UM3201

User manual

Getting started with Stellar Studio AI plugin for Artificial Intelligence (AI)

Introduction

This user manual delineates the procedural framework for constructing a comprehensive AI project tailored for Stellar microcontrollers within the Stellar Studio IDE environment. It encompasses a systematic approach to the automatic transposition of pre-trained neural networks (NN) into an efficiently optimized library and delineates the integration process within the project. The StellarStudio.AI module, a constituent fully compatible with the Stellar Studio suite, is the focal point of this documentation.

The core of this manual is a practical tutorial that guides users through the configuration of the StellarStudio.AI module. It provides detailed instructions on expediting the creation of a Stellar AI centric project. Users will learn to navigate the StellarStudio.AI's features and leverage its capabilities to streamline project development.

StellarStudio.AI is built upon the widely recognized ST Edge AI Core command-line interface (CLI) technology, known as ST Edge AI. This technology is specifically designed for deployment across various STMicroelectronics devices including STM32, Stellar MCUs and MEMS. Comprehensive documentation on the ST Edge AI Core technology is included within the final installation package for users seeking in-depth understanding and advanced operational guidance.

By following this manual, users will acquire the necessary knowledge to efficiently develop AI projects on Stellar microcontrollers, from the inception to the completion, utilizing the StellarStudio.AI component as an integral tool within the Stellar Studio ecosystem.

Figure 1. ST Edge AI Core CLI technology

1 Requirements

The StellarStudio.AI component requires the latest available Stellar Studio version (with the latest versions of the SDKs and TOOLS packages installed) for Windows operating system and it is compatible with the Stellar automotive Arm®-based device family.

In particular, the supported Stellar microcontrollers can be selected in the MCU target selection panel of the StellarStudio.AI component.

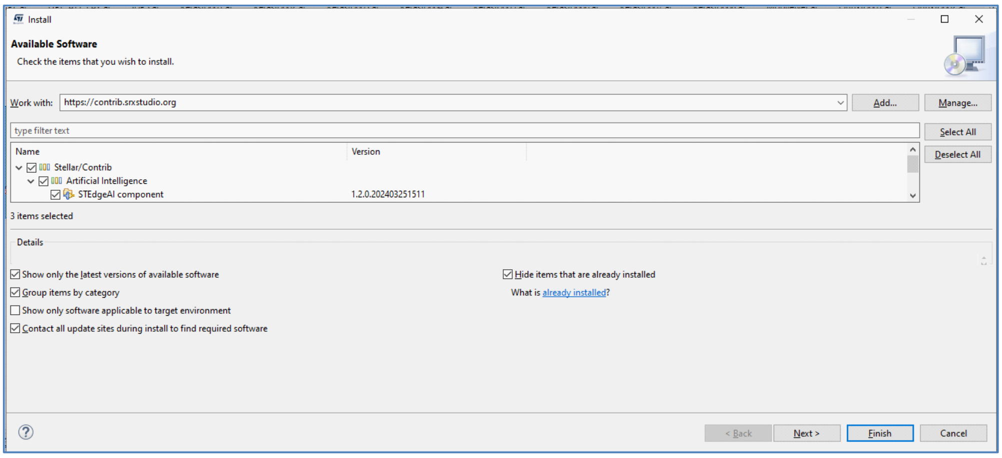
2 Overview

The StellarStudio.AI component extends the Stellar Studio by providing an automatic NN library generator optimized in computation and memory (RAM and flash memory) that converts pre-trained neural networks from the most used DL frameworks (such as Keras, TensorFlow™ lite, and ONNX) into a library that is automatically integrated in the final Stellar Studio software development kit (Stellar SDK). The project is automatically set up, ready for compilation and execution on the Stellar microcontrollers.

The StellarStudio.AI component can be easily installed in few steps directly from the Stellar Studio tool:

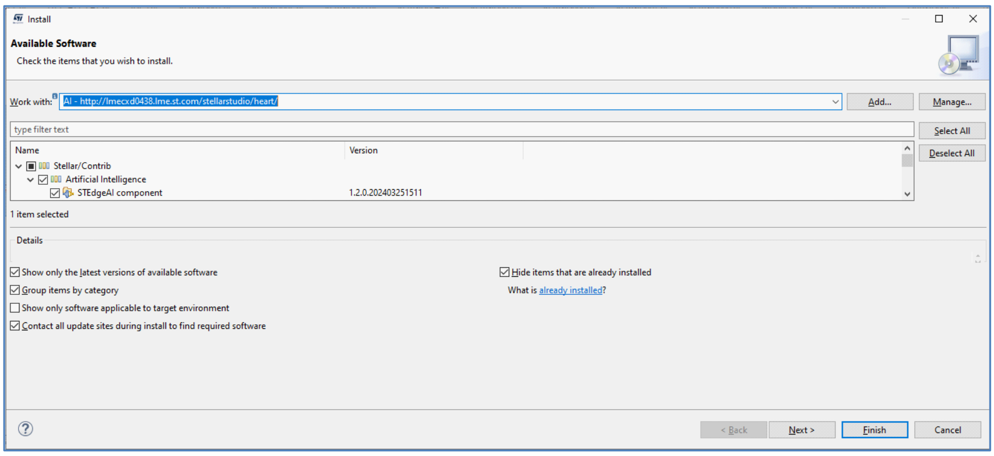
1. From the menu, select [Help]>[Install new software…]
   a. For external users, type https://contrib.srxstudio.org and press *Next*.

Figure 2. StellarStudio.AI external installation



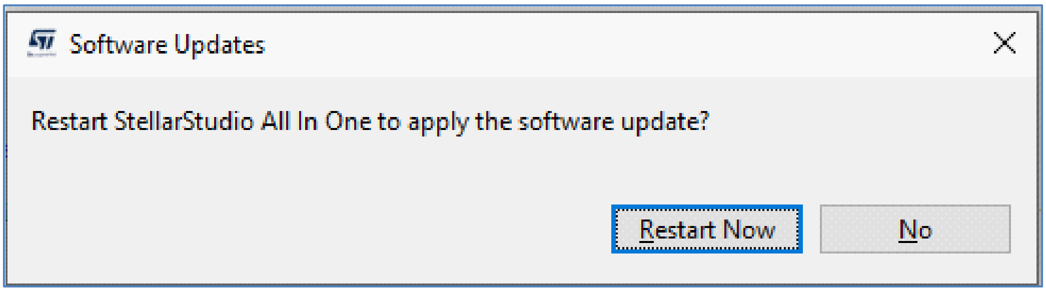
   b. For internal users, type the latest available installation link (for example, http:// lmecxd0438.lme.st.com/ stellarstudio/journey) and press *Next*.

Figure 3. StellarStudio.AI internal installation



2. Press *Finish* to start the installation.
3. At the end of the installation a Stellar Studio restart is required.

Figure 4. StellarStudio.AI installation restart



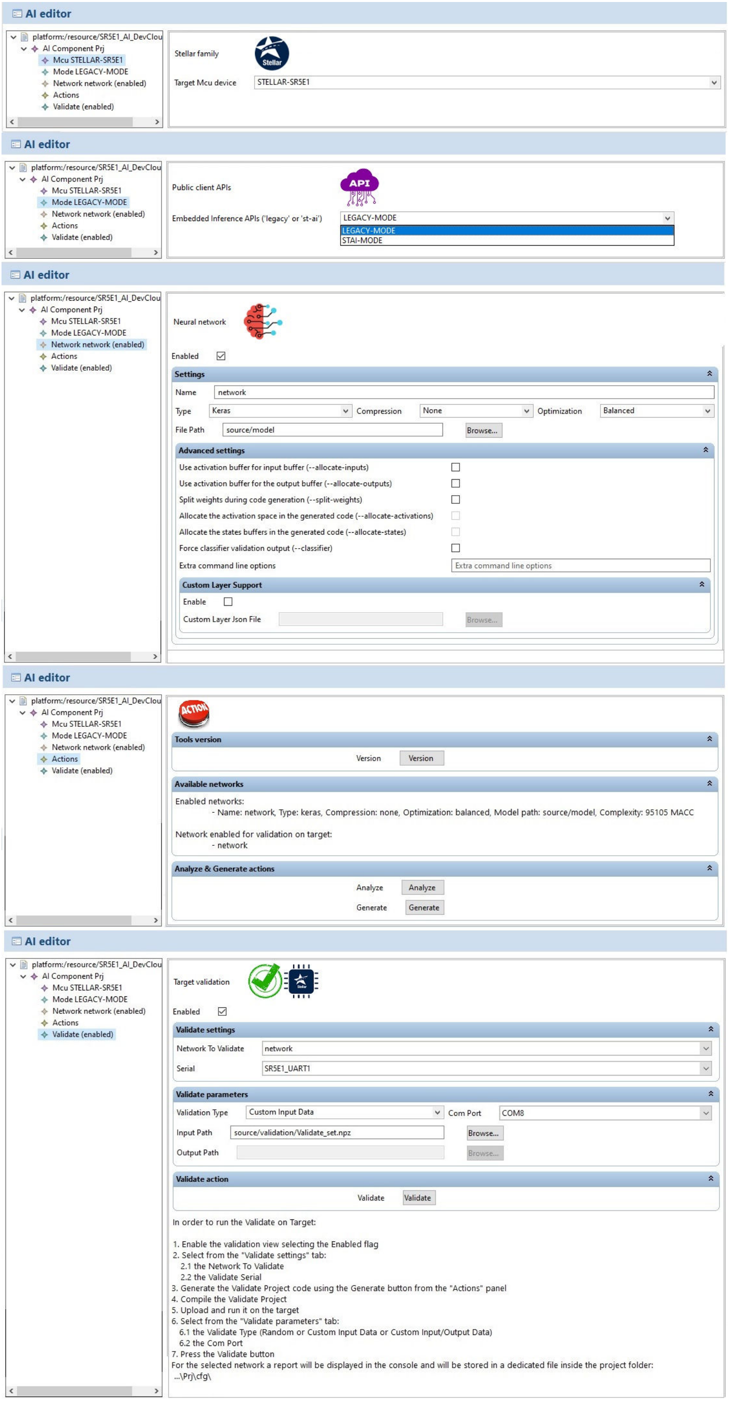
The StellarStudio.AI component can generate two kinds of project:

- Validation project that validates incrementally the results returned by the NN, stimulated by either random or user test data, for the Stellar device selected in the MCU target selection panel.
- Application template project allowing the building of AI-based applications.

