Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für Integrierte Systeme
Integrated Systems Laboratory

Department of Information Technology and Electrical Engineering

Hands-On Workshop

Real-time on-device image classification on microcontrollers

Michele Magno, PhD
Viviane Potocnik
Marco Giordano

Thursday 22$^{\text{nd}}$ August, 2024

1 Introduction

In this exercise, you will learn how to use the *TFLite* toolchain to execute TensorFlow Lite models on different Microcontroller (MCU) targets. We will use the `B-L475E-IOT01A2` board and perform real-time inference on the MCU. This is done using a simple *Python* script that establishes a Universal Asynchronous Receiver Transmitter (UART) connection to the MCU, sends the input data, and receives the classification result after inference. The data transfer will be done in a static fashion, meaning we will store the data locally and load it into the MCU from a *.npy*-array file. However, for your projects, you could also use the onboard sensors or other inputs, such as your laptop camera, and transmit the data using a serial port to the MCU. To prepare you for your final assignment, we will follow a bottom-up approach to creating a new project that will enable real-time inference on your target architecture.

2 Notation

Student Task: Parts of the exercise that require you to complete a task will be explained in a shaded box like this.

Note: You find notes and remarks in boxes like this one.

3 Preparation

We will use the STM32 CUBE IDE which you are already familiar with in order to port the *TFLite* model on your microcontroller.

It is likely that you will encounter error messages such as `ModuleNotFoundError: No module named 'XYZ'`. This means that the underlying package that implements the module *XYZ* is not yet installed in your current environment. This is different from the error `undefined reference to XYZ`, which is addressed later in this document.

Starting a new project

In the following, we will create a new `STM32 Project` using the STM32 Cube IDE . Figure 1 shows how to start the creation of a new project.

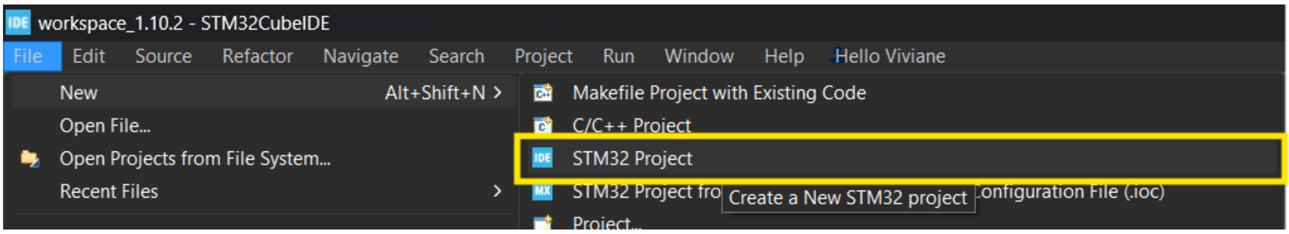
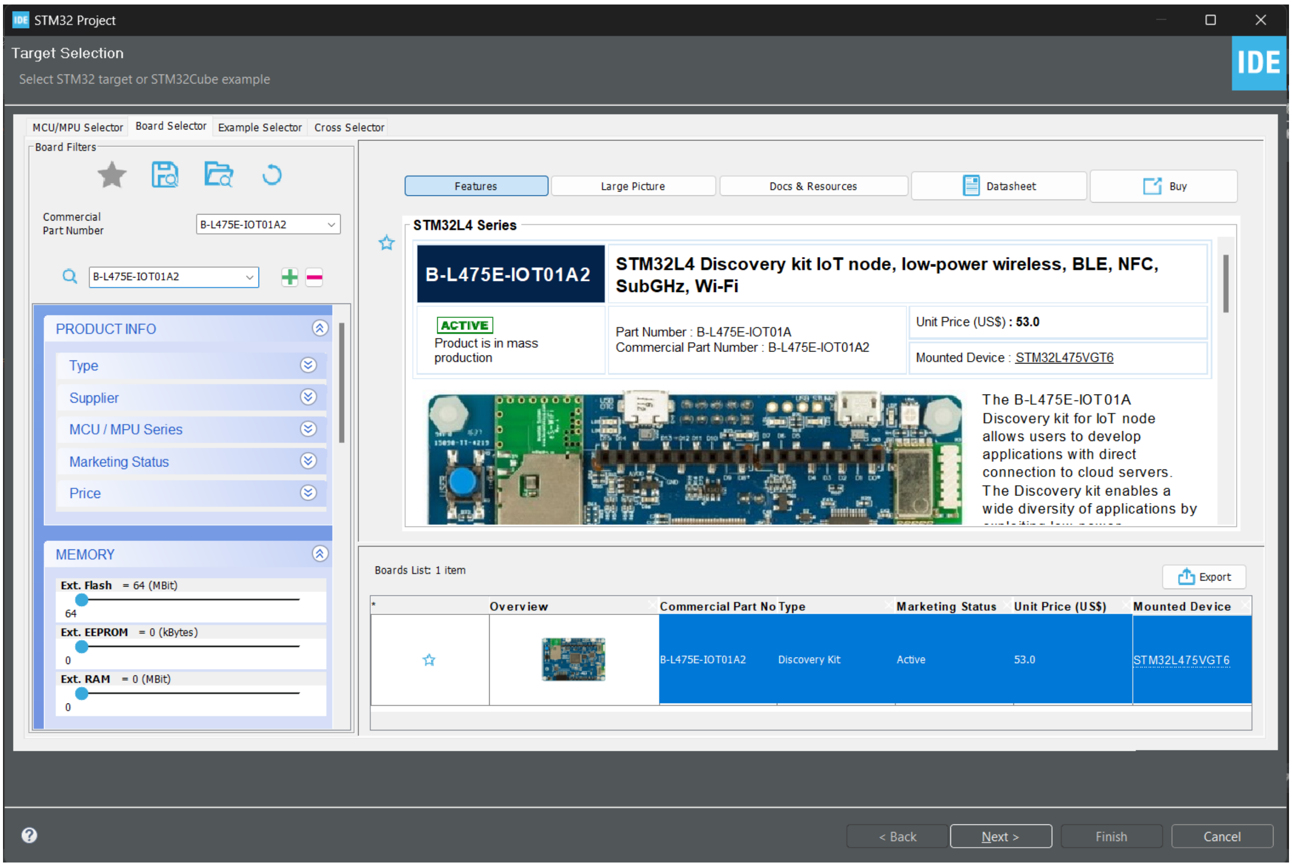


