

1. Download the AlfES® library

- Got to: https://github.com/Fraunhofer-IMS/AlfES_for_Arduino
- Download AlfES® as ZIP archive using "Download ZIP"

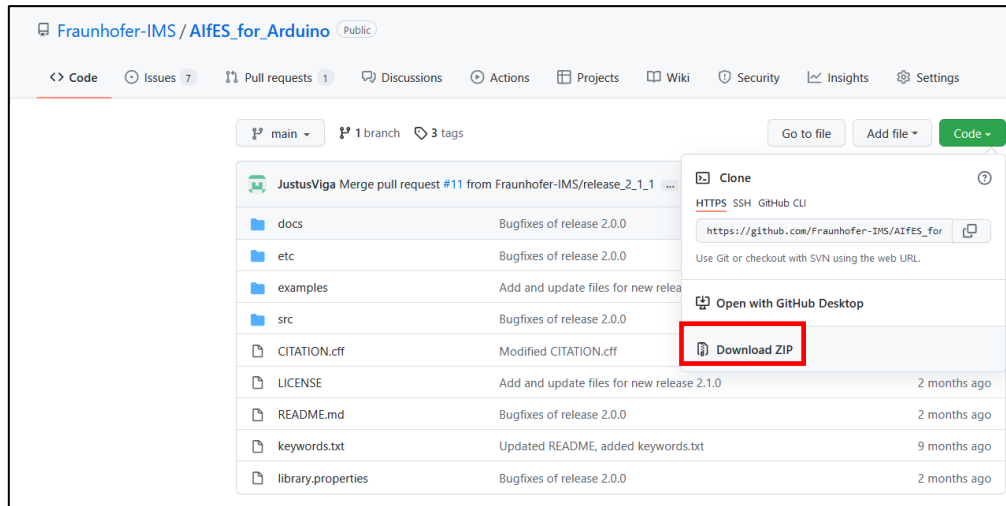


Figure 1: Download AlfES, by using the "Download Zip" Function

2. Prepare AlfES®

We have different configuration files for AlfES® to make it compatible with different IDEs. In the default configuration it is of course optimized for the Arduino IDE. Arduino specific functions like `"serial.print()"` are used. By changing the configuration files, you can make AlfES® universal. Then the classic `"printf()"` functions are used. You can then use AlfES® almost everywhere, on the PC, in a microcontroller IDE of your choice, even in a hardware simulator.

How this works is explained here step by step:

- Extract the ZIP file
- Navigate to the `"src/"` folder
- Replace the files `"aifes_config.cpp"` and `"aifes_config.h"` in the `"src/"` directory by the ones found in `"etc/aifes_configurations/pc/"`
- Note that the new config file has the extension `*.c` and not `*.cpp`
- That's it

3. Use Arm CMSIS

Of course, you can also use CMSIS in the universal version if you use e.g. another microcontroller IDE. For the installation please follow this [guide](#).

The only difference is that you don't make the changes in the file structure of the Arduino IDE, but in your just downloaded copy.

