=../../../Applications/compiler-test-app/ \
--target_config=core_config.ini --nodata_io

<Tools/Tensai_compiler/bin> cp ../model_zoo/fashion_mnist_model/data_io.c
../../../Applications/compiler-test-app/src/

<Tools/Tensai_compiler/bin> cp ../model_zoo/fashion_mnist_model/input_data.h
../../../Applications/compiler-test-app/include/
```

2.2 Generating c model files from compiler tools for MNIST model.

```
/* generate infer.c for MNIST model */
<Tools/Tensai_compiler/bin> ./tensaicc \
--model=../model_zoo/mnist_model/model.tflite \
--app_dir=../../../Applications/compiler-test-app/ \
--target_config=core_config.ini --nodata_io

<Tools/Tensai_compiler/bin> cp ../model_zoo/mnist_model/data_io.c
../../../Applications/compiler-test-app/src/

<Tools/Tensai_compiler/bin> cp ../model_zoo/mnist_model/input_data.h
../../../Applications/compiler-test-app/include/
```

Refer below document for using this model with camera
`TENSAI_FLOW/Docs/Running_MNIST_From_Camera.pdf`

2.3 Generating c model files from compiler tools for Synaptic's model.

```
/* generate infer.c for Synaptic's model */
<Tools/Tensai_compiler/bin> ./tensaicc \
```

```
--model=../model_zoo/synaptics/model_mobilenet224_v2_0.01_exp_3.tflite \
--app_dir=../../../Applications/compiler-test-app/ \
--target_config=core_config.ini --nodata_io

<Tools/Tensai_compiler/bin> cp ../model_zoo/synaptics/data_io.c
../../../Applications/compiler-test-app/src/

<Tools/Tensai_compiler/bin> cp ../model_zoo/synaptics/input_data.h
../../../Applications/compiler-test-app/include/
```

2.4 Building firmware binary image

Applications are located inside `TENSAI_FLOW/Applications/` directory, and the build needs to be triggered from the build directory inside the application. Below mentioned command shows how to build "compiler-test-app" application.

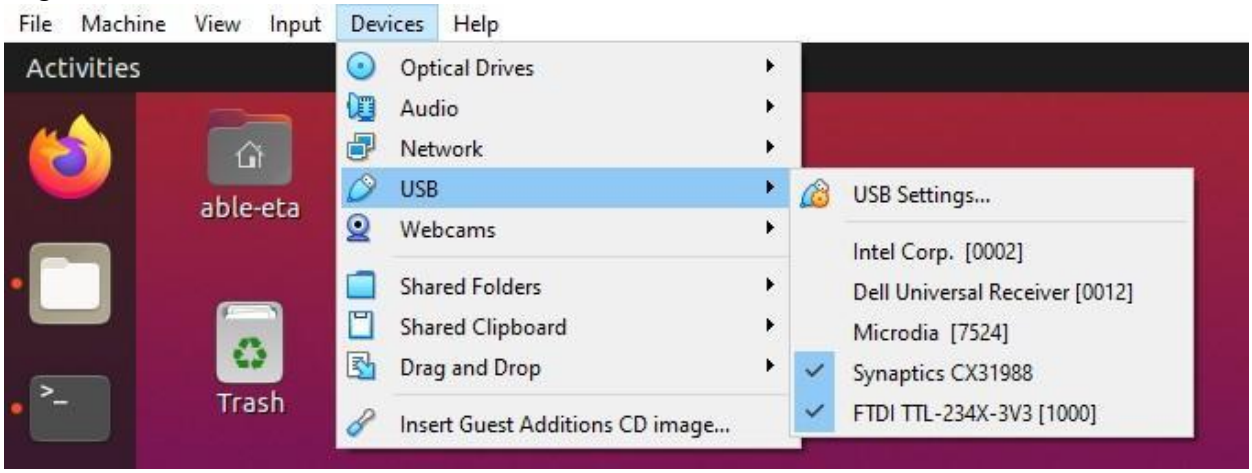
```
/* Invoke CMake to generate Makefile and configure default
  configurations*/
<TENSAI_FLOW/Applications/compiler-test-app/build> cmake ..

<TENSAI_FLOW/Applications/compiler-test-app/build> make app
```

Above steps will build the target application with default configuration. In default configurations all supported features are not enabled, if the application depends on a feature other than default, that feature needs to be enabled using steps mentioned in section 2.7.

2.5 Running Application

If using a virtual machine, then make sure to select the Synaptics CX31988 and Debug UART as given below.



Follow the below command to load an application.