Figure 1: Creating a new `STM32 Project` STM32 Cube IDE project from scratch for image classification on the MCU.

After pressing the `STM32 Project` button, a `Target Selection` window opens. Switch to the `Board Selector` view. Under `Commercial Part Number`, enter the part number of our MCU (`B-L475E-IOT01A2`). Figure 2 shows the window after entering the correct part number. Proceed by clicking `Next`.



Commercial Part Number: B-L475E-IOT01A2

B-L475E-IOT01A2

Ext. Flash = 64 (MBit)

64

Ext. EEPROM = 0 (kBytes)

0

Ext. RAM = 0 (MBit)

0

STM32L4 Series

B-L475E-IOT01A2

STM32L4 Discovery kit IoT node, low-power wireless, BLE, NFC, SubGHz, Wi-Fi

ACTIVE
Product is in mass production

Part Number : B-L475E-IOT01A
Commercial Part Number : B-L475E-IOT01A2

Unit Price (US$) : 53.0

Mounted Device : STM32L475VGT6

The B-L475E-IOT01A Discovery kit for IoT node allows users to develop applications with direct connection to cloud servers. The Discovery kit enables a wide diversity of applications by

Boards List: 1 item

*	Overview	Commercial Part No	Type	Marketing Status	Unit Price (US$)	Mounted Device
☆		B-L475E-IOT01A2	Discovery Kit	Active	53.0	STM32L475VGT6

Figure 2: `Board Selector` in the STM32 Cube IDE with the required configuration for our MCU.

Configuring UART Communication

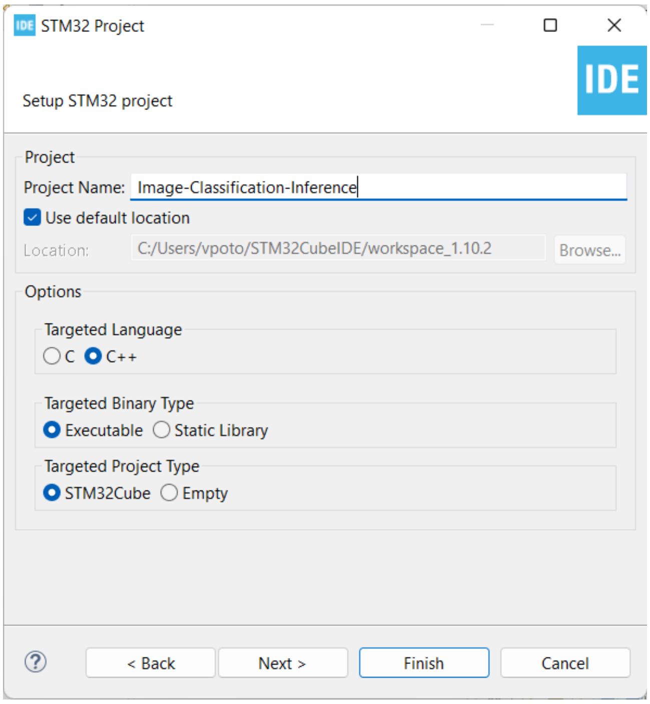
The board includes several peripheral devices such as digital microphones, several communication antennae, and other I/O. In this exercise, you will configure the General Purpose Input/Output (GPIO) pins for communication via UART and profile software implementations with STM32 CUBE IDE .

For communication with the microcontroller, we suggest using TERA TERM or `screen`. However, if you are already experienced with UART communication, feel free to use any serial terminal software you like.

STM CUBE MX provides support for advanced drivers and libraries such as Cube AI and software stacks for Bluetooth or Wi-Fi communication. These software components have to be selected in the `.ioc file` →`Software Packs` →`Manage Software Packs` menu.

Task 1:

1. Create a new project with STM32 CUBE IDE . Make sure to give the project an appropriate name. Furthermore, you have to configure the project as a `C++` project as shown in Figure 3. Select `No` when prompted to initialize the peripherals in default mode, as shown in Figure 4.
2. Configure the USART1 module by navigating to `Connectivity`→`USART1`. Choose the mode to be "Asynchronous" (making it UART1), baud rate $115\,200\,\mathrm{Bits/s}$, word length $8\,\mathrm{Bits}$, no parity bits, and 1 stop bit. Leave all other settings as they are.
3. Make sure the pins PB7, PB6 and PB5 are assigned to RX, TX and CK respectively, as shown in Figure 6. Make sure to clear the pinout assignments beforehand (Figure 5).
4. Save the project to generate the files.
5. Check the files generated in the `Core` folder of the project. Do you see the different files generated for the peripherals?
6. In order to generate a clean `main.c` file, it is recommended to generate separate source and header files for the peripherals and main file. This can be done via the *Project Manager*, as shown in Figure 7.



