Getting started with X-CUBE-AI Expansion Package for Artificial Intelligence (AI)

Introduction

This user manual provides the guidelines to build step-by-step a complete Artificial Intelligence (AI) IDE-based project for STM32 microcontrollers with automatic conversion of pre-trained Neural Networks (NN) and integration of the generated optimized library. It describes the X-CUBE-AI Expansion Package that is fully integrated with the STM32CubeMX tool. This user manual also describes optional add-on AI test applications or utilities for AI system performance and validation.

The main part of the document is a hands-on learning to generate quickly an STM32 AI-based project. A NUCLEO-F746ZG development kit and several models for Deep Learning (DL) from the public domain are used as practical examples. Any STM32 development kits or customer boards based on a microcontroller in the STM32F3, STM32F4, STM32L4, STM32L4+, STM32F7, STM32H7, or STM32WB series can also be used with minor adaptations.

The next part of the document details and describes the use of the X-CUBE-AI for AI performance and validation add-on applications. It covers also internal aspects such as the generated NN library. Additionally, more information (command-line support, supported toolboxes and layers, reported metrics) are available from the Documentation folder in the installed package.
1 General information

The X-CUBE-AI Expansion Package is dedicated to AI projects running on STM32 Arm® Cortex®-M-based MCUs.

The descriptions in the current revision of the user manual are based on:
- X-CUBE-AI r4.1.0
- Embedded inference client API 1.1.0
- Command-line interface 1.1.0

The pre-trained Keras DL model used for the example in this document is:

Note: Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

1.1 What is STM32Cube?

STM32Cube is an STMicroelectronics original initiative to significantly improve designer’s productivity by reducing development effort, time and cost. STM32Cube covers the whole STM32 portfolio.

STM32Cube includes:
- A set of user-friendly software development tools to cover project development from the conception to the realization, among which:
  - STM32CubeMX, a graphical software configuration tool that allows the automatic generation of C initialization code using graphical wizards
  - STM32CubeIDE, an all-in-one development tool with peripheral configuration, code generation, code compilation, and debug features
  - STM32CubeProgrammer (STM32CubeProg), a programming tool available in graphical and command-line versions
  - STM32CubeMonitor-Power (STM32CubeMonPwr), a monitoring tool to measure and help in the optimization of the power consumption of the MCU
- STM32Cube MCU & MPU Packages, comprehensive embedded-software platforms specific to each microcontroller and microprocessor series (such as STM32CubeF7 for the STM32F7 Series), which include:
  - STM32Cube hardware abstraction layer (HAL), ensuring maximized portability across the STM32 portfolio
  - STM32Cube low-layer APIs, ensuring the best performance and footprints with a high degree of user control over the HW
  - A consistent set of middleware components such as RTOS, USB, FAT file system, graphics and TCP/IP
  - All embedded software utilities with full sets of peripheral and applicative examples
- STM32Cube Expansion Packages, which contain embedded software components that complement the functionalities of the STM32Cube MCU & MPU Packages with:
  - Middleware extensions and applicative layers
  - Examples running on some specific STMicroelectronics development boards

1.2 How does X-CUBE-AI complement STM32Cube?

X-CUBE-AI extends STM32CubeMX by providing an automatic NN library generator optimized in computation and memory (RAM and Flash) that converts pre-trained Neural Networks from most used DL frameworks (such as Caffe, Keras, Lasagne, TensorFlow™ Lite and ConvNetJs) into a library that is automatically integrated in the final user project. The project is automatically setup, ready for compilation and execution on the STM32 microcontroller.

X-CUBE-AI also extends STM32CubeMX by adding, for the project creation, specific MCU filtering to select the right devices that fit specific criteria requirements (such as RAM or Flash memory size) for a user's NN.
The X-CUBE-AI tool can generate three kinds of projects:

- System performance project running on the STM32 MCU allowing the accurate measurement of the NN inference CPU load and memory usage
- Validation project that validates incrementally the results returned by the NN, stimulated by either random or user test data, on both desktop PC and STM32 Arm® Cortex®-M-based MCU embedded environment
- Application template project allowing the building of AI-based application

### 1.3 X-CUBE-AI core engine

The X-CUBE-AI core engine, presented in Figure 1 and Figure 2, is part of the X-CUBE-AI Expansion Package described later in Section 1.4. It provides an automatic and advanced NN mapping tool to generate and deploy an optimized and robust C-model implementation of a pre-trained Neural Network (DL model) for the embedded systems with limited and constrained hardware resources. The generated STM32 NN library (both specialized and generic parts) can be directly integrated in an IDE project or makefile-based build system. A well-defined and specific inference client API (refer to Section 8. Embedded inference client API) is also exported to develop a client AI-based application. Various frameworks (DL toolbox) and layers for Deep Learning are supported (refer to Section 12 Supported toolboxes and layers for Deep Learning).

All X-CUBE-AI core features are available through a complete and unified Command Line Interface (console level) to perform the main steps to analyze, validate, and generate an optimized NN C-library for STM32 devices (refer to [7]). It provides also a post-training quantization support for the Keras model.

A simple configuration interface is exposed. With the pre-trained DL model file, only few parameters are requested:

- **Name**: indicates the name of the generated C model (the default value is "network")
- **Compression**: indicates the compression factor to reduce the size of weight/bias parameters (refer to Section 6.1 Graph flow and memory layout optimizer)
- **STM32 family**: selects the optimized NN kernel run-time library
Figure 2 summarizes the main supported features of the uploaded DL model and targeted sub-system run-time.

- Only simple tensor input and simple tensor output are supported
  - 4-dim shape: batch, height, width, channel ("channel-last" format, refer to [9])
  - Floating-point (32b) and fixed-point (8b) types

- Generated C models are fully optimized for STM32 Arm® Cortex®-M4/M7 cores with FPU and DSP extensions

X-CUBE-AI code generator can be used to generate and deploy a pre-quantized 8-bit fixed-point/integer Keras model and the quantized TensorFlow™ Lite model. For the Keras model, a reshaped model file (.h5*) and a proprietary tensor-format configuration file (json) are required.

The code generator quantizes weights and bias, and associated activations from floating point to 8-bit precision. These are mapped on the optimized and specialized C implementation for the supported kernels (refer to [8]). Otherwise, the floating-point version of the operator is used and float-to-8-bit and 8-bit-to-float convert operators are automatically inserted. The objective of this technique is to reduce the model size while also improving the CPU and hardware accelerator latency (including power consumption aspects) with little degradation in model accuracy.
To generate the reshaped Keras model file and associated tensor-format configuration file from an already-trained floating-point Keras model, the `stm32ai` application (command-line interface) integrates a complete post-training quantization process (refer to [7]).

### 1.4 STM32CubeMX extension

STM32CubeMX is a software configuration tool for STM32 microcontrollers. In one click, it allows the creation of a complete IDE project for STM32 including the generation of the C initializing code for device and platform set up (pins, clock tree, peripherals, and middleware) using graphical wizards (such as the pinout-conflict solver, clock-tree setting helper, and others).