```
/* load program */
<TENSAI_FLOW/Applications/compiler-test-app/build># make flash
```

2.6 Clean build

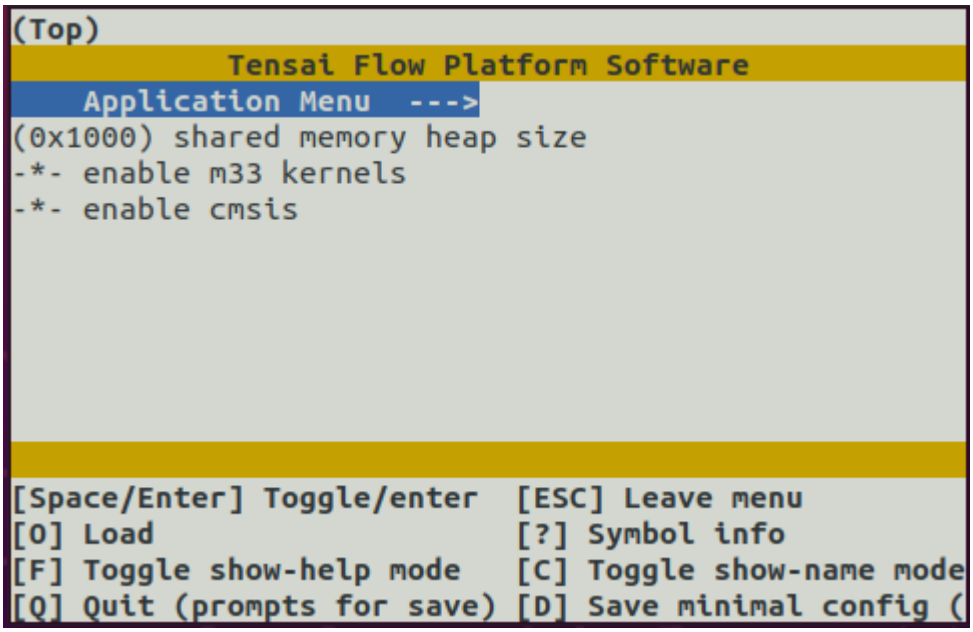
Follow the below mentioned command to clean an application.

```
/* load program */
<TENSAI_FLOW/Applications/compiler-test-app/build># make app_clean
```

2.7 Menuconfig configuration

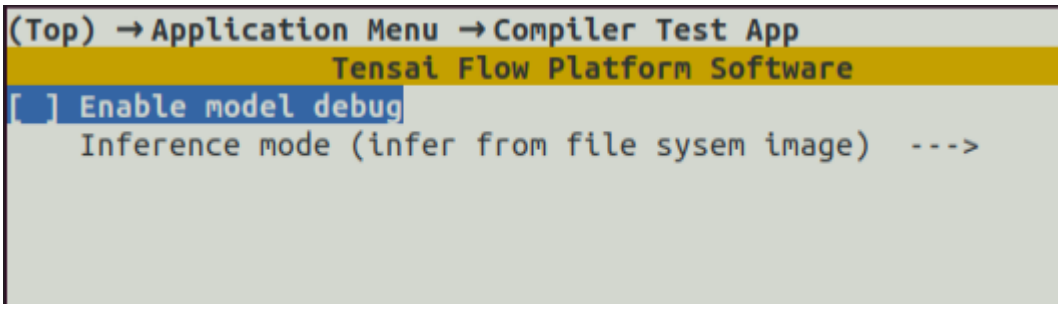
Run below command to configure the application