STM32 Project

Setup STM32 project

Project

Project Name: Image-Classification-Inference

☑ Use default location

Location: C:/Users/vpoto/STM32CubeIDE/workspace_1.10.2

Options

Targeted Language

○ C ● C++

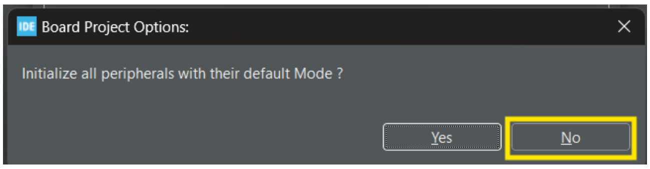
Targeted Binary Type

● Executable ○ Static Library

Targeted Project Type

● STM32Cube ○ Empty

Figure 3: Creating a new C++ STM32 Cube IDE project from scratch for image classification on the MCU.



Board Project Options:

Initialize all peripherals with their default Mode ?

Figure 4: Peripheral prompt window.

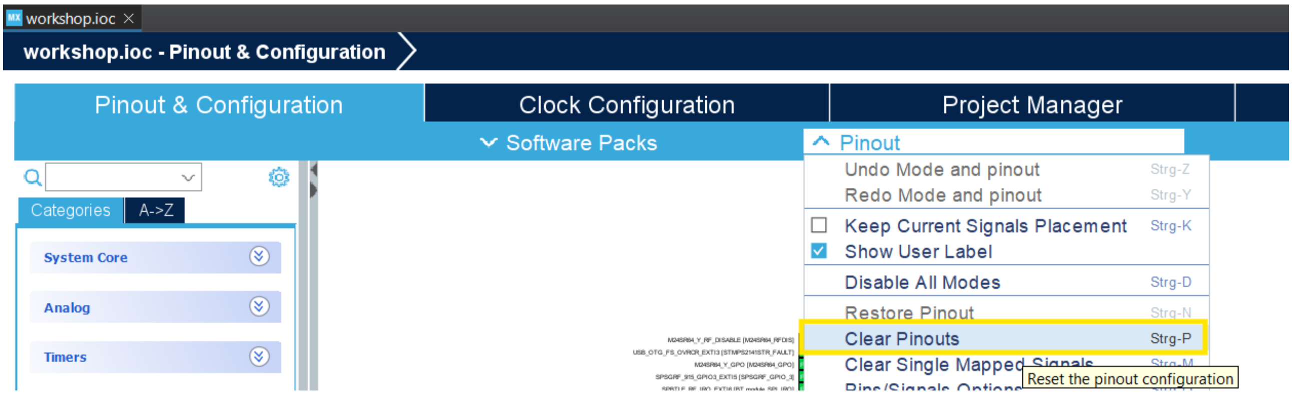
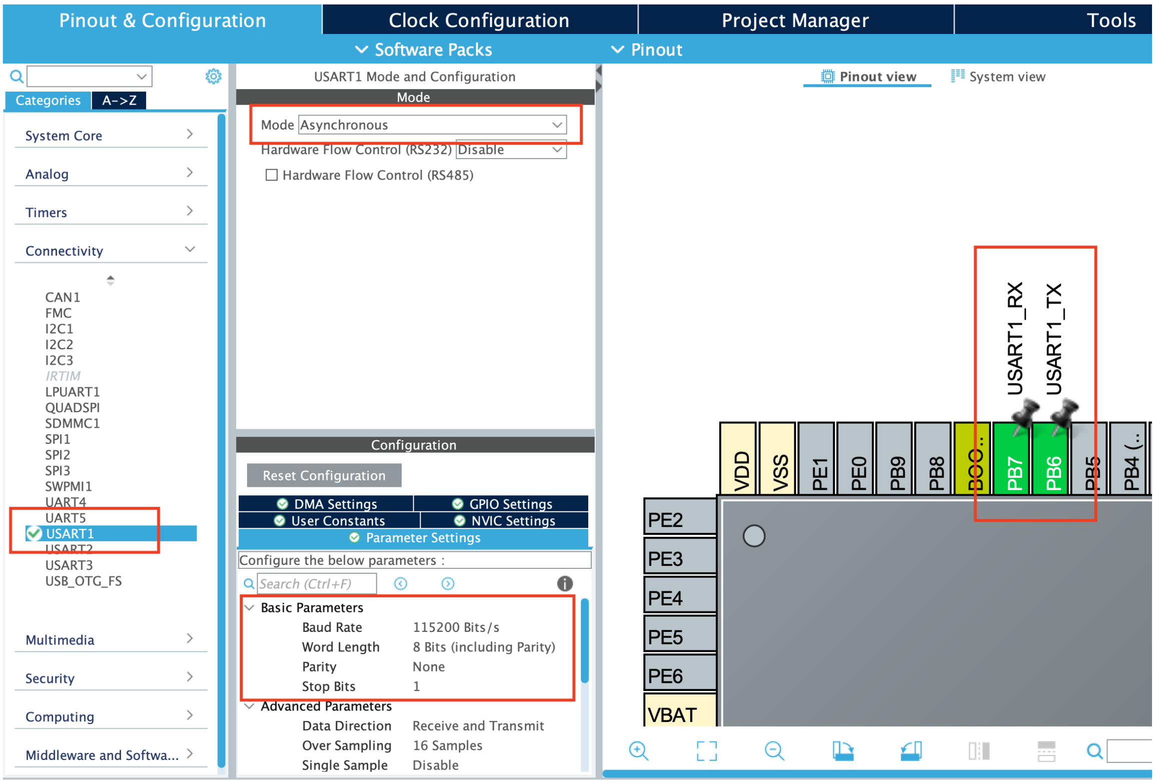


Figure 5: Clear Pinout Assignements.