**Figure 4. X-CUBE-AI core in STM32CubeMX**

From the user point of view, the integration of the X-CUBE-AI Expansion Package can be considered as the addition of a peripheral or middleware SW component. On top of X-CUBE-AI core, the following main functionalities are provided:

- **MCU filter selector** is extended with an optional specific AI filter to remove the devices that do not have enough memory. If enabled, STM32 devices without Arm® Cortex®-M4 or -M7 core are directly filtered out.
- **Provides a complete AI UI configuration wizard allowing the upload of multiple DL models. Includes a validation process of the generated C code on the desktop PC and on the target.**
- **Extends the IDE project generator to assist the generation of the optimized STM32 NN library and its integration for the selected STM32 Arm® Cortex®-M core and IDE.**
- **Optional add-on applications allow the generation of a complete and ready-to-use AI test application project including the generated NN libraries. The user must just have imported it inside his favorite IDE to generate the firmware image and program it. No additional code or modification is requested from the end user.**
- **One-click support to generate, program and run automatically an on-device AI validation firmware (including support for the external memory).**
1.5 Acronyms, abbreviations and definitions

Table 1 details the specific acronyms and abbreviations used in this document.

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI</td>
<td>Artificial Intelligence, sometimes called machine intelligence. Commonly, AI is the broad concept of machines being able to carry out tasks in a way that can be considered as &quot;smart&quot; from a human standpoint. It stands for the ability of a digital equipment to perform tasks associated with intelligent beings.</td>
</tr>
<tr>
<td>DL</td>
<td>Deep Learning (also known as deep structured learning or hierarchical learning). DL models are vaguely inspired by information processing and communication patterns in biological nervous systems.</td>
</tr>
<tr>
<td>ML</td>
<td>Machine Learning is an application of Artificial Intelligence (AI) that provides systems with the ability to automatically learn and improve from experience without being explicitly programmed.</td>
</tr>
<tr>
<td>MACC</td>
<td>Multiply-and-accumulate complexity is a unity that indicates the complexity of a DL model from a processing standpoint.</td>
</tr>
<tr>
<td>PINNR</td>
<td>Platform-independent Neural Network representation is a file generated by the front end (X-CUBE-AI core importer) to have a common and portable internal representation of the uploaded DL model for the next stages (optimizer and C-code generator).</td>
</tr>
</tbody>
</table>

1.6 Prerequisites

The following packages must be installed (refer to Section 2 Installing X-CUBE-AI):

- STM32CubeMX version 5.0.1 or later
- Additional SW pack - STM32CubeMX AI (X-CUBE-AI) 4.1.0 pack
- STM32CubeProgrammer (STM32CubeProg) version 2.1.0 or later. Except when STM32CubeIDE is used, it is necessary to install STM32CubeProgrammer to be able to benefit from the automatic validation on the target.

One of the following toolchain or IDEs for STM32 must be installed:

- STM32CubeIDE version 1.0.1 or later
- TrueSTUDIO® for STM32 v9.0.1 or later (atollic.com/truestudio)
- IAR Embedded Workbench™ IDE - ARM v8.x or v7.x (www.iar.com/iar-embedded-workbench)
- µVision® V5.25.2.0 - Keil® MDK-ARM Professional Version (www.keil.com)
- System Workbench for STM32 (SW4STM32)
- GNU Arm Embedded Toolchain (developer.arm.com/open-source/gnu-toolchain/gnu-rm)

X-CUBE-AI can be deployed on the following operating systems:

- Windows® 10
- Ubuntu® 18.4 and Ubuntu® 16.4 (or derived)
- macOS® (x64) (tested on OS X® El Capitan and Sierra)

Note: Ubuntu® is a registered trademark of Canonical Ltd.
macos® and OS X® are trademarks of Apple Inc. registered in the U.S. and other countries.

1.7 License

X-CUBE-AI is delivered under the Mix Ultimate Liberty+OSS+3rd-party V1 software license agreement (SLA0048).
After downloading, installing, and launching STM32CubeMX (version 5.0.1 or later), the X-CUBE-AI Expansion Package can be installed in a few steps.

1. From the menu, select [Help] > [Manage embedded software packages] or directly click on the [INSTALL/REMOVE] button.

   **Figure 5. Managing embedded software packs in STM32CubeMX**

2. From the Embedded Software Packages Manager window, press the [Refresh] button to get an updated list of the add-on packs. Go to the STMicroelectronics tab to find X-CUBE-AI.

   **Figure 6. Installing X-CUBE-AI in STM32CubeMX**

If X-CUBE-AI is already installed, preferably remove it before the new installation.
3. Select it by checking the corresponding box and install it by pressing the [Install Now] button. Once the installation is completed, the corresponding box becomes green and the [Close] button can be pressed.

**Figure 7.** X-CUBE-AI in STM32CubeMX
3 Starting a new STM32 AI project

3.1 MCU and board selector

After launching the STM32CubeMX application, click on the [ACCESS TO MCU SELECTOR] or [ACCESS TO BOARD SELECTOR] button. Alternately, select [File] -> [New Project...] or the CTRL-N shortcut.

![Figure 8. Creating a new project](image)

At this point, the typical STM32CubeMX flow can be used to select a specific MCU or board. An optional MCU filter entry allows the exclusion of the MCUs that do not have enough embedded memory (RAM, Flash, or both) to store the optimized STM32 NN library. This specific AI filter is shown in Figure 9.

![Figure 9. AI filter](image)

Note: This feature is not available for the board selector and usable only for one NN model.

Figure 10 illustrates the case where a DL model has been uploaded and analyzed with the default options. A pre-trained NN model (Keras type) from the public domain is used: Human Activity Recognition using CNN in Keras.
Figure 10. AI filter with default option

Figure 11 illustrates the case where a compression factor of 4 is applied.

Figure 11. AI filter with compression x4

Note: During the generation of the NN library, the size of the memory is also checked by the optimizer to notify the user if the minimal RAM and Flash size constraints are not respected according the selected MCU.
To continue, a NUCLEO-F746ZG development kit is selected as shown in Figure 12.

**Figure 12. NUCLEO-F746ZG board selection**

Click on the [Start Project] button to continue and confirm that all peripherals must be initialized with their default modes.

**Figure 13. Initialize all peripherals**

### 3.2 Hardware and software platform settings

Once an MCU or a board is selected, the related STM32 pinout is displayed. From this window, the user can set up the project by adding one or more additional software and peripherals, and configuring the clock.
If an add-on AI application (refer to Section 4.1 Adding the X-CUBE-AI component) is used, a USART-based link with the host development system is expected. For the STM32 Nucleo-144 development board, pins PD9 TX and PD8 RX are connected to the ST-LINK peripheral to support the Virtual COM port (refer to Figure 14).

**Figure 14. USART3 configuration**

For the NUCLEO-F746ZG, additional configurations of the clocks and memory sub-system are also expected to reach high-performance profile.

### 3.2.1 Increase or set the CPU and system clock frequency

1. Click the Clock Configuration tab.
   
   By default, in this lab, the system clock (SYSCLK, HCLK) is 72 MHz.

2. Type 216 in the HCLK (MHz) input blue box (refer to Figure 16) to call the clock wizard to configure automatically the PLL peripheral (and associated clock tree). If the clock wizard pop-up appears as shown in Figure 15, click on the [OK] button to continue.
3.2.2 Set the MCU memory sub-system

- From the Pinout & Configuration tab (refer to Figure 17), click on the [System Core]>[CORTEX_M7] entry to open the Cortex®-M7 configuration wizard. The core instruction and data caches and ART accelerator sub-system must be enabled.