It is based on a graphical user interface (GUI) that allows to define a neural network on which the SDK project is based and the type of processing (analyze, generate, or validate) to execute. The project, after generating the neural networks files, can be compiled and flashed on the board to validate the neural network. When the validation on target is successful, the neural network files can be used in any user application to run inferences.

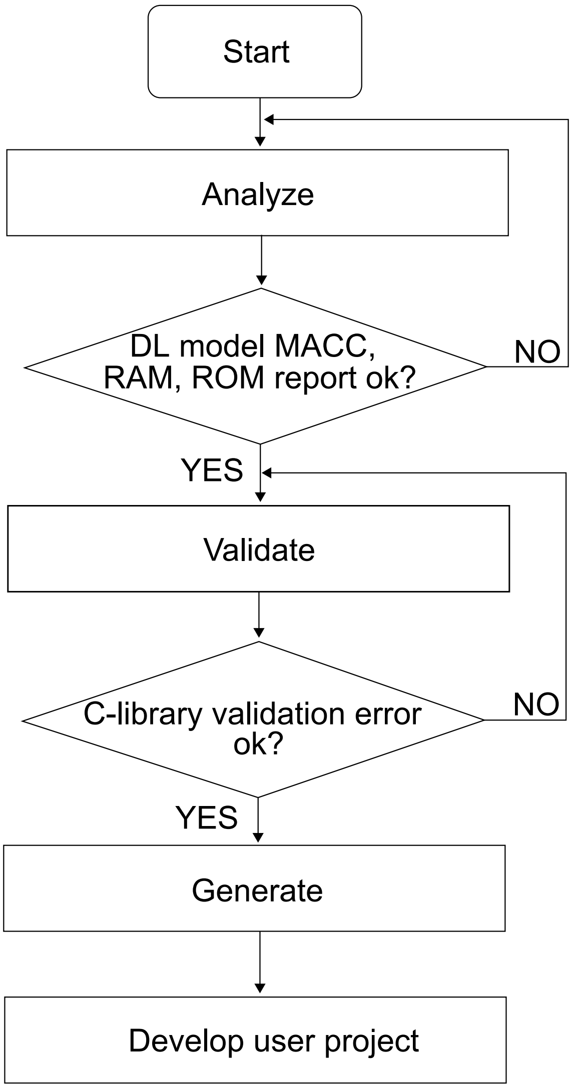
The StellarStudio.AI configuration is based on the following panels (MCU, mode, network, actions, and validate) as shown in Figure 5. ST Edge AI Core - StellarStudio.AI GUI.

Figure 5. ST Edge AI Core - StellarStudio.AI GUI



The next figure shows a typical workflow for developing a user-defined application based on neural networks using the StellarStudio.AI component.

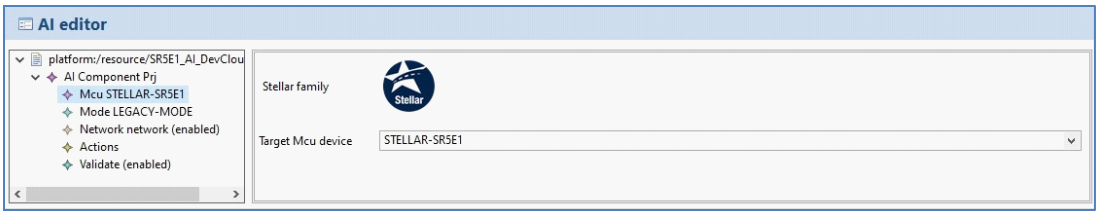
Figure 6. ST Edge AI Core - StellarStudio.AI workflow



3 MCU parameters

The MCU panel allows you to specify which Stellar target device to deploy the AI application: SR5 and SR6 devices can be selected.

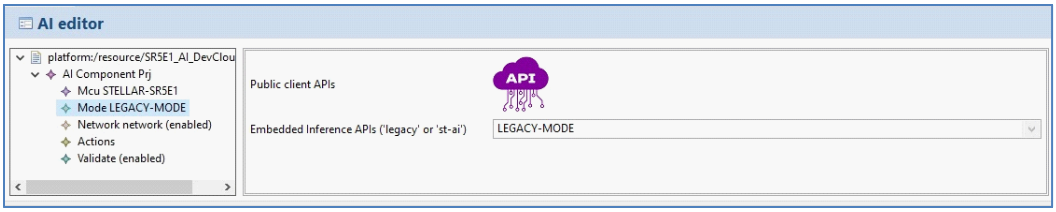
Figure 7. **MCU parameters**



4 Mode parameters

The Mode panel allows to specify which embedded public C-API is generated at runtime: standard legacy vs st-ai.
By default, legacy mode is used.
Depending on this value, the files generated are different.
For a full description of this CLI option (--c-api), you can refer to the ST Edge AI Core technology documentation.

Figure 8. Mode parameters



5 Network parameters

The network panel allows specifying one or more networks on which the StellarStudio.AI component is based on each network defined in the network panel and it can be enabled or disabled. When a network is disabled, it is not processed. In this way the user can temporarily remove one or more networks from the StellarStudio.AI process. But at least one network in the network panel must be enabled anytime.

For each new entry in the network panel, it is possible to specify the following options:

- **Enabled:** the flag to enable or disable the neural network.
- **Name:** the name of the neural network. Two or more networks with the same name can be defined in the network list, but only one can be enabled. If more networks with the same name are enabled at the same time, an error is shown. Please note that two names that differ only for the lower case/upper case of one or more characters are considered identical.
  This name is used to generate the C neural networks file names and functions.
  For a full description of this CLI option (-n/--name), you can refer to the ST Edge AI Core technology documentation.
- **Type:** the type of the deep learning (DL) framework. The following types are supported in the StellarStudio.AI component:
  - Keras
  - TensorFlow lite
  - ONNX

For a list of the layers supported, you can refer to the ST Edge AI Core technology documentation.

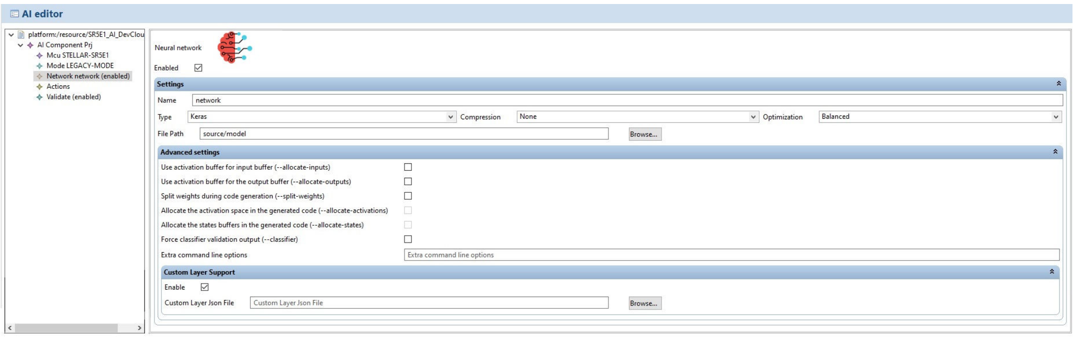
- **Compression:** the expected global factor of compression to reduce the size of the deployed c-model. Only the weights for the floating-point dense or fully connected layers are considered.
  Supported values are:
  - None: no compression, default value.
  - Lossless: applied algorithms ensuring the accuracy (structural compression).
  - Low: applied algorithms trying to reduce the size of the parameters with a minimum of accuracy loss.
  - Medium: more aggressive algorithms, the final accuracy loss can be more important.
  - High: extreme aggressive algorithms (not used).
    For a full description of this CLI option (-c/--compression), you can refer to the STMicroelectronics Edge AI Core technology documentation.
- **Optimization:** it is used to indicate the objective of the optimization passes, which are applied to deploy the c-model. Note that the accuracy/precision of the generated model is not impacted. By default, a trade-off (balanced value) is considered.
  Supported values are:
  - Time: applied the optimization passes to reduce the inference time (or latency). In this case, the size of the used RAM (activation buffer) can be impacted.
  - Ram: applied the optimization passes to reduce the RAM used for the activations. In this case, the inference time can be impacted.
  - Balanced: trade-off between the 'Time' and the 'Ram' objectives.
    For a full description of this CLI option (-O/--optimization), you can refer to the ST Edge AI Core technology documentation.
- **File path:** the path in which the model files must be stored. If the model file path is not valid or the folder does not contain any valid model file, the StellarStudio.AI processing is stopped, and an error is returned.
  For a full description of this CLI option (-m/--model), you can refer to the ST Edge AI Core technology documentation.
- **Allocate inputs:** it's enabled by default in the ST Edge AI Core CLI command line and it indicates that the activation buffer is also used to handle the input buffers, else, they should be allocated separately in the user memory space. Depending on the size of the input data, the activation buffer may be bigger but overall less than the sum of the activation buffer plus the input buffer.
  For a full description of this CLI option, you can refer to the ST Edge AI Core technology documentation.
- **Allocate outputs:** it's enabled by default in the ST Edge AI Core CLI command line and it indicates that the activation buffer is also used to handle the output buffers, else, they should be allocated separately in the user memory space.
  For a full description of this CLI option, you can refer to the ST Edge AI Core technology documentation.