```
/* menuconfig */
<TENSAI_FLOW/Applications/compiler-test-app/build># make menuconfig
```



Application specific menuconfig

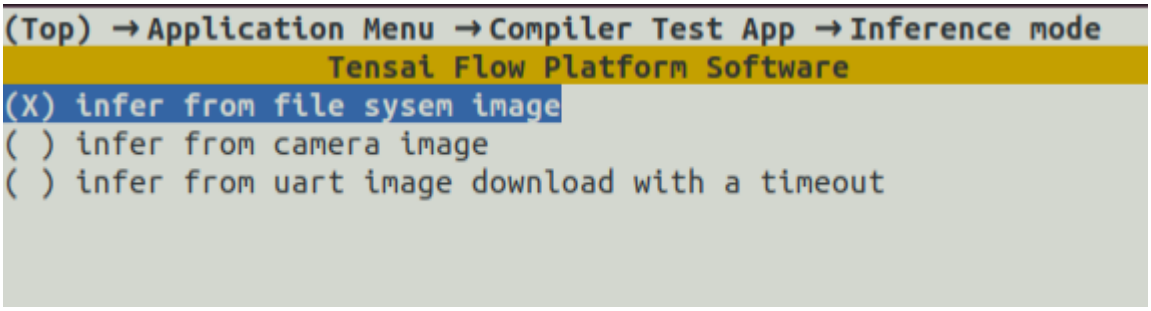
Select “Application Menu => Compiler Test App” to configure application specific configs



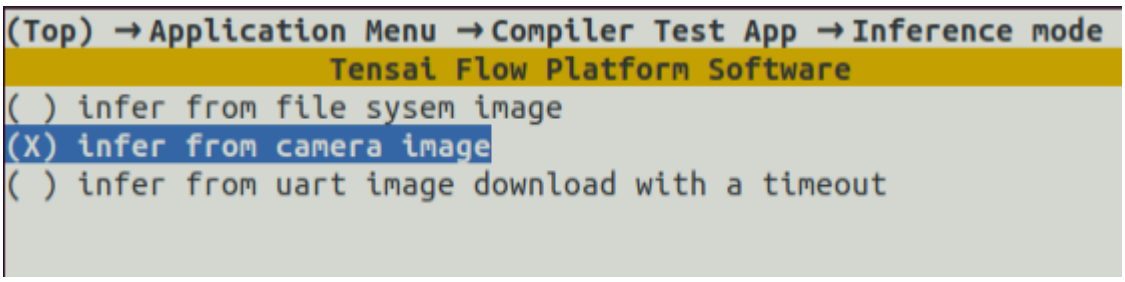
Enable model debug

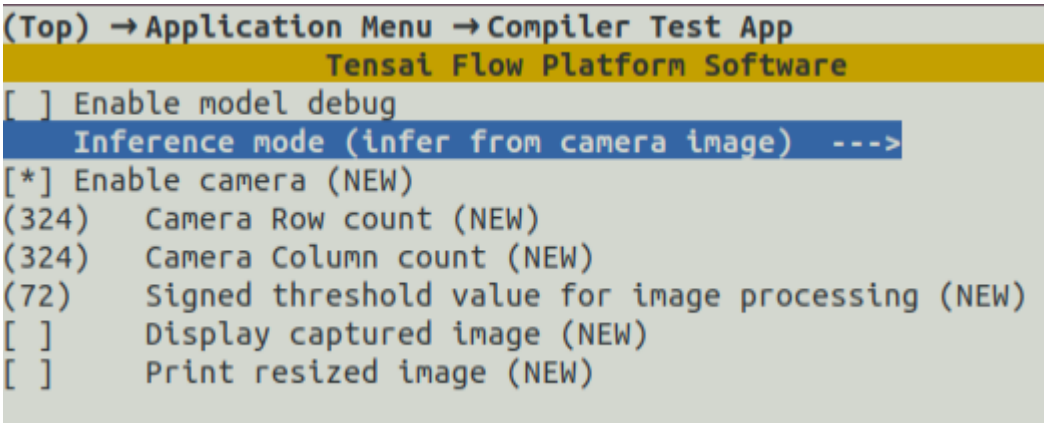
- This config controls the debug features like per layer inference time and buffer dump

Inference mode



Enable Camera: For enabling or disabling the camera.
For accessing this config Inference mode needs to be “infer from camera image





When Camera is enabled, following additional camera related settings can be selected:

- Camera Row count and Camera Column count is captured image resolution.
- Signed threshold value for image processing for removing the background from the image. Range -128 to 127.
  0 **:** -128 <= p < threshold
  p **:** threshold <= p < 127
- Display captured image: Choose this option for displaying captured image using parser script provided in the SDK in below path:
  Thirdparty/SDK/<Release SDK name>/fcp/platform/tahiti/tools

```
python parser.py -c <serial_device> -b 115200 -f <filename>

e.g python parser.py -c /dev/ttyUSB0 -b 115200 -f test
```

- Print resize image: Choose this option for printing resized and thresholded image array over debug UART. Note: Inference will be done on this image.

Shared memory size

This is for indicative purpose and should not be changed

3. Serial port debug

Open the debug UART using minicom or any other similar serial port viewer with 1152000 8N1 configuration.

```
minicom -D /dev/ttyUSB0
```

Below print will confirm if the download is proper or not for the MNIST model.