Figure 17. MCU memory sub-system (parameter settings)
3.2.3 CRC

The CRC peripheral is requested to support the NN library run-time protected mechanism. It must be enabled.

Note: This is done automatically by the tool, so that it is not required to do it manually.
4 X-CUBE-AI configuration wizard

4.1 Adding the X-CUBE-AI component

1. Click on the [Additional Softwares] button to add the X-CUBE-AI additional software to the project (refer to Figure 19).

2. From the Additional Software Component Selection window, the X-CUBE-AI/core bundle (refer to Figure 20) must be checked to be able to upload the NN models and generate the associated STM32 NN library. In this case, as the library is fully integrated as a static library, the user only needs to implement his AI-based application/middleware on top of the generated well-defined NN API [6].

3. Optionally, one of the add-on X-CUBE-AI applications (refer to Figure 21) from the X-CUBE-AI/Application bundle can be selected.
   - System Performance: standalone AI test application for performance purpose
   - Validation: AI test application for validation purpose
   - Template application: basic application template for AI application

4. Click on [OK] to finalize the selection
4.2 Enabling the X-CUBE-AI component

To enable and to configure the X-CUBE-AI component, the following additional steps are requested:

1. From the Pinout & Configuration tab, click on the [Additional Software] selector to discover the additional pieces of software. [Click on STMicroelectronics X-CUBE-AI 4.1.0] to open the initial AI configuration window.

2. Check [Artificial Intelligence Core] to enable the X-CUBE-AI core component. [Artificial Intelligence Application] must be also checked to add the add-on AI application.

The AI application that is selected here corresponds to the application enabled during the previous step (refer to Figure 21).

**Figure 22. Main X-CUBE-AI configuration panel**

- The **Main** tab provides an overview and the entry points to add or remove a network (respectively [Add model] and [Delete model] buttons). [+] can be also directly used to add a network.
- The **Platform Settings** tab indicates the handle of the USART peripheral used to report the information (AI System Performance application) or communicate with the host (AI validation application).

**Figure 23. X-CUBE-AI platform setting panel**
4.3 Uploading a pre-trained DL model file

From the Main tab, click on [Add model] or directly on [+] to open a new dedicated <model_name> configuration wizard. Alternatively, if the model was previously provided through the MCU filter, click directly on the network tab to open the NN Configuration pane.

Figure 24. NN configuration wizard

1. The text field entry is used to define the C name of the network (maximum 32 letters). This string is used directly to generate the name of the embedded client inference API (refer to [9]). If only one network is expected, the default network string name can be maintained.

2. The list box entries specify the DL toolbox used to export the DL model file and the associated file format(s) (refer to Section 12 Supported toolboxes and layers for Deep Learning for details).
   – Click on the [Browse..] button to upload the DL file(s) from the host file system. For this hands-on lab, a public Keras HAR model file is uploaded (saved model format).

3. Click on the [Analyze..] button to trigger a pre-analyze of the network reporting the dimensioning information (system integration point of view). Note that the compression factor was set before to 4, else a warning message pop-up is displayed as shown in Figure 25. If the Invalid network message box pops up, select [Window]>[Outputs] for more details in the log console (refer to Section 13 Error handling). Minimum RAM, Flash occupation and original DL model complexity are updated (refer to Section 4.4).

Figure 25. Insufficient RAM/Flash message box

<table>
<thead>
<tr>
<th>Warning</th>
</tr>
</thead>
<tbody>
<tr>
<td>! Required Ram or Flash size for network is bigger than available Ram or Flash</td>
</tr>
<tr>
<td>OK</td>
</tr>
</tbody>
</table>
Figure 26. Uploaded and analyzed DL model

Note: Additional debug/log information can be found in file C:\Users\<username>\.stm32cubemx\STM32CubeMX.log or $HOME/.stm32cubemx/STM32CubeMX.log.

By clicking on the [Advanced Settings] button ( ), it is possible to configure the network to use external Flash memory for the weights or external RAM for weights or activations.

If the project is started from an STM32 STMicroelectronics board with mounted external Flash memory or RAM, the configuration of the external Flash memory or RAM is automatic during code generation. It uses the board BSP provided in the STM32Cube MCU Package to initialize the Flash memory or the RAM correctly. The external Flash memory is used in the memory-mapped mode.
When [Use external flash] is selected, the weights are generated in a separated `network_data.bin` file and the code is generated to point at the address of the beginning of the external Flash memory.

The `network_data.bin` file must be programmed on the external Flash memory on the board manually using a tool such as STM32CubeProgrammer (STM32CubeProg).

**Note:** Using automatic validation on the target causes file `network_data.bin` to be automatically programmed on the external Flash memory on the board.

Selecting the [Use activation buffer] checkbox places the activation buffers in the external RAM at the address specified in the [Start Address] field.

Optionally, it is possible to copy the weights to the external RAM at startup. In this case, the address where to copy the weights is requested.

When using an external Flash memory or RAM on a Cortex®-M7-based STM32 microcontroller, the ICache and DCache of the CORTEX_M7 CPU are enabled automatically, and the memory protection unit is configured to give access to the external RAM or Flash memory automatically.

### 4.4 Dimensioning information report

When a DL model is processed, the dimensioning system informations presented in Table 2 and Figure 28 are reported.

#### Table 2. System informations reporting

<table>
<thead>
<tr>
<th>Reported information</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM</td>
<td>Indicates the size (in bytes) of the expected RW memory chunk used to store the intermediate inference computing values (.data or .bss section).</td>
</tr>
<tr>
<td>ROM/Flash</td>
<td>Indicates the size (in bytes) of generated RO memory chunk to store the weight/bias parameters after compression if requested (.rodata section).</td>
</tr>
</tbody>
</table>
Reported information | Description
--- | ---
Complexity | Indicates the functional complexity of the imported DL model in Multiply And Accumulate operations (MACC). It includes also an approximation of the activation functions (expressed with the same unity).

Note: The minimum RAM and Flash size requirements listed in the AI summary do not take into consideration the memory constraints of the user application (including the RAM to store the input and output tensors). Only the DL model weights/bias and activation memory requirements are considered here. NN kernel functions and specialized model code, including the minimum stack/heap size, are not considered also.

4.4.1 CPU cycles/MACC?
No theoretical relation is defined between the reported complexity and the real performance of the generated NN C library (CPU cycles / MACC). Due to the variability of the targeted environments (including Arm® tool-chain, MCU and underlying sub-system memory setting, NN topology and layers, and optimizations applied), it is difficult to estimate off-line an accurate CPU cycles/MACC vs. STM32 system settings. However, out-of-the-box, the following rough estimations can be used (for a 32-bit floating-point C model):
- STM32 Arm® Cortex®-M4: ~9 cycles/MACC
- STM32 Arm® Cortex®-M7: ~6 cycles/MACC

The add-on "AI System Performance" test application has been specifically designed to report the factual on-device performance (refer to Section 9 AI system performance application for details).