- **Split weights:** if enabled, this flag indicates that one c-array is generated by the weights/bias data tensor instead of having a unique C-array (weights buffer) for the whole.
  For a full description of this CLI option (--split-weights), you can refer to the ST Edge AI Core technology documentation.
- **Allocate activations:** if enabled, this flag allocates the activation buffers (to store the intermediate results) in the generated code (used only when the st-ai mode is selected). Moreover, in case of default behavior, they should be allocated separately in the user memory space.
  For a full description of this CLI option (--allocate-activations), you can refer to the ST Edge AI Core technology documentation.
- **Classifier:** if enabled, this flag indicates that the provided model should be considered as a classifier vs regressor. This implies that the computation of the “CM” and “ACC” metrics are evaluated, and an autodetection mechanism is used to evaluate if the model is a classifier or not.
  For a full description of this CLI option (--classifier), you can refer to the ST Edge AI Core technology documentation.
- **Extra options:** This field must be used when running the ST Edge AI Core CLI commands.
  For a full list of all CLI options, you can refer to the ST Edge AI Core technology documentation.
- **Custom Layer**
  - Enabled: to enable a custom layer (json file format).
  - Custom layer Json file: the path of the configuration file (.json file) to support the custom layers. The new .c file is automatically generated during the generation phase and built during the compilation phase.

For a full description of this CLI option (--custom), you can refer to the ST Edge AI Core technology documentation.

The figure below shows the network panel parameters.

Figure 9. Neural networks parameters



The table below summarizes the model file extensions for the different network types.

Table 1. Neural network model file extensions

DL framework	Type	File extension
Keras	Keras	One file: .h5 or .hdf5 or .keras Two files: (.h5 or .hdf5) and .json
TensorFlow lite	TFLite	.tflite
ONNX	Onnx	.onnx

6 AI commands

The StellarStudio.AI component supports the following commands (refer to the ST Edge AI Core CLI technology documentation):

- **Version**
  - Get the version of all AI tools
- **Analyze**
  - Import the model.
  - Map, render, and optimize internally the model.
  - Log and display a report.
- **Generate**
  - Import the model.
  - Map, render, and optimize internally the model.
  - Export the specialized C-files.
  - Log and display a report.
- **Validate**
  - Import the model.
  - Map, render, and optimize internally the model.
  - Execute the generated C-model (on a target).
  - Execute the original model using the original deep learning runtime framework for x86.
  - Evaluate the metrics.
  - Log and display a report.

Pushing the version command, the ST Edge AI Core tools versions are returned.

For the other commands, the same preliminary steps are applied. A report (.txt file) is systematically created and fully or partially displayed. Additional JSON files (dictionary based) are generated in the workspace directory to be parsed by the StellarStudio.AI component to retrieve the results. Note that they can also be used in a non-regression environment. The format of these files is out of the scope of this document.

```
<workspace-directory-path>\<network_name>_report.json, <network_name>_c_graph.json
```

```
<output-directory-path>\<network_name>_<cmd>_report.txt
```

Version, Analyze and Generate commands are inside the actions panel, see Figure 10. Version, analyze, and generate commands.

The Validate command is inside the validate panel, see Figure 11. Validate command.

The Analyze command is the primary command to import, parse, check, and render an uploaded pre-trained model. A detailed report provides the main system metrics to know if the generated code can be deployed on the target Stellar device. It also includes rendering information by layer or/and operator. After completion, the user can be fully confident of the imported model in terms of supported layer/operators.

The Generate command is used to generate specialized networks and data C-files. They are generated in the specified output directory. The files generated depend on the embedded public C-API selected in the mode parameter, see Figure 8. Mode parameters. The name of the files generated depends on the neural network name parameter chosen in the network panel.

The Validate command allows to import, render, and validate the generated C-files. For the validation on target, the target board must be flashed with a valid validation firmware.

When the validation on target is performed, the DL model is compared with the C model that runs on the targeted device. It requires a special StellarStudio.AI test application that embeds the generated neural networks libraries and the COM agent to communicate with the host system.

Be aware, that the main purpose of the underlying validation process is to test the generated C files with the associated network runtime library by comparison with the imported DL model.

Subsequently, only a representative and limited part of a whole validation or test dataset can be used. It has not been designed to validate the pre-trained model as during a training/test phase.

Figure 10. **Version, analyze, and generate commands**

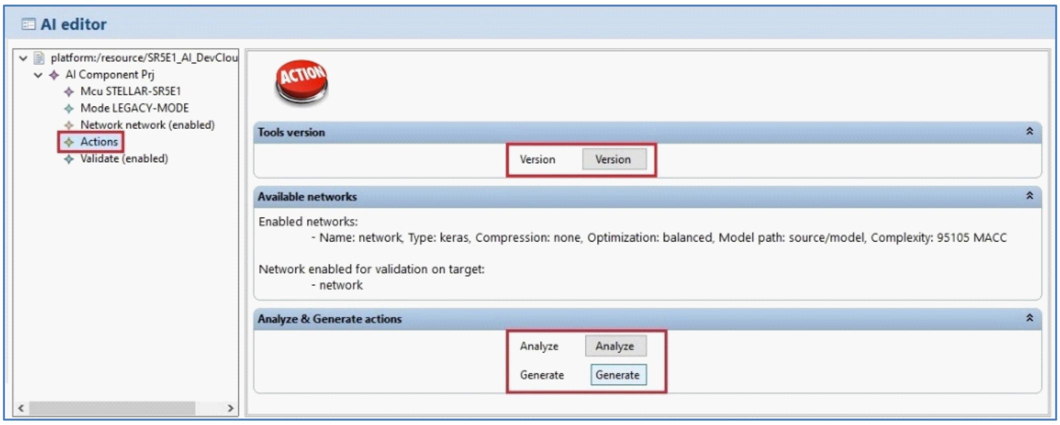
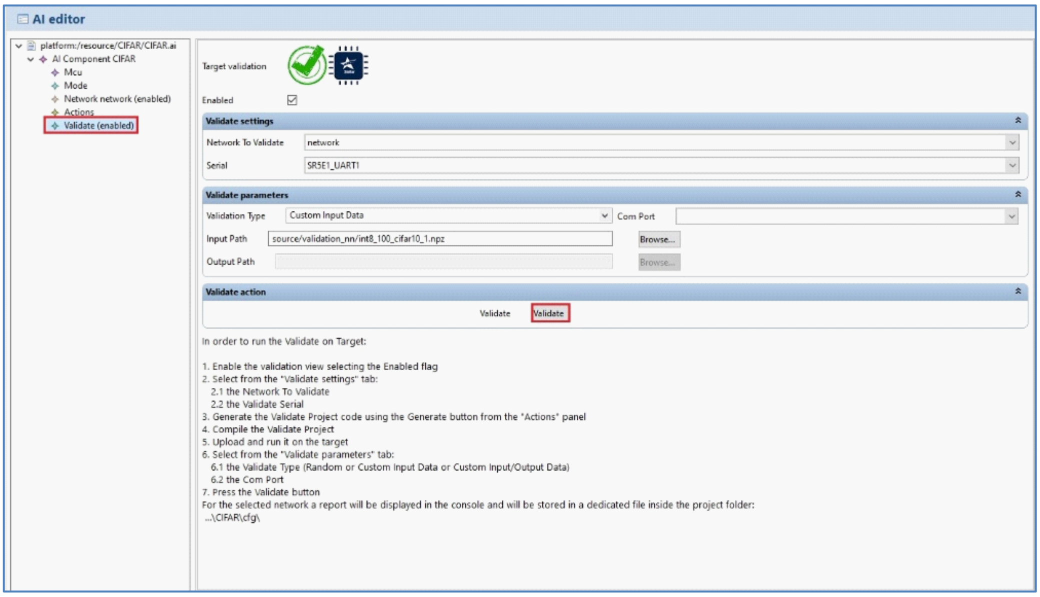


Figure 11. **Validate command**

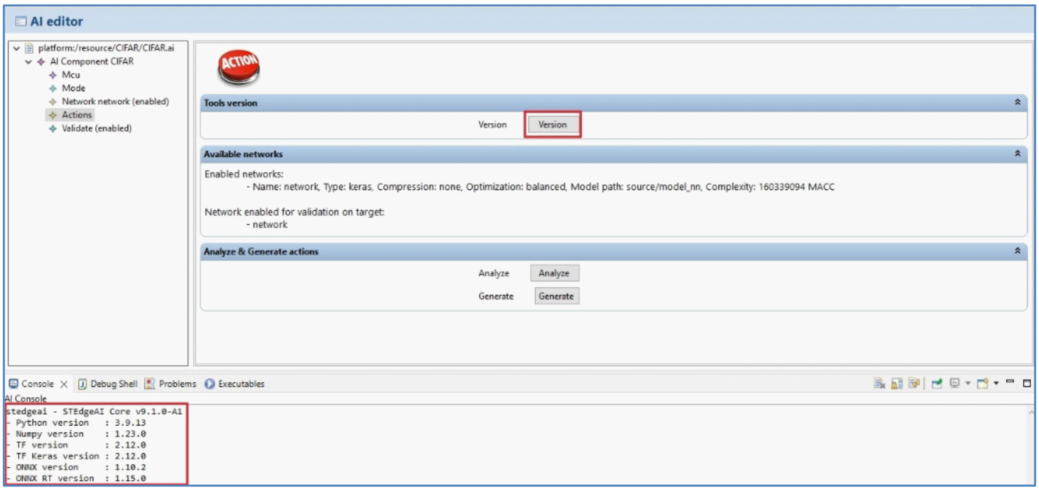


6.1 Version

The Version command returns the information of the ST Edge AI Core tools version and is displayed on the AI console, as the figure below.