STMCubeIDE

4. Create a new project

Open STMCubeIDE and create a New Project under: *File* → *New* → *STM32 Project*

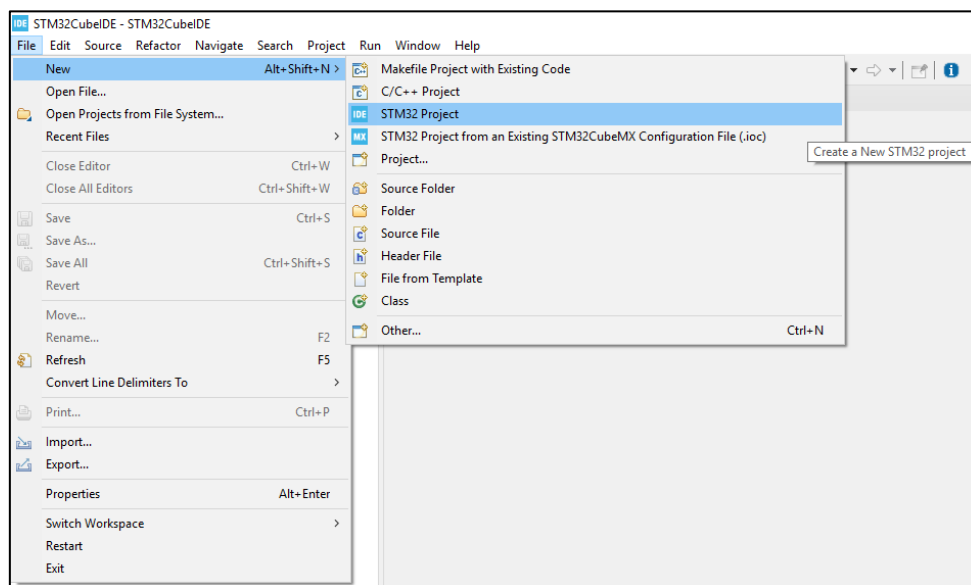


Figure 2: Create new project

Choose your MCU or Board in the next dialog.

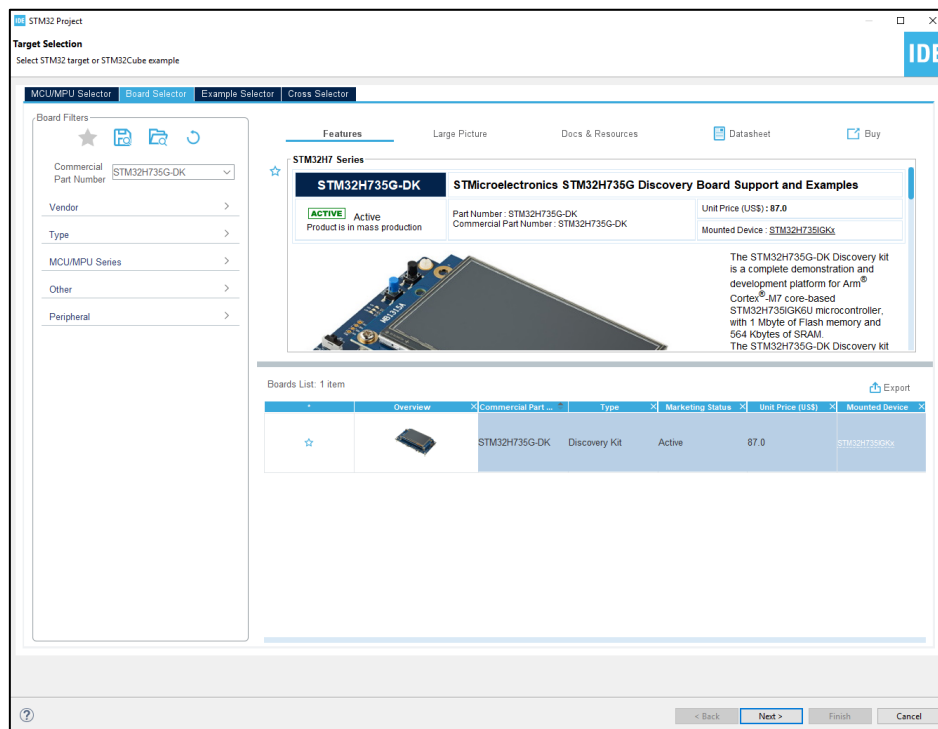


Figure 3: Choose your MCU or Board

STMCubeIDE

Give your project a name and keep the default settings.