4.4.2 Generated C-model graph representation
Click on the [Show graph] button to show the main structural information of the uploaded DL model that are considered by the C-code generator (Figure 29). It represents the internal representation of the imported DL model before applying the optimizations.
Figure 29. Generated C-model graph

`network`

- **Input**
  - ID: 0
  - Name: input_0
  - Type: Input

- **Conv2D**
  - ID: 0
  - Name: conv2d_1
  - Type: Conv2D

- **Nonlinearity**
  - ID: 0
  - Name: conv2d_1_nl
  - Type: Nonlinearity

- **Pool**
  - ID: 1
  - Name: max_pooling2d_1
  - Type: Pool

- **Reshape**
  - ID: 3
  - Name: flatten_1
  - Type: Reshape

- **Dense**
  - ID: 4
  - Name: dense_1
  - Type: Dense
4.5 Validating the generated C model

Click on the [Validate on desktop] button to launch a validation process of the generated C model. Without custom data, L2 relative error computation (refer to Section 6.2 Validation engine (L2 relative error calculation)) is mainly used. Note that this step is optional but preferable in particular when a compression factor is requested (refer to Section 6.1 Graph flow and memory layout optimizer). When the reference or ground-truth output values are provided with the associated input samples, the predicted values are used to calculate the metrics listed in Table 3 (refer to [7] for more details).

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACC</td>
<td>Classification accuracy</td>
</tr>
<tr>
<td>RMSE</td>
<td>Root mean square error (classification accuracy)</td>
</tr>
<tr>
<td>MAE</td>
<td>Mean absolute error</td>
</tr>
</tbody>
</table>

Table 3. Metrics

More detailed information is reported in the UI log console as shown in Figure 31. In particular the L2r error is also reported for each generated C layer matching with an original layer.
At this step, the uploaded DL model is ready to be integrated in the generated IDE project.

The [Validate on target] option must be used only later when the targeted device is programmed with the special test application “AI Validation”. It must be selected during the previous step (step 3 of Section 4.1 Adding the X-CUBE-AI component). Reported information and usage are fully described in Section 10 AI validation application.

The [Validate on target] button allows the user to run the validation on the target and optionally automatically generate, compile, program and run a temporary project corresponding to the current network.

For the automatic compilation, programming and run to work, verify that the proposed communication port on the target corresponds to the USART (UART, LPUART) connected to the ST-LINK for the Virtual COM port. Optionally, it is possible to force the peripheral instance to use and the pins used for the transmit and receive signals.

By default, the proposed toolchain is selected for the project. It can be updated if needed. If a JTAG interface is used to program the MCU, the default debug interface must be changed.
When pressing [OK], a temporary project is generated, compiled, programmed and started on the target. Then the regular validation of the network takes place.

4.6 Adding a new DL model

Multiple DL models can be imported. The total number is not limited by the wizard, however the initial limitation is mostly related to the sizes of available RAM and Flash in the selected STM32 MCU device. Click on the [+] button to import a new DL model and apply the same previous steps. The Main view summarizes the total RAM and Flash occupations.

Figure 33. Main view with multiple networks
5 Generating, building and programming

5.1 Generating the IDE project

The following steps show in sequence the classical STM32CubeMX process to generate the IDE project without addition of any specific AI extension:

1. Click on the Project Manager view.
2. Set the project location and name.
3. Select one Toolchain and IDE (such as EWARM for IAR™, TrueSTUDIO® for Atollic IDE, or others).
4. Update the minimum heap/stack size to minimize in a first time the possible overflow (2 Kbytes minimum of heap is expected for the "AI Validation" test application).

Figure 34. Project settings view for IDE Code generator
5. Click on the [GENERATE CODE] button to generate the code corresponding to the current project configuration (including the IDE project files).

During the generation of the IDE project, the message box shown in Figure 35 can pop up if the user has selected and enabled an add-on AI application and forgotten to set the expected platform dependency (such as USART handle). Refer to Section 4.1 Adding the X-CUBE-AI component for details.

![Figure 35. AI peripheral not fully configured](image)

At this stage, the STM32CubeMX UI application can be closed. It is possible to re-open it later with the `<project_name>.ioc` file to enable and set a new peripheral, a middleware component, or both, or perform the **Validation on target** process.

5.2 Building and programming

When the IDE project is successfully generated by the STM32CubeMX tool, the standard build process is used to build and flash the STM32 board development kit or customer board:

1. Launch the IDE application and open the generated project file
2. Build and flash the firmware image. If the AI test application has been selected, no code modification or update is expected. Otherwise, user AI-based application code must be added to use the generated inference C API.
6  X-CUBE-AI internals

6.1  Graph flow and memory layout optimizer

The C-code generator optimizer engine seeks to optimize memory usage (RAM & ROM) against inference computing time (power consumption is also considered). It is based on a dataset-less approach, which means that no trained valid or test data-set is requested to apply the compression and optimization algorithms (no re-trained/refined weights/bias stage is expected to preserve the accuracy of the initial model).

- **Weight/bias compression** (targeted factor: none, x4, x8)
  - Only applicable for dense (or fully-connected) layer type
  - Weight-sharing-based algorithm is applied (K-means clustering)
  - If "none", the initial DL model accuracy is guaranteed. The residual error (~10^-08) is related to the native-model floating-point 64-bit size against the 32-bit C-floating-point size used. For large networks however, 10^-06 is more common.

![Figure 36. Weight/bias compression](image)

The advantage of this approach is to have a quick compression process, but the final result is not lossless and the global accuracy can be impacted. A "Validation" process of the generated C-model is provided as a mitigation to evaluate the generated error (refer to Section 6.2).

- **Operation fusing**
  - Merge two layers to optimize data placement and associated computing kernel. Some layers (like "Dropout", "Reshape") are removed during the conversion or optimization, and others (like nonlinearities and pooling after a convolutional layer) are fused in the previous layer. The effect is that the converted network has often a lower number of layers compared with the original network.

![Figure 37. Operation fusing](image)
• **Optimal activation/working memory**: A R/W chunk is defined to store temporary hidden layer values (outputs of the activation operators). It can be considered as a scratch buffer used by the inference function. The activation memory is reused across different layers. As a result, the activation buffer size is defined by the maximum memory requirements of two consecutive layers.

---

### 6.2 Validation engine (L2 relative error calculation)

A simple and quick validation mechanism is provided to compare the accuracy of a generated C-model and uploaded DL model from a numerical standpoint (refer to Figure 39). Both models are fed with the same input tensors (fixed random inputs or custom data-set; refer to [10]). The L2 relative error is then calculated for all inferences. The L2 relative error of the output layer below 0.01 indicates a valid generated C model. To be more accurate, as detailed in [10], additional metrics are reported to evaluate the generated C model. The X-CUBE-AI Expansion Package provides an inference DL executing engine for all supported DL frameworks. Note that it is still possible to consider the generated optimized C-model even if the tool reports a failed validation. The performance may not be aligned with the original Python™ model but the C-model can still be used. Further inspection is required, using, for example, a custom data set and NN output tracing.

![Figure 38. Optimal activation/working buffer](image)

![Figure 39. Validation flow overview](image)

The L2 relative error is computed as shown in Figure 40 with:

- $F_j$: flattened arrays of the C code layer output $j$
- $f_i$: flattened arrays of the corresponding original layer output $i$

$$
e_i = \frac{\| F_j - f_i \|}{\| F_j \|}$$

![Figure 40. L2 computation](image)
Two executing modes are provided:

- **Validation on desktop**: this mode allows the comparison of the DL model with its generated X86 C model. It runs on the host. The related output is illustrated in Section 4.5 Validating the generated C model.
- **Validation on target**: this mode compares the DL model with the C model that runs on the targeted device. It requires a special AI test application that embeds the generated NN libraries and the COM agent to communicate with the host system. Output and usage are illustrated in Section 10 AI validation application.