For a full description of this CLI option (--tools-version), you can refer to the ST Edge AI Core technology documentation.

Figure 12. **Version process**



6.2 Analyze

The Analyze command is used to check a DL model. For each of the enabled networks in the network panel it generates a report that is shown in the Stellar Studio console during the command execution and is also stored in a .txt file within the project folder <prj>/<ai_component_name>/cfg/. The name of the report is <network_name>_analyze_report.txt.

For a full description of the CLI Analyze command, you can refer to the ST Edge AI Core technology documentation.

The report allows us to check the imported models in terms of supported layers/operators. To run the analyze command, select the actions panel and click on the analyze button (see the figure below).

Figure 13. **Analyze process**

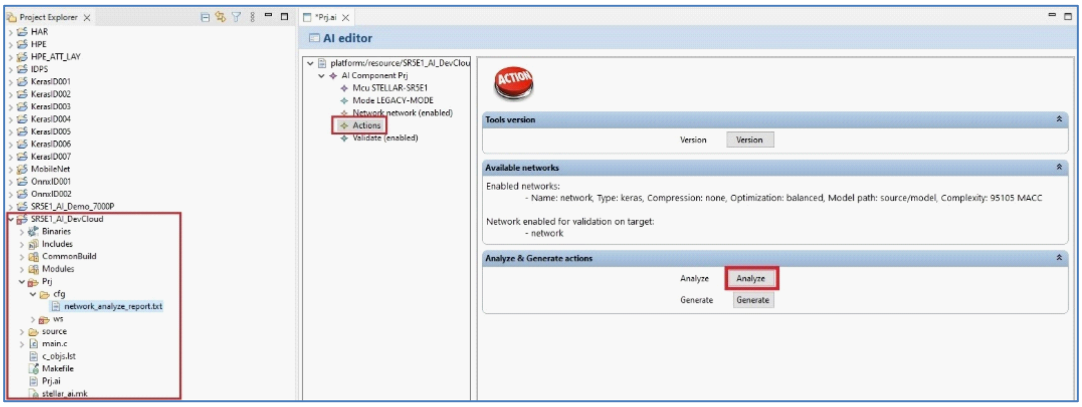


Figure 14. Example of the analyze report

```
STEdgeAI Core v1.0.0
Created date          : 2024-05-28 20:03:23
Parameters            : analyze --target stellar-e -m C:/StellarStudio-5.0/workspace/CIFAR\source/model_nn\cifar10_qkeras_gray_6795_1.h5 --name network --type keras --compression none --workspace C:/StellarStudio-5.0/workspace/CIFAR\CIFAR\ws\ --output C:/StellarStudio-5.0/workspace/CIFAR\CIFAR\cfg\

Exec/report summary (analyze)
------------------------------------------------------------------------------------------------
model file         :   C:\StellarStudio-5.0\workspace\CIFAR\source\model_nn\cifar10_qkeras_gray_6795_1.h5
type               :   keras
c_name             :   network
compression        :   none
optimization       :   balanced
target/series      :   stellar-e
workspace dir      :   C:\StellarStudio-5.0\workspace\CIFAR\CIFAR\ws
output dir         :   C:\StellarStudio-5.0\workspace\CIFAR\CIFAR\cfg
model_fmt          :   dqnn
model_name         :   cifar10_qkeras_gray_6795_1
model_hash         :   0x2af17298990b61af49bcad8201a89c09
params #           :   9,804,126 items (37.40 MiB)
------------------------------------------------------------------------------------------------
input 1/1          :   'input_1', int8(1x32x32x1), 1024 Bytes, QLinear(1.000000000,0,int8), user
output 1/1         :   'activation', f32(1x10), 40 Bytes, user
macc               :   160,339,094
weights (ro)       :   1,237,264 B (1.18 MiB) (1 segment) / -37,979,240(-96.8%) vs float model
activations (rw)   :   16,528 B (16.14 KiB) (1 segment)
ram (total)        :   17,592 B (17.18 KiB) = 16,528 + 1,024 + 40
------------------------------------------------------------------------------------------------

Model name - cifar10_qkeras_gray_6795_1
------ ------------------------------------------------- ----------------------- ---------------------- ------------ ------------------------- --- ------------------- ----------------- ------------------
m_id   layer (type,original)                             oshape                  param/size                     macc              connected to | c_size              c_macc            c_type
------ ------------------------------------------------- ----------------------- ---------------------- ------------ ------------------------- --- ------------------- ----------------- ------------------
0      input_1 (Input, InputLayer)                       [b:1,h:32,w:32,c:1]                                                                   |
------ ------------------------------------------------- ----------------------- ---------------------- ------------ ------------------------- --- ------------------- ----------------- ------------------
1      act_0 (Conversion, QActivation)                   [b:1,h:32,w:32,c:1]                                   2,048                   input_1 |                     -2,048(-100.0%)
------ ------------------------------------------------- ----------------------- ---------------------- ------------ ------------------------- --- ------------------- ----------------- ------------------
2      conv2d_A (Conv2D, QConv2D)                        [b:1,h:32,w:32,c:64]    576/2,304                   589,824                     act_0 | -1,472(-63.9%)                        Conv2D_[0]
------ ------------------------------------------------- ----------------------- ---------------------- ------------ ------------------------- --- ------------------- ----------------- ------------------
3      batch_normalization (ScaleBias, BatchNormalization) [b:1,h:32,w:32,c:64]  128/512                     131,072                  conv2d_A | -512(-100.0%)       -131,072(-100.0%)
------ ------------------------------------------------- ----------------------- ---------------------- ------------ ------------------------- --- ------------------- ----------------- ------------------
4      act_1 (Conversion, QActivation)                   [b:1,h:32,w:32,c:64]                                131,072       batch_normalization |                     -131,072(-100.0%)
------ ------------------------------------------------- ----------------------- ---------------------- ------------ ------------------------- --- ------------------- ----------------- ------------------
5      conv2d_B (Conv2D, QConv2D)                        [b:1,h:32,w:32,c:64]    36,864/147,456           37,748,736                     act_1 | -142,592(-96.7%)                      Conv2D_[1]
------ ------------------------------------------------- ----------------------- ---------------------- ------------ ------------------------- --- ------------------- ----------------- ------------------
6      batch_normalization_1 (ScaleBias, BatchNormalization) [b:1,h:32,w:32,c:64] 128/512                    131,072                  conv2d_B | -512(-100.0%)       -131,072(-100.0%)
------ ------------------------------------------------- ----------------------- ---------------------- ------------ ------------------------- --- ------------------- ----------------- ------------------
7      act_2 (Conversion, QActivation)                   [b:1,h:32,w:32,c:64]                                131,072     batch_normalization_1 |                     -131,072(-100.0%)
------ ------------------------------------------------- ----------------------- ---------------------- ------------ ------------------------- --- ------------------- ----------------- ------------------
8      max_pooling2d (Pool, MaxPooling2D)                [b:1,h:16,w:16,c:64]                                 65,536                     act_2 |                                       Pool_[2]
------ ------------------------------------------------- ----------------------- ---------------------- ------------ ------------------------- --- ------------------- ----------------- ------------------
9      conv2d_C (Conv2D, QConv2D)                        [b:1,h:16,w:16,c:128]   73,728/294,912           18,874,368             max_pooling2d | -285,184(-96.7%)                      Conv2D_[3]
------ ------------------------------------------------- ----------------------- ---------------------- ------------ ------------------------- --- ------------------- ----------------- ------------------
10     batch_normalization_2 (ScaleBias, BatchNormalization) [b:1,h:16,w:16,c:128] 256/1,024                  65,536                  conv2d_C | -1,024(-100.0%)     -65,536(-100.0%)
------ ------------------------------------------------- ----------------------- ---------------------- ------------ ------------------------- --- ------------------- ----------------- ------------------
11     act_3 (Conversion, QActivation)                   [b:1,h:16,w:16,c:128]                                65,536     batch_normalization_2 |                     -65,536(-100.0%)
------ ------------------------------------------------- ----------------------- ---------------------- ------------ ------------------------- --- ------------------- ----------------- ------------------
12     conv2d_D (Conv2D, QConv2D)                        [b:1,h:16,w:16,c:128]   147,456/589,824          37,748,736                     act_3 | -570,880(-96.8%)                      Conv2D_[4]
------ ------------------------------------------------- ----------------------- ---------------------- ------------ ------------------------- --- ------------------- ----------------- ------------------
13     batch_normalization_3 (ScaleBias, BatchNormalization) [b:1,h:16,w:16,c:128] 256/1,024                  65,536                  conv2d_D | -1,024(-100.0%)     -65,536(-100.0%)
------ ------------------------------------------------- ----------------------- ---------------------- ------------ ------------------------- --- ------------------- ----------------- ------------------
14     act_4 (Conversion, QActivation)                   [b:1,h:16,w:16,c:128]                                65,536     batch_normalization_3 |                     -65,536(-100.0%)
------ ------------------------------------------------- ----------------------- ---------------------- ------------ ------------------------- --- ------------------- ----------------- ------------------
15     max_pooling2d_1 (Pool, MaxPooling2D)              [b:1,h:8,w:8,c:128]                                  32,768                     act_4 |                                       Pool_[5]
------ ------------------------------------------------- ----------------------- ---------------------- ------------ ------------------------- --- ------------------- ----------------- ------------------
16     conv2d_E (Conv2D, QConv2D)                        [b:1,h:8,w:8,c:256]     294,912/1,179,648        18,874,368           max_pooling2d_1 | -1,141,760(-96.8%)                    Conv2D_[6]
------ ------------------------------------------------- ----------------------- ---------------------- ------------ ------------------------- --- ------------------- ----------------- ------------------
17     batch_normalization_4 (ScaleBias, BatchNormalization) [b:1,h:8,w:8,c:256] 512/2,048                    32,768                  conv2d_E | -2,048(-100.0%)     -32,768(-100.0%)
------ ------------------------------------------------- ----------------------- ---------------------- ------------ ------------------------- --- ------------------- ----------------- ------------------
18     act_5 (Conversion, QActivation)                   [b:1,h:8,w:8,c:256]                                  32,768     batch_normalization_4 |                     -32,768(-100.0%)
------ ------------------------------------------------- ----------------------- ---------------------- ------------ ------------------------- --- ------------------- ----------------- ------------------
19     conv2d_F (Conv2D, QConv2D)                        [b:1,h:8,w:8,c:256]     589,824/2,359,296        37,748,736                     act_5 | -2,284,544(-96.8%)                    Conv2D_[7]
------ ------------------------------------------------- ----------------------- ---------------------- ------------ ------------------------- --- ------------------- ----------------- ------------------
20     batch_normalization_5 (ScaleBias, BatchNormalization) [b:1,h:8,w:8,c:256] 512/2,048                    32,768                  conv2d_F | -2,048(-100.0%)     -32,768(-100.0%)
------ ------------------------------------------------- ----------------------- ---------------------- ------------ ------------------------- --- ------------------- ----------------- ------------------
21     act_6 (Conversion, QActivation)                   [b:1,h:8,w:8,c:256]                                  32,768     batch_normalization_5 |                     -32,768(-100.0%)
------ ------------------------------------------------- ----------------------- ---------------------- ------------ ------------------------- --- ------------------- ----------------- ------------------
```