USART1 Mode and Configuration

Mode

Mode Asynchronous

Hardware Flow Control (RS232) Disable

☐ Hardware Flow Control (RS485)

Configuration

Configure the below parameters :

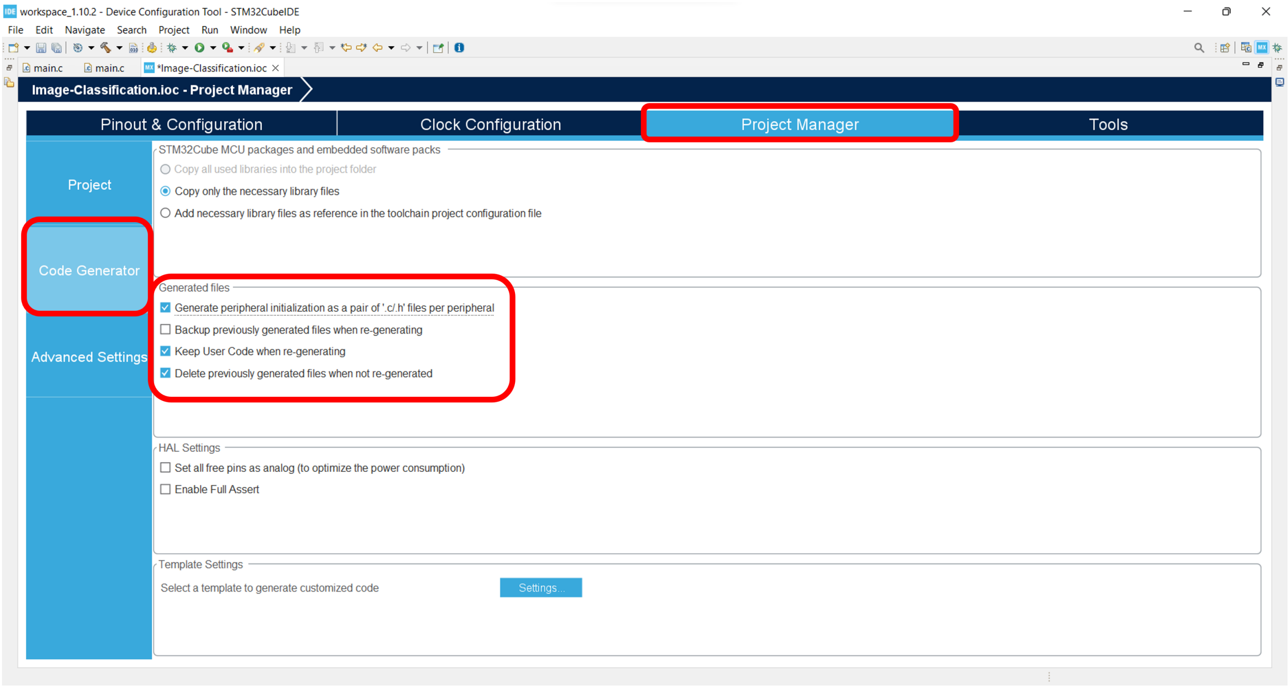
Basic Parameters

Baud Rate	115200 Bits/s
Word Length	8 Bits (including Parity)
Parity	None
Stop Bits	1

Advanced Parameters

Data Direction	Receive and Transmit
Over Sampling	16 Samples
Single Sample	Disable

Figure 6: Setting up the UART1 on the MCU.



STM32Cube MCU packages and embedded software packs

○ Copy all used libraries into the project folder

● Copy only the necessary library files

○ Add necessary library files as reference in the toolchain project configuration file

Generated files

☑ Generate peripheral initialization as a pair of '.c/.h' files per peripheral

☐ Backup previously generated files when re-generating

☑ Keep User Code when re-generating

☑ Delete previously generated files when not re-generated

HAL Settings

☐ Set all free pins as analog (to optimize the power consumption)

☐ Enable Full Assert

Template Settings

Select a template to generate customized code

Figure 7: Project Manager configuration for separating the peripherals and main function.

Printing output with UART

C as a language is designed to be portable between many hardware platforms. The standard library *stdio* is designed to implement input and output functions.

> **Task 2:** In the Project created in the previous task, add an include directive for stdio.h in the main file. Make sure you write your own code in the specified regions!

To port this standard library to any platform and hardware interface, we need to implement one function to read a single character from a stream and one function to write a single character to a stream. We can then use the well-known *stdio* functions like `printf`. The functions needed are `__io_putchar` and `__io_getchar`.

For this exercise, we would like to transfer data from the microcontroller to a computer with the *USART1* interface. We therefore use the Hardware Abstraction Layer (HAL) functions `HAL_UART_Transmit` and `HAL_UART_Receive`.

> **Task 3:**
>
> 1. Add a function definition for `_write` to write to the output stream in the appropriate code section of the main file.
> 2. Add a `printf` statement to the main function body to announce the beginning of the inference; this will furthermore test the serial communication.

> **Note:** ”Make sure you write your own code in the specified regions” Maybe put a code snippet to highlight the region
>
> ```
> /* Private user code ------------------------------/
> / USER CODE BEGIN 0 */
>
> /* USER CODE END 0 */
> ```

```
__attribute__((weak)) int _write(int file, char* ptr, int len)
{
  HAL_UART_Transmit(&hUART1, (uint8_t*)ptr, len, HAL_MAX_DELAY);
  return len;
}
```

Compiling and flashing a project

The project can be compiled by *simply clicking the* BUILD *button* as shown in Figure 8.