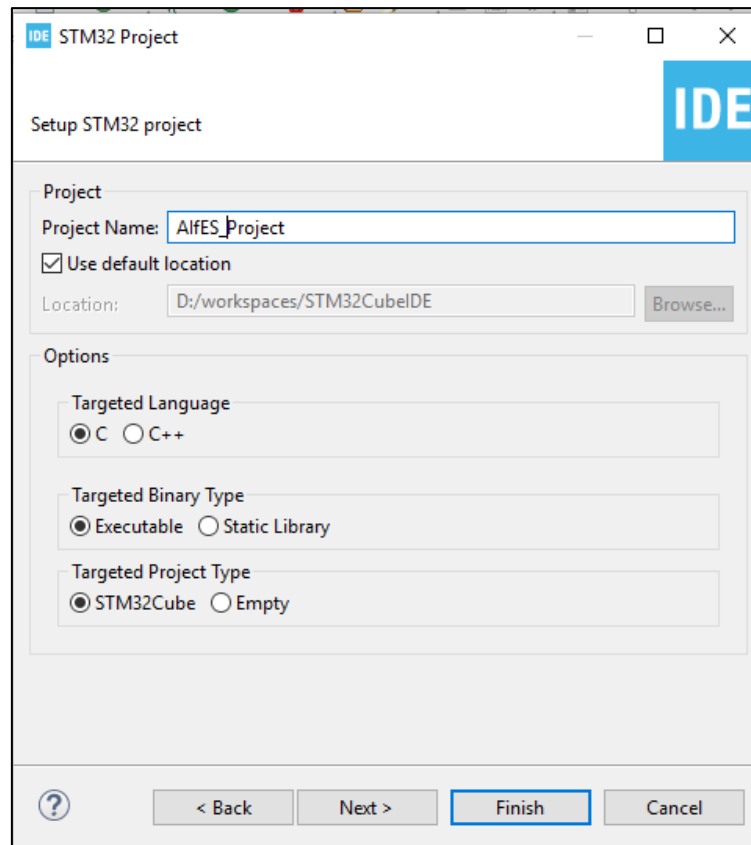


Figure 4: Project setup

Confirm the default settings with **Yes**.

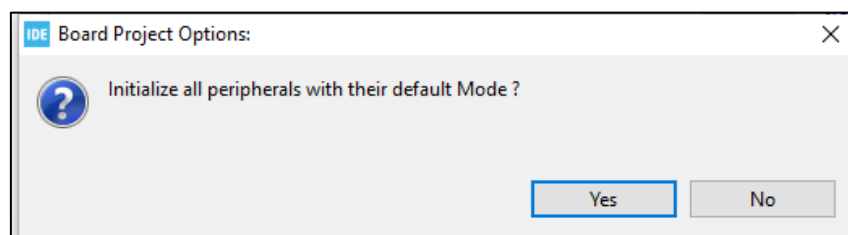


Figure 5: Default mode -> Yes

STMCubeIDE

5. Import AlfES®

Wait until your new Project is initialized, then import the AlfES® Source Code, by right click on your project and select Import.