6.3 Generate

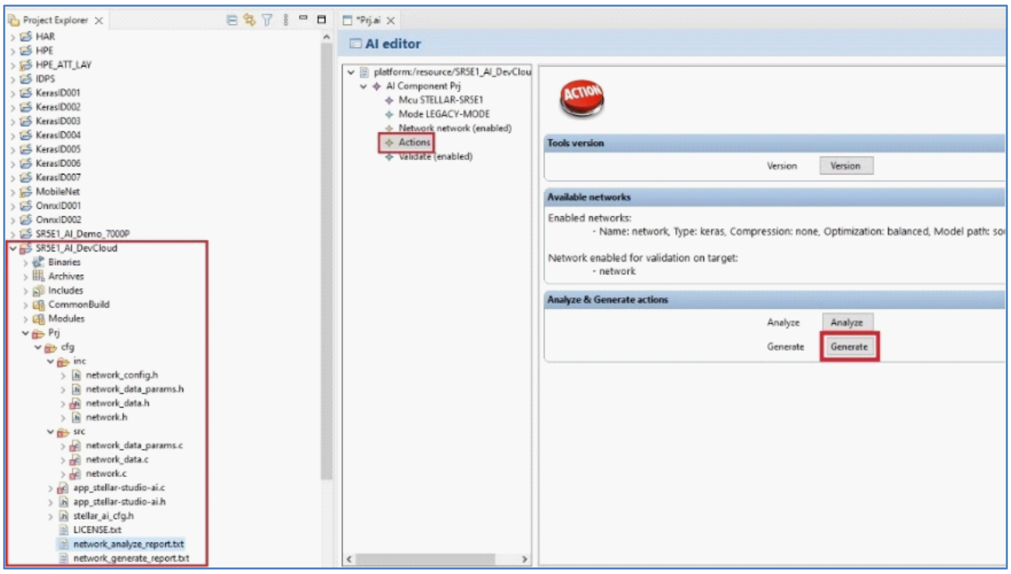
The Generate command is used to generate the C-library files for all the enabled networks within the network panel. Then users can design and develop specific applications based on the APIs of these C-libraries. For each of the enabled networks, the generate command creates the specific neural network files starting with the name of the neural network chosen within the project folder <prj>/<ai_component_name>/cfg/ and generates a report that is shown in the AI console during the command execution and it is also stored in a .txt file within the same folder. The name of the report will be <network_name>_generate_report.txt.

For a full description of the CLI Generate command, you can refer to the ST Edge AI Core technology documentation.

If an error occurs during the generation process of one of the enabled networks, the generate command continues to process the other enabled networks.

To run the generate command, select the actions panel and click on the generate button (see the figure below). A list of all available networks enabled and ready for the generation is also shown.

Figure 15. Generate process



The Figure 16. Example of the generate report is an example of the generated generate report.

Figure 16. **Example of the generate report**

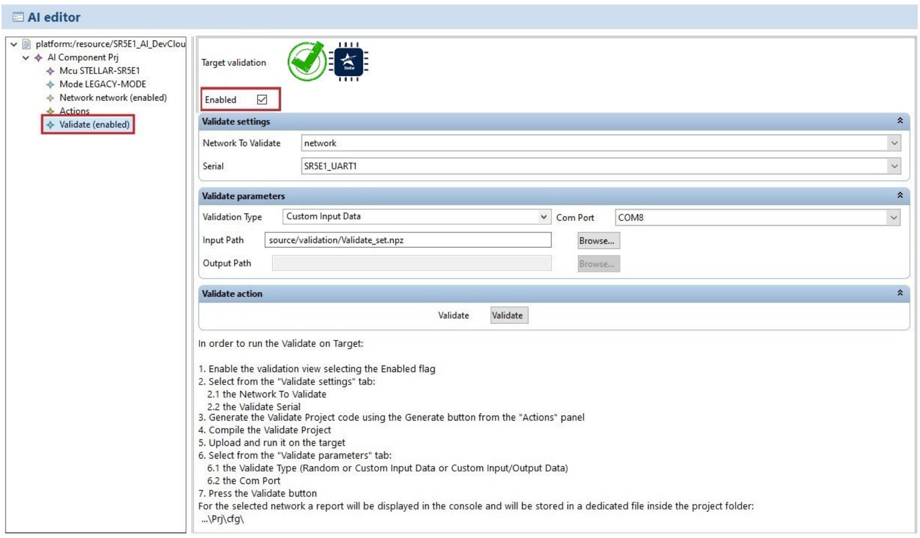
```
STEdgeAI Core v9.0.0-19801
Created date          : 2024-03-19 09:28:13
Parameters            : generate --target stellar-e --c-api legacy -m C:/StellarStudio-5.0/workspace/SR5E1_AI_DevCloud\source/model\CNN_LSTM.h5 --name network --type keras
                        --compression none --workspace C:/StellarStudio-5.0/workspace/SR5E1_AI_DevCloud\Prj\ws\ --output C:/StellarStudio-5.0/workspace/SR5E1_AI_DevCloud\Prj\cfg\

Exec/report summary (generate)
----------------------------------------------------------------------------------------------------
 model file         :   C:\StellarE\Projects\AITests\TestID01\source\model_lstm\CNN_LSTM.h5
 type               :   keras
 c_name             :   lstm
 compression        :   none
 options            :   allocate-inputs, allocate-outputs
 optimization       :   balanced
 target/series      :   generic
 workspace dir      :   C:\StellarE\Projects\AITests\TestID01\TestID01\ws
 output dir         :   C:\StellarE\Projects\AITests\TestID01\TestID01\cfg
 model_fmt          :   float
 model_name         :   CNN_LSTM
 model_hash         :   b84fd65fa67e2b886f5631c67837406e
 params #           :   8,963 items (35.01 KiB)
 ------------------------------------------------------------------------------------------------
 input 1/1          :   'input_0' (domain:activations/**default**)
                    :   80 items, 320 B, ai_float, float, (1,20,1,4)
 output 1/1         :   'dense_1_dense' (domain:activations/**default**)
                    :   1 items, 4 B, ai_float, float, (1,1,1,1)
 macc               :   95,105
 weights (ro)       :   35,980 B (35.14 KiB) (1 segment) / +128(+0.4%) vs float model
 activations (rw)   :   2,304 B (2.25 KiB) (1 segment) *
 ram (total)        :   2,304 B (2.25 KiB) = 2,304 + 0 + 0
 ------------------------------------------------------------------------------------------------
 (*) 'input'/'output' buffers can be used from the activations buffer

PASS:  79%|#######9  | 84/106 [00:00<00:00, 22.81it/s]
PASS:  97%|#########7| 103/106 [00:00<00:00, 27.55it/s]

 Generated files (7)
 ----------------------------------------------------------------------------------
 C:\StellarE\Projects\AITests\TestID01\TestID01\cfg\lstm_config.h
 C:\StellarE\Projects\AITests\TestID01\TestID01\cfg\lstm.h
 C:\StellarE\Projects\AITests\TestID01\TestID01\cfg\lstm.c
 C:\StellarE\Projects\AITests\TestID01\TestID01\cfg\lstm_data_params.h
 C:\StellarE\Projects\AITests\TestID01\TestID01\cfg\lstm_data_params.c
 C:\StellarE\Projects\AITests\TestID01\TestID01\cfg\lstm_data.h
 C:\StellarE\Projects\AITests\TestID01\TestID01\cfg\lstm_data.c

PASS:  97%|#########7| 103/106 [00:00<00:00, 27.55it/s]

Creating txt report file C:\StellarE\Projects\AITests\TestID01\TestID01\cfg\lstm_generate_report.txt

PASS:  97%|#########7| 103/106 [00:00<00:00, 27.55it/s]
PASS:  97%|#########7| 103/106 [00:20<00:00, 27.55it/s]
PASS: 100%|##########| 106/106 [02:17<00:00, 13.66s/it]
```