However, this is not entirely true as you have to be very careful when starting your own exploration. We will give you a basic intuition of what can go wrong and how to fix it. In Figure 9, a general compilation flow is shown. We start with a simple C program. C is a *compiled language*, i.e. a compiler is needed to translate source code to machine-readable code. One of the most famous ones is the GNU Compiler Collection (GCC), which is also used in the STM32 CUBE IDE . Compilers translate source code in four steps: 1) Preprocessing, 2) Compiling, 3) Assembling, and 4) Linking. If you want to read up on the different steps you can refer to this blog post.

We will focus mainly on the **Linking** part, as this is usually where things go wrong. The **Assembler** goes through all the files in your `C/C++` project and generates so-called *object code (binary)*. This is essentially pure machine code, which runs on your target architecture. However, since we have multiple files in our project such as header files, or external libraries, we need the **Linker** to combine everything into a single *executable*. As the name suggests, this file contains the code which is in the end run on the MCU to perform the inference.

However, the **Linker** needs some help when using libraries and external files such as *CMSIS* or *TFLite*. Both tools contain a hierarchy of definitions and macros. Macros are essentially function definitions used across several source files. These definitions are usually all listed in so-called header files. You request the use of a header file in your program by *including* it, with the C preprocessing directive `#include`. Thus, you can access the macros within the *included* header file.

In order for a project to successfully compile, we have to tell the **Linker** the order of macro definitions. Otherwise, it will try to link functions that have not yet been defined. If you encounter error messages during building such `undefined reference to XYZ`, it is probably due to the misordering of the paths within your project. You can check the order of the search paths under `Properties`→`Paths and Symbols`→`Paths and Symbols`→`Includes`. Make sure you select ”Add to all configurations” and ”Add to all languages” checkboxes while adding.

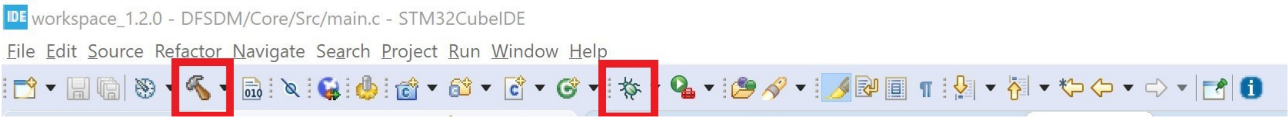

IDE workspace_1.2.0 - DFSDM/Core/Src/main.c - STM32CubeIDE
File Edit Source Refactor Navigate Search Project Run Window Help

Figure 8: The Build and Debug icons in the Toolbar.

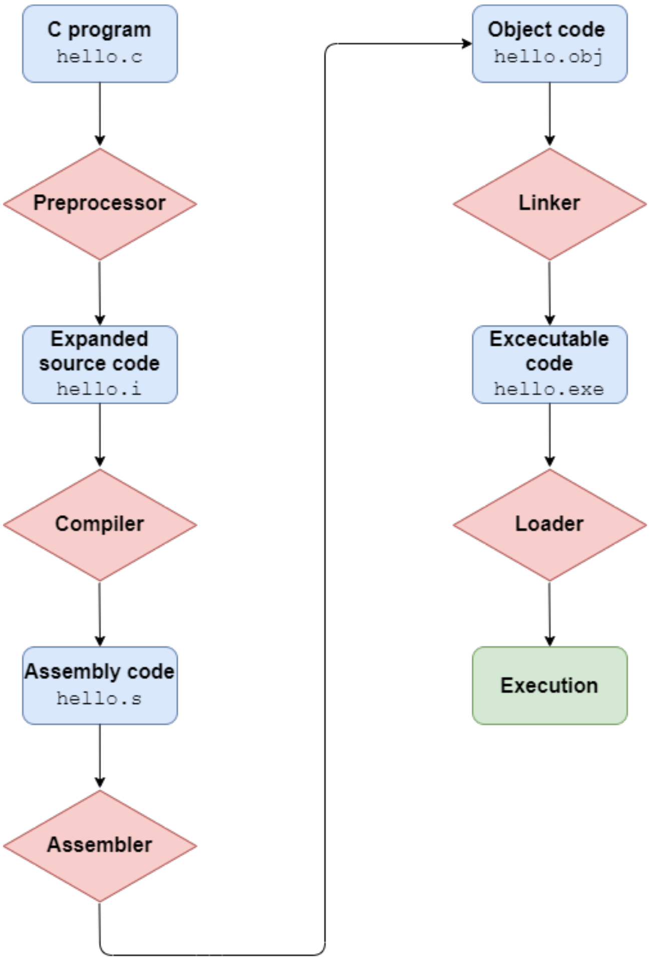


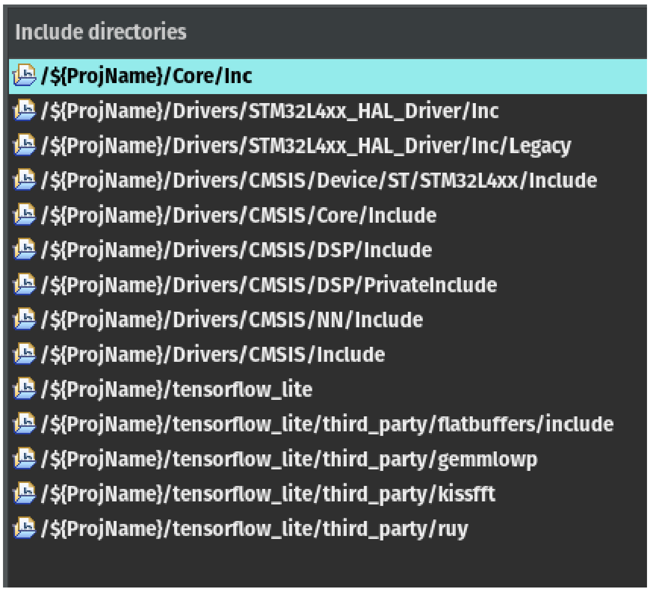
Figure 9: General compilation flow.