**Validation on target features:**

- Automatic detection of the connected STM32 boards
- The signature of the embedded generated C model is checked with the validated DL model
- The L2 error is only calculated on the last output layer (because of the COM speed to upload the data)
- Additional information is reported such as inference executing time by layer or others (refer to Section 10 AI validation application)

**Figure 41. Validation on target**
7 Generated STM32 NN library

Only the specialized (DL model dependent) C files are generated for each imported DL model. The name of these files are prefixed with the network name provided by the user (refer to Section 4.3 Uploading a pre-trained DL model file). They are based on an internal and private API implemented by the network_runtime.a library:

- `<name>.c` and `<name>.h` files for the topology
- `<name>_data.c` and `<name>_data.h` files for the weights/bias

Note: Generated specialized data and network files are common to all toolchains and STM32 MCU series.

The network_runtime.a library or NN computing kernel library is provided as a static library:

- All unused symbols and methods are removed at link stage
- Not based on a network-graph interpreter approach (like ARM-NN, lite deployment environment) since it is not optimal for devices with limited memory resources.

7.1 Firmware integration

Figure 42 illustrates the MCU integration model and view (including run-time dependencies) of the generated STM32 NN package.

For the application layer, the exported NN library is considered as a “black box” or self-content object. Only the specialized files, network topology (`<name>.c` and `<name>.h` files), and weights/bias parameters (`<name>_data.c` and `<name>_data.h` files) are provided as source files. They are based on a common network run-time library (`network_runtime.a`). The dependencies with the system run time are minimal:

- standard libc memory manipulation functions (`memcpy`, `memset`). They are generally provided by the MCU toolchain.
- CMSIS library to support the Cortex®-M optimized operations (FPU and DSP instructions). It is part of the STM32 HAL package.
- `malloc/free` is currently expected to support the “Recurrent-type” layer (“GRU” and “LSTM” layers). In future releases, the use of a static buffer allocation approach is planned instead. Performance impact is mitigated by the number of recurrent cell units and associated processing time.
• A mathematical library (with DSP/FPU support) is also requested to support the `expf`, `powf`, `tanhf`, and `sqrtf` functions.
• A minimal stack is requested (real value can be measured with the "AI system performance" application; refer to Section 9 AI system performance application).

Note: All external dependencies must be solved during the end-user link stage (firmware image generation). Activation memory buffers can be allocated dynamically in the heap or as a global array (.bss and .data sections). Refer to the `ai_<name>_init()` function to show how to pass the weights/bias buffer and activation memory buffers to the NN core library.

7.2 Library source tree view

When the IDE project is created, the network runtime library is exported into sub-folder `<project_name>/Middlewares/ST/AI/`. Specialized or dedicated NN files are stored in standard STM32CubeMX Inc and Src sub-folders. Requested specific files from the CMSIS-DSP libraries are also added.

```
...
<project_name>
|  |- Inc
|  |  |- app_x-cube-ai.h /* entry points - MX_X_CUBE_AI_xx() fcts */
|  |  |- bsp_ai.h /* BSP AI adapt. for AI validation/systemperf application */
|  |  |- constants_ai.h /* BSP constant AI definition */
|  |  |- <name_1>_h /* specialized NN files */
|  |  |- <name_1>_data.h
|  |  |- <name_2>_h
|  |  |  ...
|  |- Src
|  |  |- app_x-cube-ai.c
|  |  |- <name_1>.c /* specialized NN files */
|  |  |- <name_1>_data.c
|  |  |  ...
|  |  ...
| |--Middlewares
|   `- ST/AI
|      |-- include
|      |  `- *.h /* Internal/private AI headers */
|      |-- lib
|      |  `- network_runtime.a /* generic run-time library */
|      \-- Application
|         |-- SystemPerformance /* generic sample application */
|         |  |- Inc
|         |  |  `- aiSystemPerformance.h
|         |  `- Src
|         |  `- aiSystemPerformance.c
...
```

The file name `network_runtime.a` depends on the X-CUBE-AI version and IDE used, such as `NetworkRuntime410_CM4_GCC.a` or `NetworkRuntime410_CM4_IAR.a`.

7.3 Multi-network inference API

The `app_x-cube-ai.c` and `app_x-cube-ai.h` files provide also a generic multi-network inference API, which can be used by the AI client application. It is very close to the native embedded inference client API (refer to [9]); only the `create()` function is different. The C-name string of the network must be passed to create the instance of the underlying network. This interface is mainly used by the add-on AI test applications to have a generic way to address the different embedded networks.
7.4 Re-entrance and thread safety considerations

No internal synchronization mechanism is implemented to protect the entry points against concurrent accesses. If the API is used in a multi-threaded context, the protection of the instantiated NN(s) must be guaranteed by the application layer itself.

To minimize the usage of the RAM, a same activation memory chunk (SizeSHARED) can be used to support multiple network. In this case, the user must guarantee that an on-going inference execution cannot be preempted by the execution of another network.

\[
\text{SizeSHARED} = \text{MAX(AI\_name\_DATA\_ACTIVATIONS\_SIZE)} \quad \text{for name = "net1" ... "net2"}
\]

Note: If the preemption is expected for real-time constraint or latency reasons, each network instance must have its own and private activation buffer.

7.5 Code and data placement considerations

For the current STM32 memory architecture (STM32L4/STM32F4/STM32F3-based and STM32F7/STM32H7-based), no specific data or code placement is expected for performance reason. The Flash ART peripheral and the Arm® core sub-system cache (Cortex®-M7-based architecture) efficiently limit memory latency side effects. NN code (.text section) and RO data (.rodata section) can be placed in the internal Flash area. RW data (.data and .bss sections) must be placed in the embedded SRAM. The client stack is used; it must be placed in a zero-wait-state memory.

Note: There is no memory retention requirement on the activation buffer. It can be really considered as a scratch or working buffer. Between two inferences, buffer can be reused for pre-processing purpose for example, or the associated memory device can be switched off when the system goes into Deep Sleep.

7.6 Debug considerations

The library must be considered as an optimized black box in binary format (sources files are not deliveries). There is no support for run-time internal data or state introspection. Mapping and port of the NN is guaranteed by the X-CUBE-AI generator. Some integration issues can be highlighted by the ai\_name\_get_error() function.
8 Embedded inference client API

To use the generated NN code, a simple embedded inference client API is generated (see the `ai_<name>_XX()` functions in Figure 42). It is part of the `<project_name>/Src/<name>.h` file. All functions and macros are generated according to the C-network name provided. For usage and detailed description, refer to [9].
9 AI system performance application

The AI system performance application is a self and bare-metal on-device application, which allows the out-of-the-box measurement of the critical system integration aspects of the generated NN. The accuracy performance aspect is not and cannot be considered here. The reported measurements are:

- CPU cycles by inference (duration in ms, CPU cycles, CPU workload)
- Used stack and used heap (in bytes)

Execute the following series of steps in sequence to run the application:

1. Open and configure a host serial terminal console connected via a COM port (usually supported by a Virtual COM port over a USB connection, such as an ST-LINK/V2 feature).
2. Set the COM setting. It must be aligned with the setting of the STM32 USART (refer to Section 3.2 Hardware and software platform settings):
   - 115200 bauds
   - 8 bits
   - 1 stop bit
   - No parity
3. Reset the board to launch the application

When the application is running, typing `p` or `P` in the console suspends the main loop. The application embeds a minimal interactive console, which supports the following commands:

Possible key for the interactive console:
- `[q,Q]`      quit the application
- `[r,R]`      re-start (NN de-init and re-init)
- `[p,P]`      pause
- `[h,H,?]`    this information
- `xx`        continue immediately

9.1 System run-time information

Figure 43 and Figure 44 show the first part of the log, which indicates the useful information of the STM32 runtime or executing environment for the Keil® and Atollic IDEs respectively: device ID, system clock value, used toolchain, and others.