6.4 Validate

The Validate command allows to import, render, and validate the C-libraries related to the enabled networks in the network panel. To execute the validate command, the validate procedure must be enabled by selecting the validate panel and setting the enabled flag (see Figure 17. Validate enable flag).

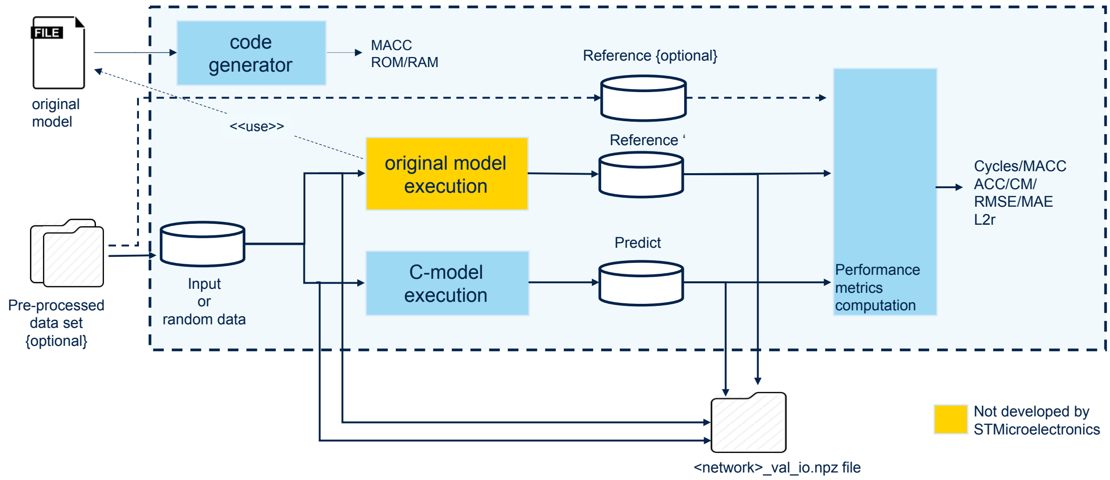
For a full description of the CLI Validate command, you can refer to the ST Edge AI Core technology documentation.

Figure 17. **Validate enable flag**



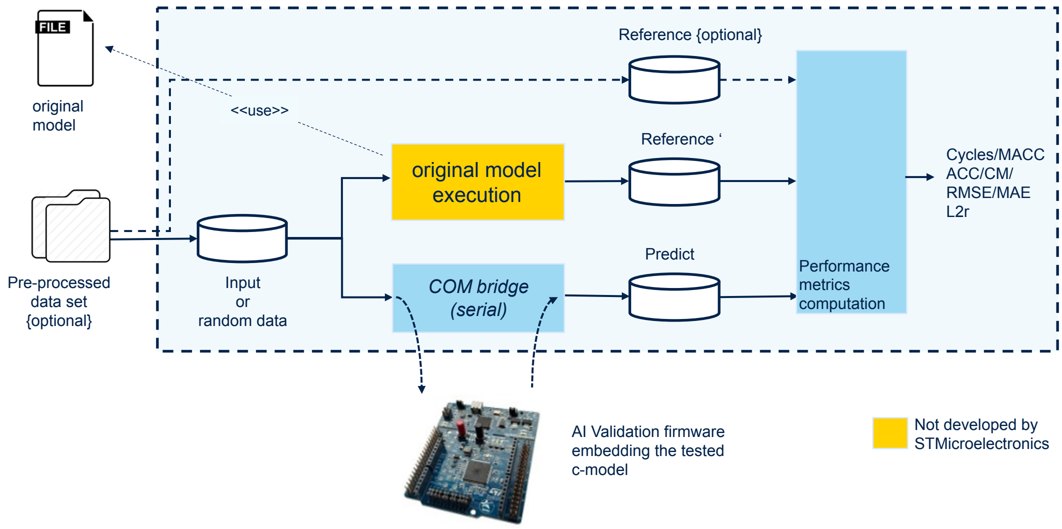
A simple and quick validation mechanism is provided to compare the accuracy of a generated model and the uploaded DL model from a numerical standpoint. Both models are fed with the same input tensors (fixed random inputs or custom dataset). To be more accurate, additional metrics are reported to evaluate the generated C model.

Figure 18. **Validate flow overview**



Only the validation on target is provided to compare 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.

Figure 19. **Validate on target**



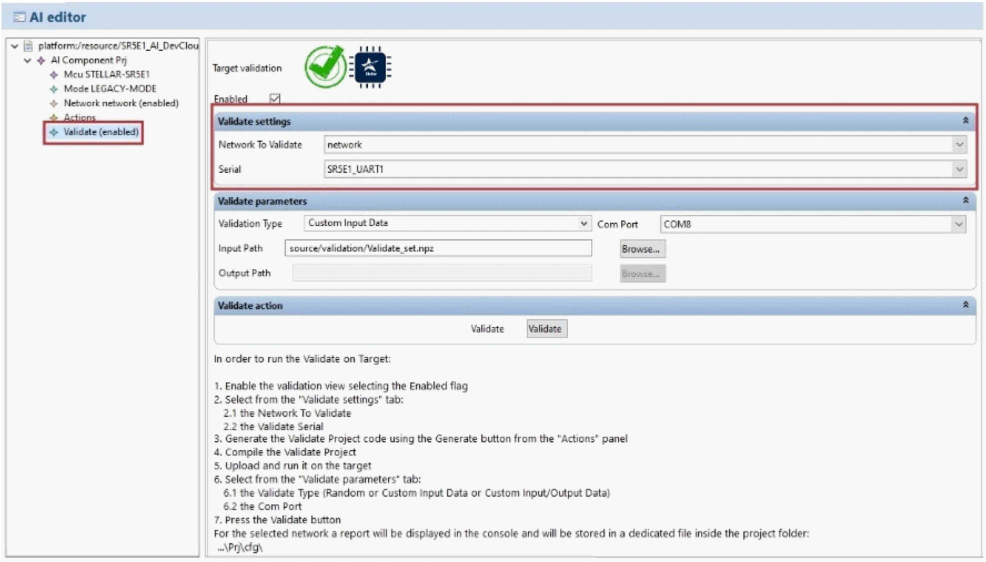
The Validate procedure is based on the communication via serial port between the host (that sends the validation data to the target) and the target (that processes the received data).

For this reason, within the validate panel of the StellarStudio.AI component a validate serial port must be selected. It is possible to select as validate serial one of the UART available for the Stellar device selected in the MCU Target selection panel. If no validate serial port is selected an error is returned during the compilation phase.

The network to validate must also be selected from a list that contains the names of all enabled networks. Only one network at a time can be validated. It must be chosen before running the generating process. If no network is selected (for example, NONE) as the network to validate, an error is returned during the compilation of the validate project.

The Figure 20. Validate settings shows the section validate settings of the validate panel of the StellarStudio.AI component in which it is possible to select the network to validate and the validate serial.

Figure 20. Validate settings



To validate the C-libraries, the following steps are required:

- The C-libraries of all enabled networks must be generated and included in a validate project.
  - Some configuration files, depending on the selected network to validate, are generated too. Because of that, it is needed to select the right network to validate (if more than one is present and enabled) in the validate settings before the generation command execution.
- The validate project must be compiled, downloaded on the target, and executed.
  - A makefile is also automatically generated to build the necessary files.
- The validate procedure must be run from the StellarStudio.AI component pushing the validate button.

The first step is the same as the generate command with the validation enabled flag set in the validate panel and the StellarStudio.AI component generation is run (button generate in Figure 15. Generate process).