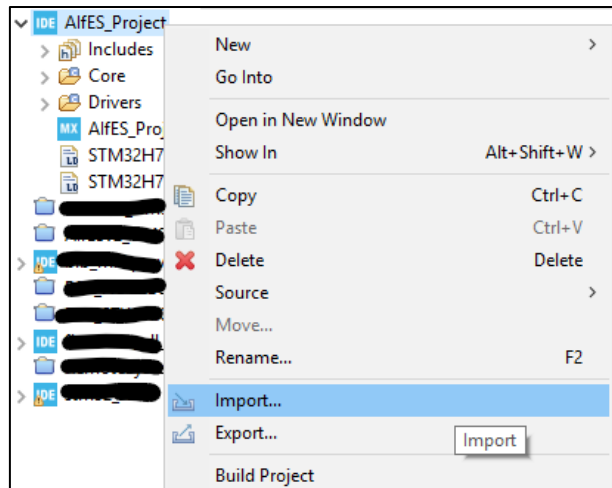


Figure 6: Right click on your project -> Import

Choose General → File System

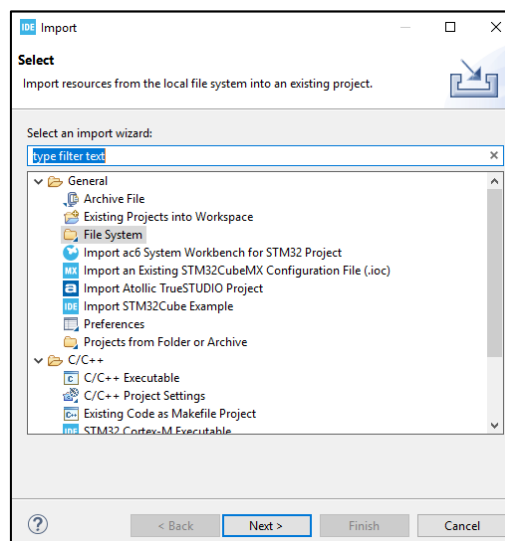


Figure 7: Choose General → File System

AlfES - Installation guide



STMCubeIDE

Open the path of your extracted ZIP archive and Import the complete **src** Folder. Under "Into Folder", use **[Project_Name]/Core/aifes**. Click Finish.

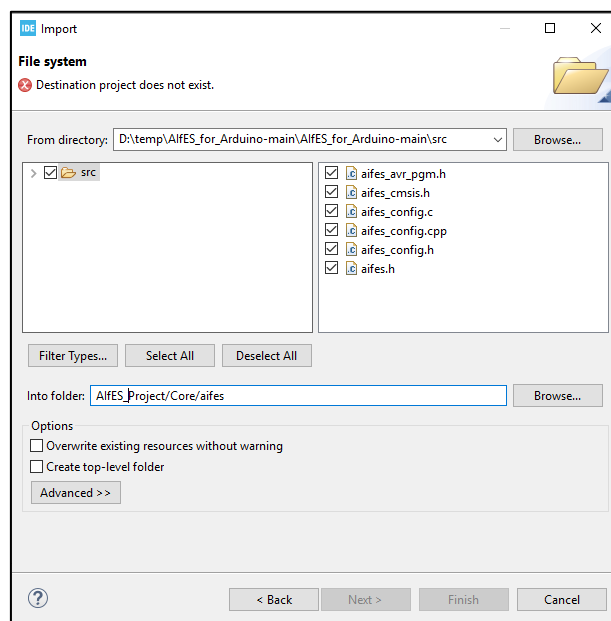


Figure 8: Import AlfES®

Open the Project Properties by right click on your project and select properties.

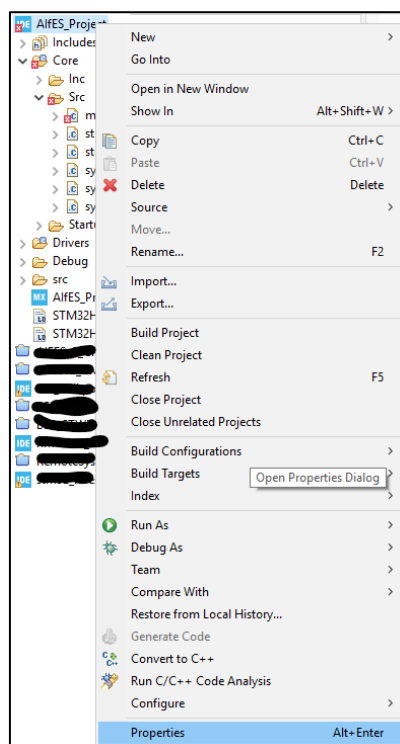


Figure 9: Project properties

STMCubeIDE

Under C/C++ Build → Settings → MCU Settings activate the **printf** and **scanf** options for floats.