Task 4 (Project preparation):

1. Extract the exercise materials.
2. Include the CMSIS (*delete the folder automatically generated by the IDE*) and TFLite toolchains that you just downloaded in your new project.
3. Make sure that your search paths for compilation and linking are set up correctly.
4. Under `Core→Inc` add a new folder called `models`. This is where we will store the TFLite model header files for the inference step on the MCU.

Note: When adding the includes, do this through the IDE's file browser. Also check that you have a Symbol `CMSIS_NN` and make sure to add `tensorflow_lite` as a source location in the `Paths and Symbols` section of the Project properties.

Note: In this exercise, we have provided the *CMSIS* and *TFLite* toolchains for you. If you want to update the toolchains for your own projects, you can find both CMSIS and TFLite on GitHub. It is usually good to check these repositories from time to time as more and more kernels are added to these toolchains, which could improve your performance and accuracy significantly.



Include directories

```
/${ProjName}/Core/Inc
/${ProjName}/Drivers/STM32L4xx_HAL_Driver/Inc
/${ProjName}/Drivers/STM32L4xx_HAL_Driver/Inc/Legacy
/${ProjName}/Drivers/CMSIS/Device/ST/STM32L4xx/Include
/${ProjName}/Drivers/CMSIS/Core/Include
/${ProjName}/Drivers/CMSIS/DSP/Include
/${ProjName}/Drivers/CMSIS/DSP/PrivateInclude
/${ProjName}/Drivers/CMSIS/NN/Include
/${ProjName}/Drivers/CMSIS/Include
/${ProjName}/tensorflow_lite
/${ProjName}/tensorflow_lite/third_party/flatbuffers/include
/${ProjName}/tensorflow_lite/third_party/gemmlowp
/${ProjName}/tensorflow_lite/third_party/kissfft
/${ProjName}/tensorflow_lite/third_party/ruy
```

Figure 10: Include directories

4 Preparing the Inference step on the MCU

In the exercise materials we provide you with a `Jupyter notebook` to convert a trained *Keras* model via *TFlite* to a *C header file*. The header file contains all the network parameter information used for *inference*. We will start with a very simple dataset, i.e. the Fashion MNIST dataset, and a straightforward network implementation.

Task 5:

1. Open the `Jupyter notebook` file `lab7.ipynb` with the editor of your choice and follow the instructions. We will start with **Task 1**.
2. What is the loss and accuracy that you can achieve on the test set?
   - Loss: ______________________________
   - Accuracy: ______________________________
3. Briefly explain the difference between the test and validation set.
   - Validation set: ______________________________
   - Test set: ______________________________
4. What is the size of the `.h5` model and the `.tflite` model? By how much can we reduce the model size?
   - Size of the `.h5` file: ______________________________
   - Size of the `.tflite` file: ______________________________

- Compression factor: ______________________________

____

Now you have successfully converted your *Keras* model to a *TFLite* model for the MCU.

5 Deploying the network to the MCU for inference

Deploying the network effectively means storing the network's parameters (e.g., weights and biases), usually in the MCU's read-only memory, as well as generating the C code implementing the network's computational graph, managing intermediate buffer memory and calling (optimized) kernel implementations of individual layers. Therefore, the first prerequisite for a successful deployment is ensuring that the network's size, given the number of parameters and their precision, is smaller than the available storage space.

Task 6:

1. Move on to **Task 2** in the `Jupyter Notebook`.
2. Count the model parameters, considering the weights and biases within each layer. Verify your results by comparing them with the output of the `summary()` [a] method.

   ____

   ____

3. By how much can you reduce the model size by performing full 8-bit quantization?

   ____

   ____

4. What accuracy can we achieve with the fully quantized model? Why do you think the quantized model might achieve higher accuracy than the full precision model?

   ____

   ____

[a] Note that similar functionality is covered in PyTorch by the `torchsummary` library.

Another hardware-associated constraint that must be addressed when deploying a model on an MCU is represented by the memory limitations. These limitations refer to the available read-write memory used for the intermediate buffers; in the absence of tiling and multi-buffered memory accesses, the memory requirements of a network can be defined as:

$$M = \max_{l \in L} l_{in} + l_{out} + l_p \tag{1}$$

, where l_{in} represent the input activations of a layer l of a network comprised of L layers, l_{out} represent the output activations, whilst l_p are the layer's parameters. Similarly to the storage requirements, the memory limitations also depend on the precision used to represent the data.

Task 7:

1. What are the memory requirements of your network? Do they fit the constraints of your target platform?

In the next step, we have to include the trained network in our project and deploy it on the MCU.

Task 8:

1. In the exercise material we provide you the application source code and header file to run the inference on the MCU. Add the `.h` and `.cpp` file at the right locations to your project.
2. Surround the inference call (running the inference step) with a *CycleCounter* to benchmark the number of cycles taken for an inference step. You can find the code for this step later in the document.
3. Add the model `.h` file at the correct location to your project that you generated from the `Jupyter notebook`. Make sure you include the model in your application.
4. Now you can compile and flash your project.
5. Once you verify that the build is successful and the application is started, the `printf` statement from the main function body, announcing the inference step, can be commented out.

Here you can find the code to count the cycles of a function on your MCU.