Note: *For each new network to validate (if more than one has been loaded and enabled in the network panel), a new generation phase is needed, after having selected the right network to validate in the validate settings. If NONE is selected, a compilation error is generated when building the validate project.*

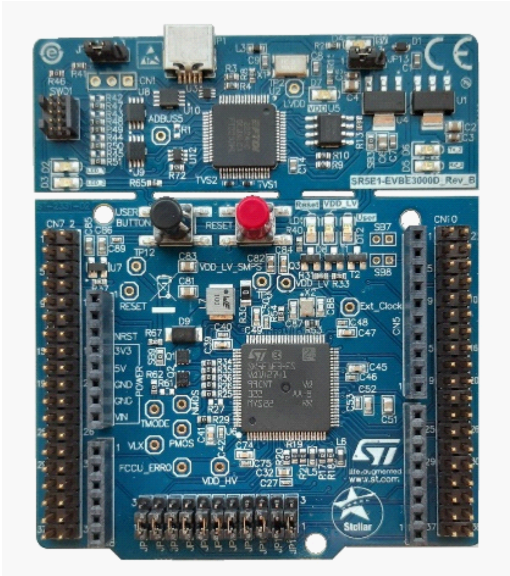
When the generation of the neural network C-libraries is completed, the next step is to create and build (using one of the SDK C-compilers supported) a Stellar SDK validate project including the neural network C-libraries generated. Inside the main application, after the platform setup, it is enough to invoke the API AIValidateStart(). The code below shows a typical main source code of a Stellar SR5E1 line SDK validate project to start the validation of a neural network C-model.

```
#include <test_env.h>
#include <uart.h>
#include <io.h>
#include <irq.h>
#include <stdio.h>

#include "stellar_ai_cfg.h"

#if (STELLAR_AI_VALIDATE == TRUE)
#include "stellar_ai.h"
#endif /* #if (STELLAR_AI_VALIDATE == TRUE) */

/*
 * Example of system initialization function for SR5E1-EVB3000D board
 */
void SystemInit(void)
{
  /* Enable interrupts.*/
  osal_sys_unlock();

  test_env_init((TestInit_t)
```

```
                (TEST_INIT_CLOCK    |
                 TEST_INIT_GPIO     |
                 TEST_INIT_BOARD    |
                 TEST_INIT_IRQ      |
                 TEST_INIT_OSAL));

  gpio_set_pin_mode(UART1_RX, UART1_RX_CFG);
  gpio_set_pin_mode(UART1_TX, UART1_TX_CFG);

  /* Initialize UART driver instance used for IO redirection.*/
  uart_init(&DRV_UART1);

  /* Configure UART driver instance used for IO redirection.*/
  (void)uart_set_prio(&DRV_UART1, IRQ_PRIORITY_5);
  (void)uart_set_rx_drv_mode(&DRV_UART1, UART_RX_DRV_MODE_INT_SYNC);
  (void)uart_set_tx_drv_mode(&DRV_UART1, UART_TX_DRV_MODE_INT_SYNC);
  (void)uart_set_baud(&DRV_UART1, UART_BAUDRATE_115200);
  (void)uart_set_presc(&DRV_UART1, UART_PRESCALER_DIV1);
  (void)uart_set_parity(&DRV_UART1, UART_PARITY_NONE);
  (void)uart_set_over(&DRV_UART1, UART_OVERSAMPLING_16);
  (void)uart_set_sbit(&DRV_UART1, UART_STOPBIT_1);

  /* Initialize Runtime IO module.*/
  io_init(&DRV_UART1);

  /* Start Runtime IO module.*/
  io_start();

  /* Enabling the Data Cache when validation is disabled.*/
  SCB_EnableDCache();
}

#if (STELLAR_AI_VALIDATE == TRUE)
/*
 * Application entry point for validation process
 */
\ main(void) {

  /* System initialization.*/
  SystemInit();

  /* Run the validate procedure.*/
  aiValidateStart();

  /* never here...*/
  while (true) {
  }
}
#else
/*
 * Application entry point for inference run
 */
int main(void) {

  /* System initialization.*/
  SystemInit();

  printf("#### AI application to run inference.\r\n");

  /* Run user application based on the AI neural network. */
  ai_application();

  /* never here...*/
  while (true) {
  }
}
#endif /* #if (STELLAR_AI_VALIDATE == TRUE) */
```

After compiling, the validate project must be flashed on a target Stellar board using one of the available tools (via USB Stellar link or JTAG interface). The procedure of flashing depends on the target board used. For example, using a discovery SR5E1-EVB3000D revision B board it can be done simply by using the OpenOCD software under Stellar studio environment and the StellarLINK hardware interface of the board.

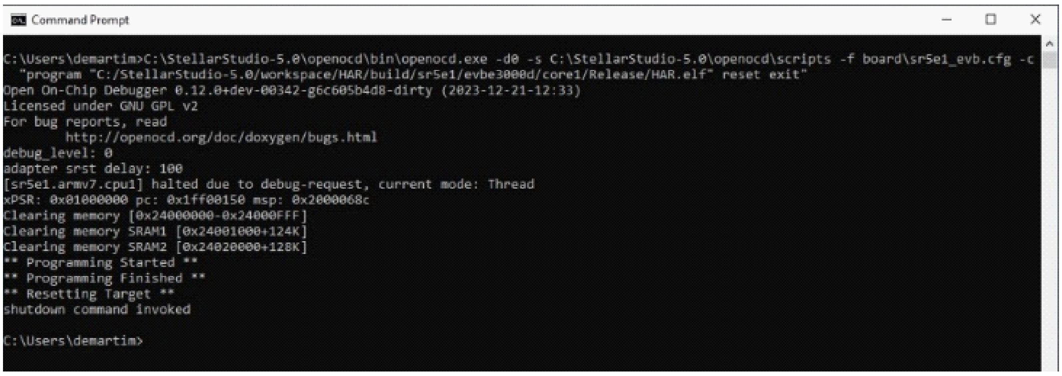
Figure 21. SR5E1-EVB3000D revision B board



After power-on the SR5E1-EVB3000D revision B board with a USB cable (attached to the PC), the command to flash and download the validation binary code with the OpenOCD software is:

C:\StellarStudio-5.0\openocd\bin\openocd.exe -d0 -s C:\StellarStudio-5.0\openocd\scripts -f board\sr5e1_evb.cfg -c "program "C:/StellarStudio-5.0/workspace/HAR/build/sr5e1/evbe3000d/core1/Release/HAR.elf" reset exit" and can be simply executed under a command prompt:

Figure 22. How to flash the code on a target board using OpenOCD



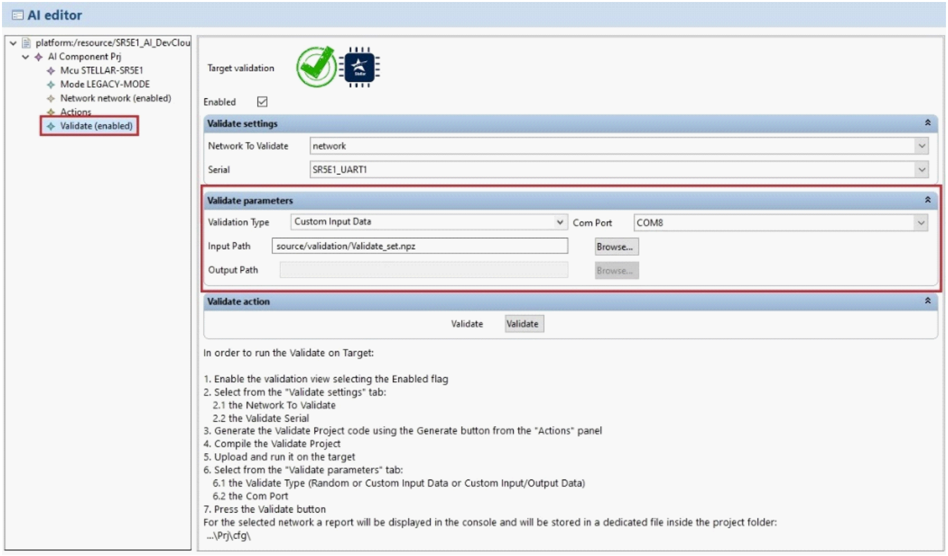
The messages ** Programming started ** and ** Programming finished ** show the result of the command.

For more details about how to install StellarLINK drivers and how to use OpenOCD software you can refer to the Stellar Studio documentation.

After the flash programming is completed successfully, the code is automatically executed on the target board, and it waits in a loop for the incoming data to validate the model under evaluation.

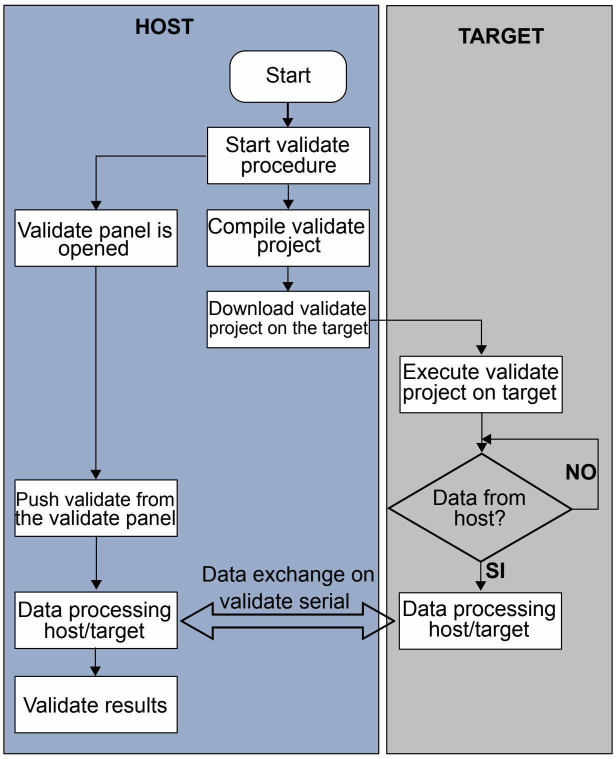
The next step is to select the validate parameters in the section validate parameters of the validate panel (see Figure 23. Validate parameters):

Figure 23. **Validate parameters**



The figure below shows the flowchart of the validate procedure.