Figure 43. System run-time information - Keil® IDE
9.2 Embedded C-model network information

This second part shown in Figure 45 indicates the main static characteristics of the generated NN(s). In particular, it provides the RAM/Flash size (in bytes, respectively activation/weights fields) and the logical complexity (MACC, complexity field). Shape definitions of the input and output tensors are also reported. These informations are available also by the client application code through the \texttt{ai\_name\_get\_info()} client API function.

Figure 45. C-model network information
9.3 Embedded C-model run-time performance

As illustrated in Figure 46 and Figure 47, the last part of the log (main loop) reports the measured out-of-the-box system performance. Random inputs are injected in the network to measure the number of CPU cycles by inference (CPU cycles). The CPU workload and cycles/MACC are deduced from this value. During the measurement, the IRQs are masked.

- duration indicates the duration in ms for one inference.
- CPU cycles indicates the number of CPU cycles for one inference.
- CPU workload indicator corresponding to the associated CPU workload during 1 s.
- cycles/MACC is the number of CPU cycles by MACC operation.

**Figure 46. C-model run-time performance**

| Results for "net1", 16 inferences @216MHz/216MHz (complexity: 874970 MACC) |
| Duration: 28.847 ms (average) |
| CPU cycles: 6058169 -198595/+29532 (average,-/+)|
| CPU workload: 2% |
| cycles/MACC: 6 (average for all layers) |
| used stack: 352 bytes |
| used heap: 0:0 0:0 <req:allocated,req:released> cfg=0 |

**Figure 47. C-model run-time performance with heap and stack checking**

| Results for "net2", 16 inferences @216MHz/216MHz (complexity: 4550024 MACC) |
| Duration: 150.323 ms (average) |
| CPU cycles: 32469972 -42110/+14510 (average,-/+)|
| CPU workload: 15% |
| cycles/MACC: 7 (average for all layers) |
| used stack: 352 bytes |
| used heap: 0:0 0:0 <req:allocated,req:released> cfg=0 |

**Note:** "used heap" indicates the number of malloc() and cumulated allocated size (respectively free()) requested during the execution of all inferences. The counter is not reset between two inferences or test iterations to detect hypothetic memory leak. In the present case, the minimum heap size is 29568 / #iter = ~2Kbytes.

**Caution:** Today, the heap monitor is only supported for a GCC-based environment.
10  

AI validation application

The AI validation application is a self and bare-metal on-device application, which supports the Validation on device as presented in Section 6.2 Validation engine (L2 relative error calculation). It provides a USART-based interface with the host to export the inference API.

The whole AI validation application is either generated and programmed as a normal IDE project, or compiled automatically and downloaded from STM32CubeMX itself (refer to Figure 48).

If this application is not generated automatically, the following steps are requested:
1. When the board is programmed, reset or restart it.
2. Reopen (if closed) the ioc file with which firmware was generated.
3. Go to the AI configuration panel and open the <network_name> tab, which must validated.
4. Click on the [Validation on target] button to start the validation process. Before clicking on the [OK] button, the user has the possibility to indicate the host COM port that is used as shown in Figure 48. Otherwise, all available COM ports are discovered to detect a valid connected STM32 board (the first board found is used).

Figure 48. Host COM port selector for validation on device

As for Validation on desktop, the final result is reported in the [Validation status] field. More detailed information is reported in the UI log console.

10.1 System run-time information

The first part of the reported log shown in Figure 49 indicates the main system information: device ID, clock frequency, memory sub-system configuration, list of the embedded networks. For the validated network, shape-in and shape-out tensor description is provided as well as used AI tools versions.

Figure 49. System run-time information
10.2 Embedded C-model run-time performance

The second part of the log shown in Figure 50 reports the out-of-the-box system performance measurements (duration average executing time by inference). cycles/MACC is deducted from the duration value. During the measurement, the IRQs are masked.

- duration indicates the duration in ms for one inference.
- CPU cycles indicates the number of CPU cycles for one inference.
- cycles/MACC is the number of CPU cycles by MACC operation.

Figure 50. C-model run-time performance

```
#   #
...... 10/10
RMW State : batches=10 dur=24.266 ms 1.491KiB/s (wb=10.547KiB, sb=240B)
Results for 10 inference(s) @72/72MHz (mac=274970)
    duration    : 70.046 ms (average)
    CPU cycles  : 5676924 (average)
    cycles/MACC : 6.49 (average for all layers)
```

10.3 Layer-by-layer run-time performance

The next part of the log shown in Figure 51 provides the additional information about the generated C model: name and type of the implemented C layer (Clayer/id/desc), output shape (oshape), and average executing time by inference (ms).

Figure 51. Layer-by-layer results - Validation on target

```
Inspector report (layer by layer)
signature  : EF7C5473
n_nodes     : 7
num_inferences : 10

<table>
<thead>
<tr>
<th>Clayer_id</th>
<th>desc</th>
<th>oshape</th>
<th>ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>10011/(Merged Conv2d / Pool)</td>
<td>(10, 44, 1, 128)</td>
<td>24.277</td>
</tr>
<tr>
<td>1 4</td>
<td>10005/(Dense)</td>
<td>(10, 1, 1, 128)</td>
<td>53.165</td>
</tr>
<tr>
<td>2 9</td>
<td>10009/(Nonlinearity)</td>
<td>(10, 1, 1, 128)</td>
<td>0.010</td>
</tr>
<tr>
<td>3 5</td>
<td>10005/(Dense)</td>
<td>(10, 1, 1, 128)</td>
<td>1.303</td>
</tr>
<tr>
<td>4 5</td>
<td>10009/(Nonlinearity)</td>
<td>(10, 1, 1, 128)</td>
<td>0.010</td>
</tr>
<tr>
<td>5 6</td>
<td>10005/(Dense)</td>
<td>(10, 1, 1, 6)</td>
<td>0.066</td>
</tr>
<tr>
<td>6 6</td>
<td>10014/(Softmax)</td>
<td>(10, 1, 1, 6)</td>
<td>0.015</td>
</tr>
</tbody>
</table>
```

10.4 Final result for validation on target

The last part of the log shown in Figure 52 provides the final result of the validation process. It is similar to the result of the validation on desktop but only the L2 error on the last layer is reported (refer to Section 6.2 Validation engine (L2 relative error calculation)).
10.5 Returned error during the connection

The following USART setting is used by default:

- 8 bits
- 1 stop bit
- No parity
- 115200 bauds

If redefined by the user, the baud rate must be kept aligned across all settings (refer to Section 3.2 Hardware and software platform settings and Figure 48. Host COM port selector for validation on device).

10.5.1 Error: no connected board, invalid firmware, or board restart needed

Indicates that no board is connected or can be found, or that firmware is not the expected "AI Validation" firmware. This error can also indicate an incoherent firmware state, in which case the board must be restarted.