Define:

```
    /* DWT (Data Watchpoint and Trace) registers,
    only exists on ARM Cortex with a DWT unit */
#define KIN1_DWT_CONTROL             (*((volatile uint32_t*)0xE0001000))
  /*!< DWT Control register */
#define KIN1_DWT_CYCCNTENA_BIT       (1UL<<0)
  /*!< CYCCNTENA bit in DWT_CONTROL register */
#define KIN1_DWT_CYCCNT              (*((volatile uint32_t*)0xE0001004))
  /*!< DWT Cycle Counter register */
#define KIN1_DEMCR                   (*((volatile uint32_t*)0xE000EDFC))
  /*!< DEMCR: Debug Exception and Monitor Control Register */
#define KIN1_TRCENA_BIT              (1UL<<24)
  /*!< Trace enable bit in DEMCR register */

#define KIN1_InitCycleCounter() \
  KIN1_DEMCR |= KIN1_TRCENA_BIT
  /*!< TRCENA: Enable trace and debug block DEMCR
  (Debug Exception and Monitor Control Register */

#define KIN1_ResetCycleCounter() \
  KIN1_DWT_CYCCNT = 0
  /*!< Reset cycle counter */

#define KIN1_EnableCycleCounter() \
  KIN1_DWT_CONTROL |= KIN1_DWT_CYCCNTENA_BIT
  /*!< Enable cycle counter */

#define KIN1_DisableCycleCounter() \
  KIN1_DWT_CONTROL &= ~KIN1_DWT_CYCCNTENA_BIT
  /*!< Disable cycle counter */

#define KIN1_GetCycleCounter() \
  KIN1_DWT_CYCCNT
  /*!< Read cycle counter register */
```

C Code:

```
uint32_t cycles; /* number of cycles */

KIN1_InitCycleCounter(); /* enable DWT hardware */
KIN1_ResetCycleCounter(); /* reset cycle counter */
KIN1_EnableCycleCounter(); /* start counting */
foo(); /* call function and count cycles */
cycles = KIN1_GetCycleCounter(); /* get cycle counter */
KIN1_DisableCycleCounter(); /* disable counting if not used any more */
```

Note: Our real-time inference script in the next tasks reads the bytes directly from the serial stream. Make sure that you do not have any `printf` statements such as e.g. from the *Cycle-Counter* in your code that contaminate the stream.

6 Real-time inference on the MCU

Finally, we can perform the inference on the MCU. For this purpose, we provide you with a small Graphical User Interface (GUI), which is programmed in the `test.py` script provided in the exercise materials. In Figure 11, the general working principle of the test script is shown. We send the test image data together with its label to the MCU. After one inference step we read out the UART port to retrieve the predicted label.

In order to achieve real-time operation for our system, our model's latency (i.e., the interval between the model receiving the input and said model producing the prediction) has to be smaller than the data acquisition time, the latter being emulated here using the `test.py` script. The inference latency can thus be considered a third hardware-associated constraint, further determining the energy consumption (i.e., $E = P \cdot t$) of our system. Although the number of Floating Point Operations (FLOPs) might represent a sufficiently good proxy for the latency when comparing different networks with a similar architecture, optimizations such as the usage of Single Instruction Multiple Data (SIMD)-based kernels, tiling, or double buffering could make the FLOPs-based comparison obsolete. It is thus recommended, when optimizing a neural network considering an accuracy-latency trade-off, to perform hardware-in-the-loop optimizations by measuring the network's latency on the target platform.

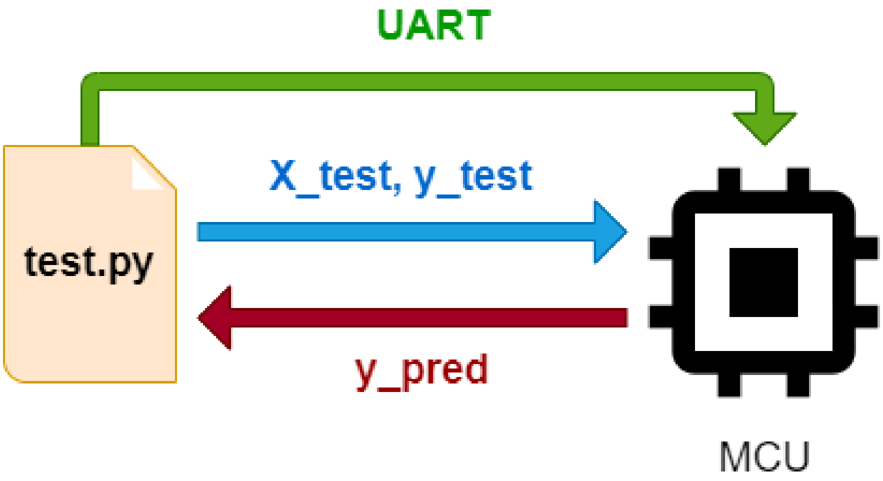


Figure 11: Real-time Inference step on the MCU.

Task 9:

1. Open the `Anaconda Prompt` shell and navigate to the location of your `inference.py` script. Make sure your environment is activated.
2. Check out the source code of the `inference.py` script and try to understand how it works.
3. To perform the inference run the following command: `python inference.py`. *Attention:* You might have to modify the script slightly with your COM port.
   For UNIX users, change the port in the script to the path you use for the UART connection i.e. `/dev/ttyACM0`
4. What is the latency and memory usage of your network? ____________________