```
---------------
Begin infer test
Inference Results :
Max Diff: 0, avg Diff: 0, nDiff = 0/10
Inference output for mnist : TWO

total infer time in 35 msec, in cycles 858921
total time (file read + infer) in 43 msec, in cycles 1058228
---------------
```

4. Using a new model file

Generate the c model using the tensaicc compiler. The input and output of the compiler may vary based on models. Take data_io.c file present in mnist model as a reference to see how to use camera / static images from an array. Alternatively, passing the --data_io flag will generate a stock template prefilled with a single input to get data and compare as many outputs to the expected outputs in the post-process function.

file : data_io.c

```
34  void get_data(int8_t *in_buf_0, uint16_t in_buf_0_dim_0, uint16_t in_buf_0_dim_1, uint16_t in_buf_0_dim_2)
35  {
36  #ifdef CONFIG_INFER_FROM_CAMERA
37    captureAndProcessImage(in_buf_0, in_buf_0_dim_0, in_buf_0_dim_1);
38  #else
39    readFromFileSystem(FILE_ID(pIn), in_buf_0, ALIGN_DOWN_TO_4(IN_SIZE));
40  #endif
41  }
```

The get_data function will be called from the infer.c file. The user can preprocess the data and pass it onto the in_buf_0 pointer in the data_io.c file. Use above as reference on how to capture image from camera and on how to read the static image from the input_data.h file. In this example pIn is the static image byte array.

The name of the file will be of format FILE_ID(variable_name). In this example to access variable pIn inside input_data.h use FILE_ID(pIn)

file : input_data.h

```
1  #define IN_SIZE 784
2
3
4  static const q7_t __SYSRODATA__ pIn[IN_SIZE]  ALIGN_TO(4) = {
5  -128, -128, -128, -128, -128, -128, -128, -128, -128, -128,
6  };
```

For static images user has to generate the input_data.h by referring to below document
`TENSAI_FLOW/Tools/Tensai_compiler/doc/TensaiCompiler_UserGuide.pdf`

Please note that the input_data.h file should not be manually modified.
In case of static images, to access the expected output variables inside input_data.h use below format

```
int8_t* pExpect = allocMem(SHM,ALIGN_UP_TO_4(OUT_SIZE0));
if(pExpect == NULL)
  return;

readFromFileSystem(FILE_ID(pExpect0), pExpect, ALIGN_UP_TO_4(OUT_SIZE0));
```

Use below example input_data.h file for reference

```
#define OUT_SIZE0 10

static const q7_t __SYSRODATA__ pExpect0[OUT_SIZE0]  ALIGN_TO(4) = {
-128, -66, 47, -120, -124, -128, -128, -122, -127, -127
};
```

Input_data.h file will be modified by scripts to move the data into the file system to save memory as shown below.

```
#define IN_SIZE 784

#ifdef CONVERT_2_BIN
static const q7_t __SYSRODATA__ pIn[IN_SIZE]  ALIGN_TO(4) = {
-128, -128, -128, -128, -128, -128, -128, -128, -128, -128, -128,
};
#else
#define pIn_fileId _ID('A', 'A', 'A', 'A' )
#endif

#define OUT_SIZE0 10

#ifdef CONVERT_2_BIN
static const q7_t __SYSRODATA__ pExpect0[OUT_SIZE0]  ALIGN_TO(4) = {
28, 77, 5, 59, 24, 9, 32, 1, 35, 1
};
#else
#define pExpect0_fileId _ID('B', 'B', 'B', 'B' )
#endif
```

Build and run the application as mentioned in this document.

5. Building M33/CAPE/LLE binaries

Applications are located inside `TENSAI_FLOW/Applications/` directory, and the build needs to be triggered from the build directory inside any application. Below mentioned command shows how to build M33/cape/lle binaries.

Prerequisite: Needs chesscc compiler for building cape/lle binaries. Also chesscc compiler path has to be exported to environment variables using source command.

```
/* Invoke CMake to generate Makefile and configure default
  configurations*/
```

```
<TENSAI_FLOW/Applications/compiler-test-app/build> cmake ..

<TENSAI_FLOW/Applications/compiler-test-app/build> make sdk
```

Above steps will build binaries for all the cores viz M33, cape2a, cape2b, llea and lleb. Also it will assemble all the binaries and make the final image.

Building binaries for individual cores is also supported.