To check that the firmware is correctly programmed, open a host serial terminal console at boot time, which generates an ASCII-based log. Do not forget to close the connection before to launching the Validation on target process again.
10.5.2 Error: “network_name” is not a valid network
Indicates that the expected C model identified by its name is not available in the connected board. See the UI log console (Outputs window) for more details.

10.5.3 Error: the embedded STM32 model does not match the C model
Indicates that the signature of the generated C model is not coherent with the expected model. The parameters used to check the signature are:

- RAM/ROM size
- MACC
- Number of nodes
- Tool versions

See the UI log console (Outputs window) for more details.
11 AI template application

When selected, the generated IDE project is not really a complete AI template application with an example of a basic AI application to use the generated C models. `aiSystemPerformance.h` and `aiSystemPerformance.c` files represent a good example for this purpose. Only the specific generated files are generated. This can be used as started point to develop an initial bare-metal application with two simple entry points (`init` and `process` functions). Refer to [9] for details.
The X-CUBE-AI core currently supports the following DL toolboxes:

- Keras: //keras.io/
- Caffe: //caffe.berkeleyvision.org/
- ConvNetJs: //cs.stanford.edu/people/karpathy/convnetjs/
- TensorFlow™ Lite: www.tensorflow.org/lite/

**Note:** TensorFlow is a trademark of Google Inc.

For each toolbox, only a subset of all possible layers and layer parameters are supported, depending on the expressive power of the network C API, and on the parser for the specific toolbox. Supported configurations are detailed in [8].
13 Error handling

The X-CUBE-AI core handles a range of different errors and reports them to the user as detailed in [7].
14 FAQs

14.1 Log files for debug purpose?
When a “validation on target” process is performed, all messages exchanged with the target (including the data) are stored in a dedicated log file:
C:\Users\<username>\stm32cubemx\ai_stm32_msg.log
If a validation or generation process fails, additional debug/log info is available in file:
C:\Users\<username>\stm32cubemx\STM32CubeMX.log

14.2 Multi-network limitations?
There is no real limitation to use multiple networks other than RAM and Flash availability. Each network has its own set of ai_<name>_XXX() inference functions (refer to Section 8 Embedded inference client API). If the activation memory chunk is shared between multiple networks, care must be taken. No preemption is allowed (refer to Section 7.4 Re-entrance and thread safety considerations).

14.3 Unable to compile file “arm_dot_prod_f32.c”
The compilation of arm_dot_prod_f32.c may fail when the IDE project files are regenerated:

```
compiling arm_dot_prod_f32.c...
../Drivers/CMSIS/Include/arm_math.h(314): error: #35: #error directive: "Define according the used Cortex core ARM_MATH_CM7, ARM_MATH_CM4, ARM_MATH_CM3, ARM_MATH_CM0PLUS or ARM_MATH_CM0"
#error "Define according the used Cortex core ARM_MATH_CM7, ARM_MATH_CM4, ARM_MATH_CM3, ARM_MATH_CM0PLUS or ARM_MATH_CM0"
../Drivers/CMSIS/DSP_Lib/Source/BasicMathFunctions/arm_dot_prod_f32.c: 0 warnings, 1 error
```

According to the targeted STM32 device, the following C defines must be redefined in the project setting:

```
ARM_MATH_CM7, __FPU_PRESENT=1
```

14.4 Used heap or stack: disabled or not yet supported
This log indicates that the stack (respectively heap) monitor is explicitly disabled or not yet implemented for the toolchain used.

<table>
<thead>
<tr>
<th>Toolchain</th>
<th>Stack monitor</th>
<th>Heap monitor</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCC</td>
<td>Supported</td>
<td>Supported</td>
<td>SW4STM32</td>
</tr>
<tr>
<td>IAR™ 8.x and 7.x</td>
<td>Supported</td>
<td>Not implemented</td>
<td>-</td>
</tr>
<tr>
<td>MDK-ARM</td>
<td>Not implemented</td>
<td>Not implemented</td>
<td>-</td>
</tr>
</tbody>
</table>

Example of heap and stack monitoring activation:
14.5 **Why is “used heap” always zero?**

For GCC-based projects only:

```
used heap : 0:0 0:0 (req:allocated,req:released) cfg=0
```

Such a result is not necessarily a problem. Most of the `network_runtime.a` library is based on a preallocated R/W buffer scheme (activation buffers). For some specific layers (recurrent layer type), the current implementation requests to allocate dynamically a part of these work buffers through the `malloc()` function.

The heap monitor is based on a toolchain specific mechanism, which allows the wrapping of the system `malloc()` and `free()` functions. To enable this wrapping, the `-Wl,--wrap=malloc -Wl,--wrap=free` miscellaneous linker options must be set in the build system as shown in Figure 54.

![Figure 54. Linker options to enable the heap monitor](image)

14.6 **Formatted floating-point numbers are empty for a GCC-based project**

The following link option must be added to output a formatted floating-point number:

```
-u _printf_float
```

14.7 **CPU cycles/MACC?**

Refer to Section 4.4.1 CPU cycles/MACC? and Section 9 AI system performance application.
14.8 Is it necessary to enable or configure a TIMER peripheral?
This is not necessary. The mechanism to measure the number of CPU cycles by inference uses a dedicated Arm® Cortex®-M debug unit (DWT: Data Watch-point and Trace unit), which is available on all supported STM32 devices. It uses a free-running counter that is clocked by the CPU clock (HCLK system clock).

14.9 How to update only the exported NN library in my generated project?
This is straightforward if the STM32CubeMX design guide lines are applied (/* USER CODE BEGIN*/,/*... END*/ tags) for the changes to the exported and generated files. The <project_name>.ioc file can be directly re-opened to upload a new NN model and to update the IDE project.

14.10 Is it possible to export an NN library for a non-STM32CubeMX-based project?
Since the exported NN library is located in a well-defined and self-content sub-folder (refer to Section 7 Generated STM32 NN library), this AI sub-folder can be fully copied in the source tree of the destination project:
1. Create a new dummy STM32CubeMX project for the user’s STM32 MCU device.
2. Generate the IDE project for the user’s toolchain/IDE. This step is requested to include the correct network_runtime.a library, which is toolchain- and Arm®-Cortex®-M-architecture dependent.
3. Copy the generated AI sub-folder in the source tree of the new project.
4. Add the network.c and network_data.c files, and the network_runtime.a library to the system build and update the C/C++ compiler and linker options as described in Section 7 Generated STM32 NN library.
5. Return to step 1 to update and evaluate a modified NN model.