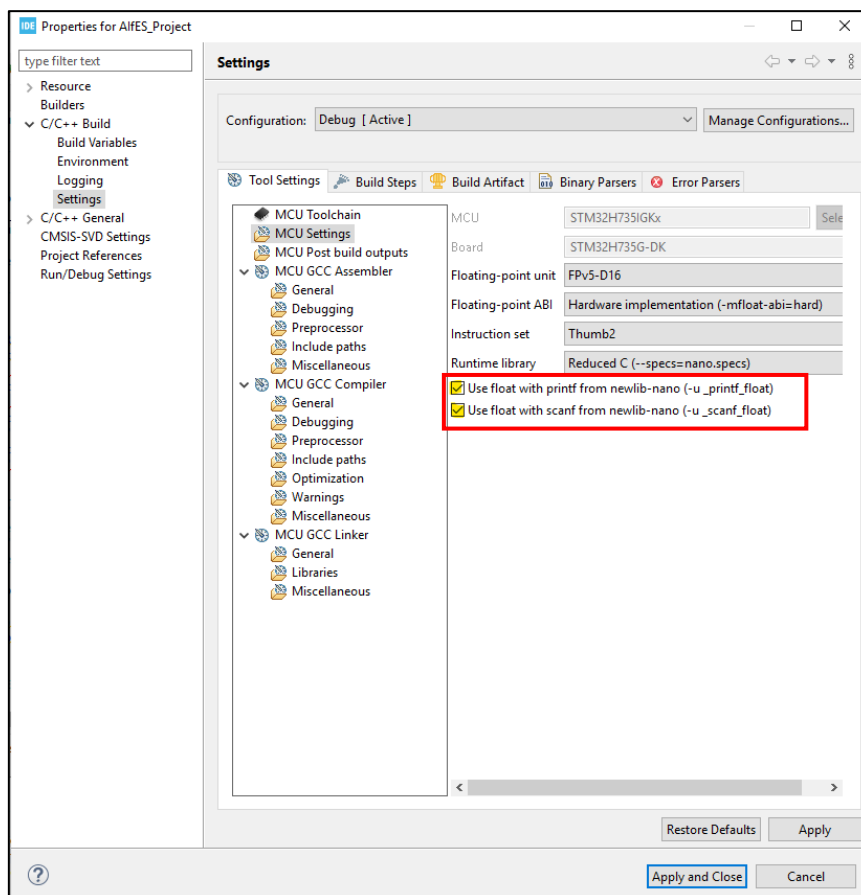


Figure 10: Activate the **printf** and **scanf** options for floats

STMCubeIDE

Also, under properties in MCU GCC Compiler --> Include paths, add the **.../Core/aifes** to the Include paths. Click Apply and Close.