```
/* Very first time only once we have to execute make sdk command.
  It will build all the binaries and final image */
<TENSAI_FLOW/Applications/compiler-test-app/build> make sdk


/* To generate llea binary only */
<TENSAI_FLOW/Applications/compiler-test-app/build> make llea

/* To generate lleb binary only */
<TENSAI_FLOW/Applications/compiler-test-app/build> make lleb


/* To generate cape2a binary only */
<TENSAI_FLOW/Applications/compiler-test-app/build> make cape2a

/* To generate M33 binary only */
<TENSAI_FLOW/Applications/compiler-test-app/build> make mcusdk

/* When any of binaries are newly built, we need to update final
  image */
<TENSAI_FLOW/Applications/compiler-test-app/build> make image
```

6. Inference image through UART download

This feature allows the user to define a set of input images in a folder on the host machine, and then automatically step through the image set using each as an input to an inference function running on the Katana evaluation board.
Images have to be kept inside a folder. Formats supported are jpg, jpeg, png formats. The script will convert all the images into a bin format which will be downloaded to the board one by one. The image name will be printed along with the respective inferencing output for each image. Once all the images are inferred, the script will exit.
After flashing binary, Katana will wait for a timeout for the image to be downloaded, after

timeout Katana will infer on the inbuilt image the application was built with. After inferencing Katana will again wait for the image download with timeout.

Run below commands to install the dependencies

```
pip install pyserial
pip install numpy
sudo chmod 666 <COM_PORT>        e.g. sudo chmod 666 /dev/ttyUSB0
pip install tensorflow==2.4.1
```

Refer data_io.c file inside Tools/Tensai_compiler/model_zoo/mnist_model/ folder to get details on the API used to infer from UART download.

```
#define ERR_LIST_SZ 10
void get_data(int8_t *in_buf_0, uint16_t in_buf_0_dim_0, uint16_t in_buf_0_dim_1, uint16_t in_buf_0_dim_2)
{
#ifdef CONFIG_INFER_FROM_CAMERA
  captureAndProcessImage(in_buf_0, in_buf_0_dim_0, in_buf_0_dim_1);
#elif CONFIG_INFER_FROM_FILESYSTEM
  readFromFileSystem(FILE_ID(pIn), in_buf_0, ALIGN_DOWN_TO_4(IN_SIZE));
#elif CONFIG_INFER_FROM_DOWNLOAD
  bool status = inferFromSerialDownloadImage(FILE_ID(pIn), in_buf_0, IN_SIZE);
  if(status == false)
  {
    readFromFileSystem(FILE_ID(pIn), in_buf_0, ALIGN_DOWN_TO_4(IN_SIZE));
  }
#else
  #error "INVALID CHOICE"
#endif
}
```

Enable infer from uart download by using make menuconfig

```
(Top) → Application Menu → Compiler Test App → Inference mode

( ) infer from file sysem image
( ) infer from camera image
(X) infer from uart image download with a timeout
```

Build and flash the application, then run the below script.
The model file has to match with the model with which the application was built and the COM_PORT must not be used by other utilities like Minicom or Putty before running this tool.

```
cd Tools/image_converter/
python3 img_update.py -m <MODEL_FILE> -a ../array2bin/array2bin.c -i
<TEST_IMAGES_FOLDER> -d <COM_PORT>
```

```
e.g. python3 img_update.py -m
../Tensai_compiler/model_zoo/mnist_model/model.tflite -a
../array2bin/array2bin.c -i ../Tensai_compiler/model_zoo/mnist_model/ -d
/dev/ttyUSB0
```

Refer below screenshot which shows a sample output of the tool.

```
/home/mohammed/Release/TENSAI_FLOW-Tensai_beta_rc_0.4/Tools/Tensai_compiler/model_zoo/mnist_model/bins/two.bin
Waiting for banner
IMAGE_DOWNLOAD_FINISHED
----Outputs ----
[0]:-128
[1]:-66
[2]:47
[3]:-120
[4]:-124
[5]:-128
[6]:-128
[7]:-122
[8]:-127
[9]:-127
Inference output for mnist : TWO
total infer time in 35 msec, in cycles 858313
total time (file read + infer) in 226 msec, in cycles 5562463
Begin infer test
GP:18 BUSY IO_MUX:0x0
IMAGE_DOWNLOAD_WINDOW_OPEN



/home/mohammed/Release/TENSAI_FLOW-Tensai_beta_rc_0.4/Tools/Tensai_compiler/model_zoo/mnist_model/bins/zero.bin
Waiting for banner
IMAGE_DOWNLOAD_FINISHED
----Outputs ----
[0]:114
[1]:-128
```