Figure 24. **Validate flowchart**



The validate parameters to set are:

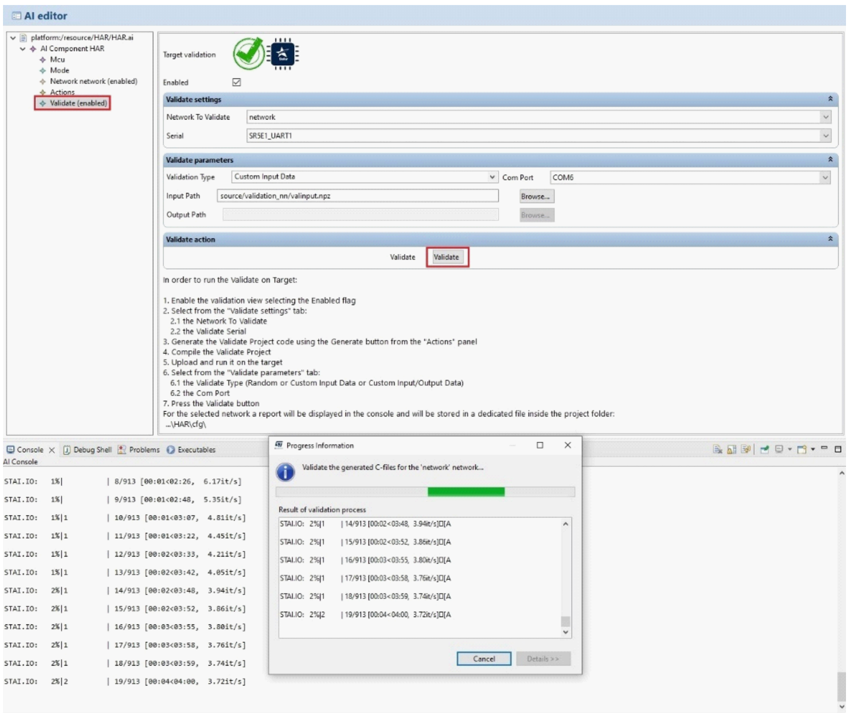
- **Com port**: it is the HOST COM on which the target is connected on. It can be selected manually between one of the COM ports available in the list or if "Auto" (default choice) is selected an automatic detection will start looking for the available COM ports in the system.

- **Validate type**: it is the type of custom test data used by the validate procedure. The user can select among:
  - **Random**: an internal self-generated random dataset is used (default value).
  - **Custom input data**: a custom dataset is used. In this case the user has to provide a single file containing the dataset. The supported file extensions are:
    - **.npz**: in this case the file can contain both inputs and expected outputs or the only inputs. If the only inputs are provided, the expected outputs are automatically obtained by the model files using the l2r metric.
    - **.npy** *or* ***.csv:*** in this case the file contains only the inputs. The expected outputs are automatically obtained by the model files using the l2r metric.
  - **Custom input/Output data:** a custom dataset is used. In this case the user has to provide both the custom input data and expected custom output data. The supported file extensions are.**npz**, .**npy** or **.cvs** for both custom input and expected output data. Note that if an **.npz** file containing both input and expected output is provided as custom input data, the custom output data file is ignored.
- **Input path**: it is the custom input data file (.npz, .npy or .csv).
- **Output path**: it is the custom output data file (.npz, .npy or .csv).

For a full description of these CLI options (-vi/valinput and -vo/valoutput), you can refer to the ST Edge AI Core technology documentation.

When all the validate parameters are correctly selected, the validate procedure can be started by clicking the validate button in the validate panel. See the figure below.

Figure 25. Validate process



The validate procedure will generate for the selected network validated a report that is shown in the Stellar Studio console during the command execution and is also stored in a .txt file within the project folder <prj>/<ai_component_name>/cfg/. The name of the validation report generated will be <network_name>_validate_report.txt.

Please, before starting the validate procedure, verify that the COM related to the UART selected as validate serial is not busy on another task, otherwise the communication between the host and the target fails and the validation procedure returns an error. The communication is done @ 115200 bps.

When a validate procedure is completed, a new one with different parameters can be started from the validate panel. But if the user wants to add new networks to the validate project or wants to enable/disable some networks already defined in the network panel, it is recommended to restart the validate procedure doing a manual code generation file cleaning and then a new code generation.

The validate procedure can also be stopped during its execution. Note, if the validate procedure is stopped, it could be necessary to disconnect and then reconnect the target to the host before starting a new validate procedure.

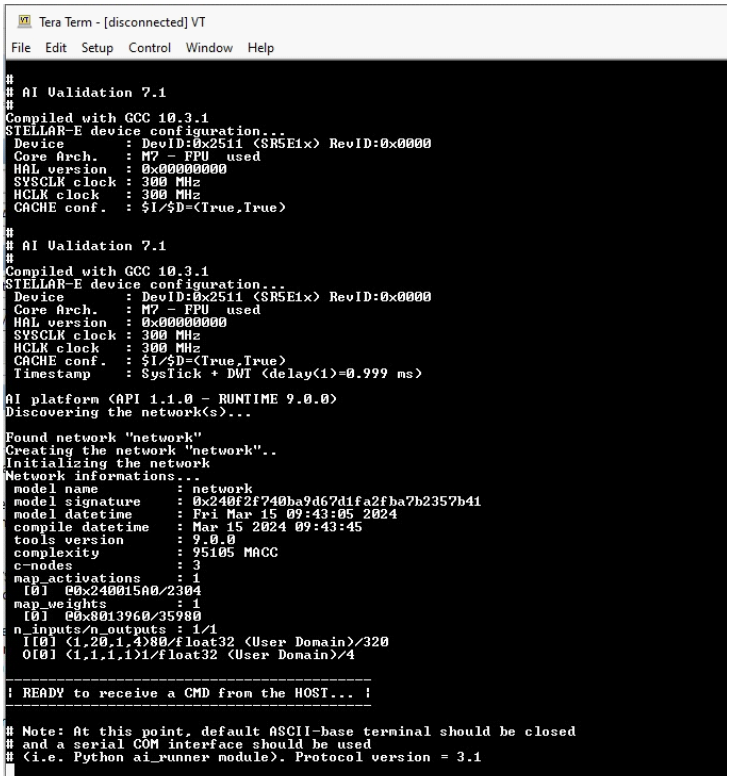
When a validate procedure has been completed, a new one with different parameters can be started from the validate panel.

But if the user wants to add new networks or wants to enable/disable some networks already defined, it is recommended to restart the validate procedure by doing a manual code generation file cleaning and then a new code generation. The validate procedure can also be stopped during its execution. If the validate procedure is stopped, it could be necessary to disconnect and then reconnect the target to the host before starting a new procedure. If in the validate project, the RuntimeIO driver has been also added, the information about the networks added is printed on the serial port. To see them, just open a terminal emulator on that serial port.

Note, if the user configures the same serial in both RuntimeIO driver and in the StellarStudio.AI component, it will be mandatory to disconnect the terminal emulator before starting the validate procedure to avoid a communication failure.

The Figure 26. Neural network runtime information shows the typical information of the networks included in the Stellar Studio SDK validation project printed on the serial console when the RuntimeIO driver is included in the validate project.

Figure 26. Neural network runtime information



7 Supported compilers

The StellarStudio.AI component is fully working with the following Arm compilers:

- GNU Arm embedded toolchain 10.3-2021.10
- HighTec clang version 8.1.0
- IAR ANSI C/C++ compiler V9.30.1.335/W64
- ARMCLANG Arm compiler for embedded 6.21

The Stellar Studio ecosystem with both StellarStudio.AI component and SDKs installed already contain some AI demo applications for some specific boards, to be used as reference.

Revision history

Table 2. Document revision history

Date	Version	Changes
04-Sep-2023	1	Initial release.
14-Jun-2024	2	Document status changed from ST Restricted to public. Updated *Section Introduction, Section 1: Requirements and Section 2: Overview*. Added *Section 3: MCU parameters and Section 4: Mode parameters.*. Updated *Section 5: Network parameters and Section 6: AI commands.*. Added *Section 6.1: Version*. Updated *Section 6.2: Analyze, Section 6.3: Generate and Section 6.4: Validate* Removed *"Embedded inference client API"* and all subsection. Updated *Section 7: Supported compilers*. Removed *"Supported deep learning toolboxes and layers"* and all subsection. Removed *"How to run a c-model locally"*. Removed *"Key metrics"* and all subsection.
18-Nov-2024	3	Updated *Section 1: Requirements, Section 2: Overview, Section 3: MCU parameters, Section 5: Network parameters, Section 6.4: Validate and Section 7: Supported compilers.*
27-May-2025	4	Updated Section 3: MCU parameters. Updated Table 1. Neural network model file extensions.

Contents

List of tables

List of figures