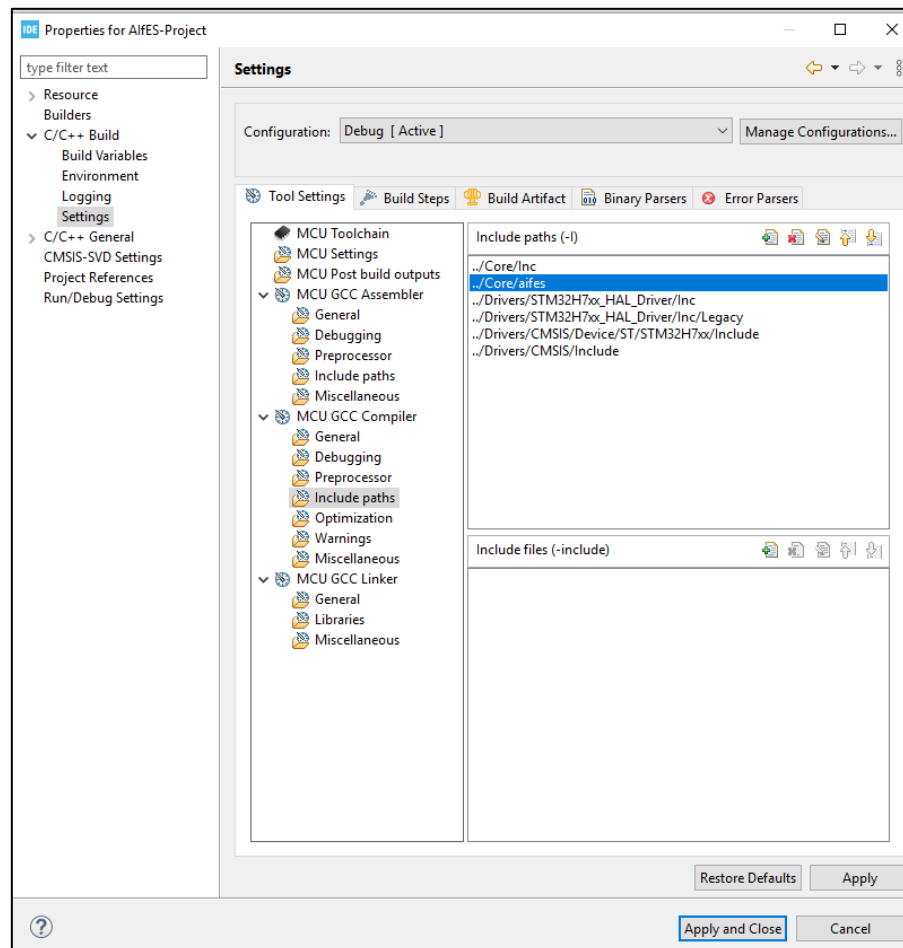


Figure 11: Include path

6. Run an Example

Open the **main.c** in Core and add following Code to the corresponding code sections

Add to "USER CODE BEGIN Includes"

```
/* USER CODE BEGIN Includes */
#include "aifes.h"
/* USER CODE END Includes */
```



STMCubeIDE

Add to “USER CODE BEGIN PFP”

```
/* USER CODE BEGIN PFP */
#ifdef __GNUC__
/* With GCC, small printf (option LD Linker->Libraries->Small printf
   set to 'Yes') calls __io_putchar() */
#define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
#else
#define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
#endif /* __GNUC__ */

/* USER CODE END PFP */
```

Add to “USER CODE BEGIN WHILE” and “USER CODE END 3”

```
/* USER CODE BEGIN WHILE */
while (1){
    printf("\n\n\nAIfES\n\n");

    //Tensor for the input data
    float input_data[] = {0.0f, 1.0f}; // Input
data for the XOR ANN (0.0 / 1.0)
    uint16_t input_shape[] = {1, 2}; //
Definition of the input shape
    aitenor_t input_tensor = AITENSOR_2D_F32(input_shape, input_data); //
Creation of the input AIfES tensor with two dimensions and data type F32 (float32)

    // ----- Layer definition -----
    -----

    // Input layer
    uint16_t input_layer_shape[] = {1, 2}; // Definition of the input layer shape
(Must fit to the input tensor. The first dimension is the batch size.)
    ailayer_input_f32_t input_layer = AILAYER_INPUT_F32_M( /*input dimension=*/ 2,
/*input shape=*/ input_layer_shape); // Creation of the AIfES input layer

    // Hidden dense layer
    float weights_data_dense_1[] = {-10.1164f, -8.4212f, 5.4396f, 7.297f, -7.6482f, -
9.0155f}; // Hidden layer weights
    float bias_data_dense_1[] = {-2.9653f, 2.3677f, -1.5968f};
// Hidden layer bias weights
    ailayer_dense_f32_t dense_layer_1 = AILAYER_DENSE_F32_M( /*neurons=*/ 3,
/*weights=*/ weights_data_dense_1, /*bias=*/ bias_data_dense_1); // Creation of the AIfES
hidden dense layer with 3 neurons

    // Hidden layer activation function
    ailayer_sigmoid_f32_t sigmoid_layer_1 = AILAYER_SIGMOID_F32_M();

    // Output dense layer
    float weights_data_dense_2[] = {12.0305f, -6.5858f, 11.9371f}; // Output dense
layer weights
    float bias_data_dense_2[] = {-5.4247f}; //Output dense layer
bias weights
    ailayer_dense_f32_t dense_layer_2 = AILAYER_DENSE_F32_M( /*neurons=*/ 1,
/*weights=*/ weights_data_dense_2, /*bias=*/ bias_data_dense_2); // Creation of the AIfES
output dense layer with 1 neuron
```