   ____

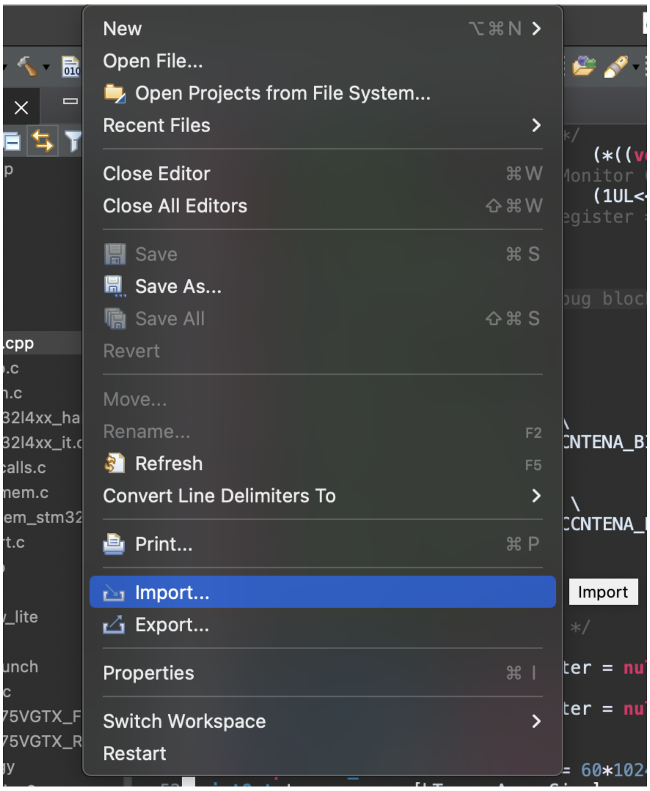
ℰ **Congratulations! You have reached the end of the exercise.** ℰ
If you are unsure of your results, discuss them with an assistant.

7 Solutions

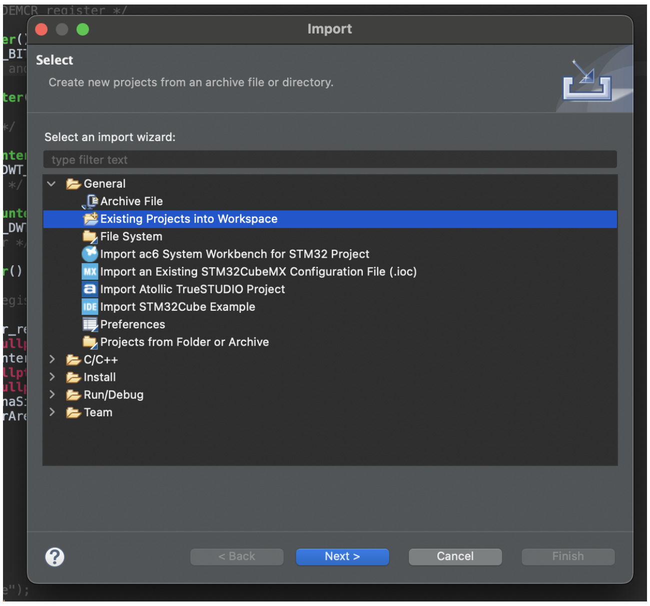
In case you did not finish the tasks in time, the hand-out you were given also presents the solutions.

To import a project in STM32CubeIDE you have to follow the following steps:

- Click on file and Import



- Select ”Existing objects into the workspace”


Import

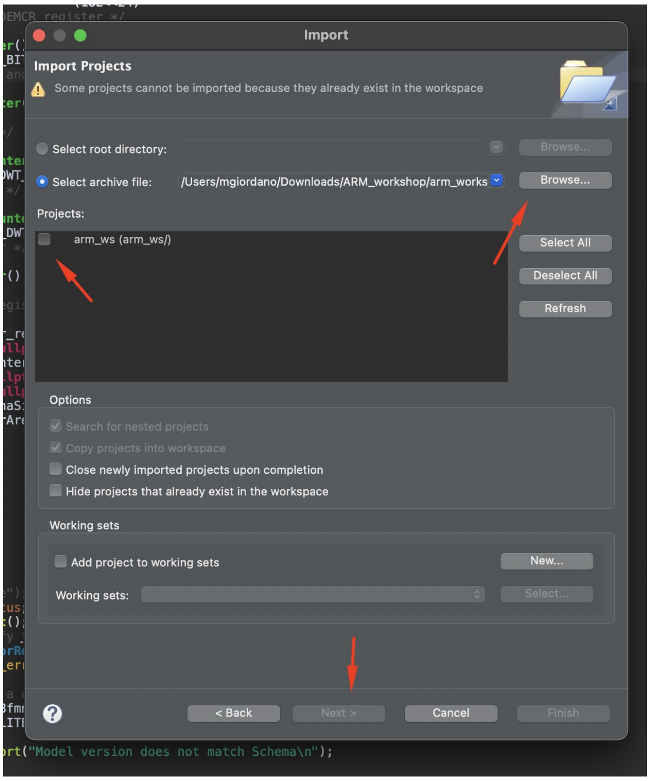
Select

Create new projects from an archive file or directory.

Select an import wizard:

type filter text
General
Archive File
Existing Projects into Workspace
File System
Import ac6 System Workbench for STM32 Project
Import an Existing STM32CubeMX Configuration File (.ioc)
Import Atollic TrueSTUDIO Project
Import STM32Cube Example
Preferences
Projects from Folder or Archive
C/C++
Install
Run/Debug
Team
< Back
Next >
Cancel
Finish

- Check ”Select Archive file and browse to the right location and continue to import the project.



Import

Import Projects

⚠ Some projects cannot be imported because they already exist in the workspace

○ Select root directory:

● Select archive file: /Users/mgiordano/Downloads/ARM_workshop/arm_works

Projects:

□ arm_ws (arm_ws/)

Options

☑ Search for nested projects

☑ Copy projects into workspace

□ Close newly imported projects upon completion

□ Hide projects that already exist in the workspace

Working sets

□ Add project to working sets

Working sets:

```
rt("Model version does not match Schema\n");
```