14.11 Command-line interface?
A complete command-line interface is provided in the X-CUBE-AI Expansion Package (refer to [7]).
15 References and documentation

15.1 References used in this user manual

Table 5. References

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>[7]</td>
<td>X-CUBE-AI stm32ai command-line interface</td>
<td>Documentation embedded in X-CUBE-AI Expansion Package in</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Documentation/command_line_interface.html(1)</td>
</tr>
<tr>
<td>[8]</td>
<td>X-CUBE-AI supported toolbox and layers</td>
<td>Documentation embedded in X-CUBE-AI Expansion Package in</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Documentation/layer-support.html(1)</td>
</tr>
<tr>
<td>[9]</td>
<td>X-CUBE-AI embedded inference client API</td>
<td>Documentation embedded in X-CUBE-AI Expansion Package in</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Documentation/layer-support.html(1)</td>
</tr>
<tr>
<td>[10]</td>
<td>X-CUBE-AI evaluation report and metrics</td>
<td>Documentation embedded in X-CUBE-AI Expansion Package in</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Documentation/evaluation_metrics.html(1)</td>
</tr>
</tbody>
</table>

1. Refer to Section 15.2 Access to X-CUBE-AI in-package documentation.

15.2 Access to X-CUBE-AI in-package documentation

Follow the steps below for access to the documentation available in the X-CUBE-AI Expansion Package:
1. Hover the cursor over one of the X-CUBE-AI modes and click on [details and documentation]

Figure 55. In-package documentation access (1 of 2)
2. Click on [SW Pack documentation]

Figure 56. In-package documentation access (2 of 2)

3. A browser window opens that lists the available X-CUBE-AI documentation

Figure 57. In-package documentation index
## Table 6. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-Jan-2019</td>
<td>1</td>
<td>Initial release.</td>
</tr>
<tr>
<td>19-Jul-2019</td>
<td>2</td>
<td>Updated for X-CUBE-AI 4.0.0:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• added quantization</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• added the command-line interface</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• added the support of TensorFlow™ Lite</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Simplified the user manual referring to documents in the Expansion Package for:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Section 8 Embedded inference client API</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Section 11 AI template application</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Section 12 Supported toolboxes and layers for Deep Learning</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Section 13 Error handling</td>
</tr>
<tr>
<td>11-Oct-2019</td>
<td>3</td>
<td>Updated for X-CUBE-AI 4.1.0:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• added the support of the quantization model for TensorFlow™ Lite</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• added the support of external memories for validation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Simplified the user manual:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• removed Section 14.2 Custom data set file format?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• added Section 15.2 explaining how to get access to in-package</td>
</tr>
<tr>
<td></td>
<td></td>
<td>documentation</td>
</tr>
</tbody>
</table>
6.2 Validation engine (L2 relative error calculation) ........................................ 28

7 Generated STM32 NN library ................................................................. 30
  7.1 Firmware integration ........................................................................... 30
  7.2 Library source tree view ..................................................................... 31
  7.3 Multi-network inference API ............................................................... 31
  7.4 Re-entrance and thread safety considerations ....................................... 32
  7.5 Code and data placement considerations ............................................. 32
  7.6 Debug considerations ......................................................................... 32

8 Embedded inference client API .............................................................. 33

9 AI system performance application ....................................................... 34
  9.1 System run-time information ............................................................... 34
  9.2 Embedded C-model network information ............................................ 35
  9.3 Embedded C-model run-time performance .......................................... 36

10 AI validation application ...................................................................... 37
  10.1 System run-time information ............................................................. 37
  10.2 Embedded C-model run-time performance ......................................... 38
  10.3 Layer-by-layer run-time performance ................................................ 38
  10.4 Final result for validation on target .................................................... 38
  10.5 Returned error during the connection ................................................ 39
    10.5.1 Error: no connected board, invalid firmware, or board restart needed . 39
    10.5.2 Error: “network_name” is not a valid network ............................... 40
    10.5.3 Error: the embedded STM32 model does not match the C model .... 40

11 AI template application ......................................................................... 41

12 Supported toolboxes and layers for Deep Learning .............................. 42

13 Error handling ...................................................................................... 43

14 FAQs ..................................................................................................... 44
  14.1 Log files for debug purpose? .............................................................. 44
  14.2 Multi-network limitations? ................................................................. 44
  14.3 Unable to compile file “arm_dot_prod_f32.c” ...................................... 44
  14.4 Used heap or stack: disabled or not yet supported ............................. 44
List of tables

Table 1. Definition of terms used in this document ................................................... 6
Table 2. System informations reporting .................................................................. 19
Table 3. Metrics ...................................................................................................... 22
Table 4. Heap and stack monitoring support ............................................................ 44
Table 5. References .................................................................................................. 47
Table 6. Document revision history ......................................................................... 49
List of figures

Figure 1. X-CUBE-AI core engine .................................................. 3
Figure 2. X-CUBE-AI overview ..................................................... 4
Figure 3. Quantization flow ............................................................. 4
Figure 4. X-CUBE-AI core in STM32CubeMX ................................. 5
Figure 5. Managing embedded software packs in STM32CubeMX .... 7
Figure 6. Installing X-CUBE-AI in STM32CubeMX ......................... 7
Figure 7. X-CUBE-AI in STM32CubeMX ......................................... 8
Figure 8. Creating a new project ..................................................... 9
Figure 9. AI filter ........................................................................ 9
Figure 10. AI filter with default option ............................................ 10
Figure 11. AI filter with compression x4 ......................................... 10
Figure 12. NUCLEO-F746ZG board selection ............................... 11
Figure 13. Initialize all peripherals ............................................... 11
Figure 14. USART3 configuration ................................................. 12
Figure 15. Clock wizard pop-up .................................................... 12
Figure 16. System clock settings ................................................... 13
Figure 17. MCU memory sub-system (parameter settings) ............. 13
Figure 18. Enabling the CRC peripheral ....................................... 14
Figure 19. Additional software button ........................................... 15
Figure 20. Adding the X-CUBE-AI core component ....................... 15
Figure 21. Add-on X-CUBE-AI applications ................................ 15
Figure 22. Main X-CUBE-AI configuration panel ............................ 16
Figure 23. X-CUBE-AI platform setting panel ................................ 16
Figure 24. NN configuration wizard .............................................. 17
Figure 25. Insufficient RAM/Flash message box ........................... 17
Figure 26. Uploaded and analyzed DL model ................................. 18
Figure 27. Setting for external memories ...................................... 19
Figure 28. Integrated C-model (runtime-view) ............................... 20
Figure 29. Generated C-model graph .......................................... 21
Figure 30. Validation status field .................................................. 22
Figure 31. Validate on desktop - log report .................................. 23
Figure 32. Validation on target .................................................... 23
Figure 33. Main view with multiple networks ............................... 24
Figure 34. Project settings view for IDE Code generator ................ 25
Figure 35. AI peripheral not fully configured ............................... 26
Figure 36. Weight/bias compression ............................................ 27
Figure 37. Operation fusing ....................................................... 27
Figure 38. Optimal activation/working buffer ............................... 28
Figure 39. Validation flow overview ............................................ 28
Figure 40. L2 computation .......................................................... 28
Figure 41. Validation on target .................................................... 29
Figure 42. MCU integration model and view ................................. 30
Figure 43. System run-time information - Keil® IDE ...................... 34
Figure 44. System run-time information - Atollic IDE ................... 35
Figure 45. C-model network information ..................................... 35
Figure 46. C-model run-time performance ................................... 36
Figure 47. C-model run-time performance with heap and stack checking 36
Figure 48. Host COM port selector for validation on device ........... 37
Figure 49. System run-time information ....................................... 37
Figure 50. C-model run-time performance ................................... 38
Figure 51. Layer-by-layer results - Validation on target ............... 38
Figure 52. Final report for validation on target ..................................................... 39
Figure 53. AI valid - Initial log ........................................................................ 40
Figure 54. Linker options to enable the heap monitor ............................................ 45
Figure 55. In-package documentation access (1 of 2) ............................................. 47
Figure 56. In-package documentation access (2 of 2) ............................................ 48
Figure 57. In-package documentation index ......................................................... 48