STMCubeIDE

```
// Output layer activation function
ailayer_sigmoid_f32_t sigmoid_layer_2 = AILAYER_SIGMOID_F32_M();

// ----- Define the structure of the model -----
-----

aimodel_t model; // AIfES model
ailayer_t *x;    // Layer object from AIfES to connect the layers

// Connect the layers to an AIfES model
model.input_layer = ailayer_input_f32_default(&input_layer);
x = ailayer_dense_f32_default(&dense_layer_1, model.input_layer);
x = ailayer_sigmoid_f32_default(&sigmoid_layer_1, x);
x = ailayer_dense_f32_default(&dense_layer_2, x);
model.output_layer = ailayer_sigmoid_f32_default(&sigmoid_layer_2, x);

aialgo_compile_model(&model); // Compile the AIfES model

// ----- Print the model structure -----
-----

printf("----- Model structure -----\\n");
aialgo_print_model_structure(&model);
printf("-----\\n");

// ----- Allocate and schedule the working memory for
inference -----

// Allocate memory for result and temporal data
uint32_t memory_size = aialgo_sizeof_inference_memory(&model);

char *memory_ptr = (char *) malloc(memory_size);
// Here is an alternative if no "malloc" should be used
// Do not forget to comment out the "free(memory_ptr);" at the end if you use this
solution
// byte memory_ptr[memory_size];

// Schedule the memory over the model
aialgo_schedule_inference_memory(&model, memory_ptr, memory_size);

// ----- Run the inference -----
-----

// Create an empty output tensor for the inference result
uint16_t output_shape[2] = {1, 1};
float output_data[1*1]; // Empty data array of size output_shape
aitensor_t output_tensor = AITENSOR_2D_F32(output_shape, output_data);

aialgo_inference_model(&model, &input_tensor, &output_tensor); // Inference /
forward pass

// ----- Print result -----
-----

printf("\\n\\nResults:\\n");
printf("input 1:\\tinput 2:\\treale output:\\tcalculated output:\\n");
printf ("%f", input_data[0]);
```



STMCubeIDE

```
printf ("\t\t");
printf ("%f", input_data[1]);
printf ("\t\t");
printf ("1.0");
printf ("\t\t");
printf ("%f\n", ((float* ) output_tensor.data)[0]);

// How to print the weights example
// printf(F("Dense 1 - Weights:"));
// print_aitensor(&dense_layer_1.weights);
// printf(F("Dense 1 - Bias:"));
// print_aitensor(&dense_layer_1.bias);

free(memory_ptr);

HAL_Delay(500);

/* USER CODE END WHILE */
/* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
```

Add to “USER CODE BEGIN 4”

```
/* USER CODE BEGIN 4 */
/**
 * @brief Retargets the C library printf function to the USART.
 * @param None
 * @retval None
 */
PUTCHAR_PROTOTYPE
{
    /* Place your implementation of fputc here */
    /* e.g. write a character to the USART1 and Loop until the end of transmission */
    HAL_UART_Transmit(&huart3, (uint8_t *)&ch, 1, 0xFFFF);

    return ch;
}
/* USER CODE END 4 */
```

7. Known issues

The example code used USART3 (*USER CODE BEGIN 4*). However, some boards such as the STM32F746 do not support this. Choose here another one like e.g. USART1.

```
HAL_UART_Transmit(&huart1, (uint8_t *)&ch, 1, 0xFFFF);
```

8. Legal Disclaimer

The author of this installation guide assumes no responsibility or liability for any errors or omissions in the contents of this document that may result in damage to any kind of hardware or software. The information contained in this manual is provided without warranty of completeness, functionality, accuracy, usefulness or timeliness...