life.augmented

UM2526

ユーザ・マニュアル

人工知能（AI）用 X-CUBE-AI 拡張パッケージ入門

概要

このユーザ・マニュアルでは、学習済みの ニューラル・ネットワーク （NN）の自動変換機能および生成された最適化ライブラリの実装機能が搭載された STM32 マイクロコントローラ用の完全な 人工知能（AI）の IDE ベースのプロジェクトを段階的に構築するためのガイドラインを提供します。STM32CubeMX ツールと完全に統合された X-Cube-AI 拡張パッケージ について説明します。このユーザ・マニュアルでは、AI システムの性能および検証用のオプションのアドオン AI テスト・アプリケーションまたはユーティリティについても説明しています。

本書では、主に STM32 AI ベースのプロジェクトを迅速に生成する体験学習形式で説明しています。パブリック・ドメインから NUCLEO-F746ZG 用の開発キットおよび 深層学習（DL）用のいくつかのモデルが、実用例として使用されます。STM32F0、STM32F3、STM32F4 、STM32G0 、STM32G4 、STM32L0 、STM32L4 、STM32L4+ 、STM32L5 、STM32F7 、STM32H7 、STM32U5 、STM32WB、または STM32WL シリーズのマイクロコントローラ・ベースの STM32 開発キットまたはカスタマ・ボードは、いずれも微調整して使用することができます。

本書の次の部分では、AI の性能および検証用のアドオン・アプリケーションのために X-Cube-AI を使用する方法について詳しく説明します。生成された NN ライブラリなどの内部要素もカバーされます。また、インストールされたパッケージの ドキュメント フォルダから詳細な情報（コマンドライン・サポート、サポートされているツールボックスとレイヤ、レポートされたメトリック）を入手可能です。

STM32 Cube.AI

STM32 Cube

1 一般情報

X-Cube-AI 拡張パッケージ は、STM32 Arm® Cortex®-M ベースのマイクロコントローラで実行される AI プロジェクト専用です。

このユーザ・マニュアルの現在のリビジョンの説明は以下に基づきます。

- X-Cube-AI 7.1.0
- 組込み推論クライアント API 1.2
- コマンドライン・インタフェース 1.6

本書の例に使用する学習済みの Keras DL モデルは、次のとおりです。

- https://github.com/Shahnawax/HAR-CNN-Keras：Keras で CNN を使用した 人間行動認識

注 Arm は、米国内およびその他の地域にある Arm Limited 社（またはその子会社）の登録商標です。

1.1 STM32Cube とは

STM32Cube は開発の工数、時間、コストを削減して設計者の生産性を大幅に向上させるために ST マイクロエレクトロニクスが独自に提唱している取り組みです。STM32Cube は、STM32 ポートフォリオ全体に対応しています。

STM32Cube には、次の内容が含まれます。

- コンセプトから実現までのプロジェクト開発をカバーする扱いやすいソフトウェア開発ツール群：
  - STM32CubeMX：グラフィックウィザードにより初期化 C コードを自動的に生成する、グラフィックインタフェースを備えたソフトウェア設定ツール
  - STM32CubeIDE：ペリフェラル設定、コード生成、コード・コンパイル、デバッグの機能を備えたオールインワンの開発ツール
  - STM32CubeProgrammer（STM32CubeProg）：グラフィック・バージョンとコマンドライン・バージョンで使用できるプログラミング・ツール
  - STM32CubeMonitor（STM32CubeMonitor、STM32CubeMonPwr、STM32CubeMonRF、STM32CubeMonUCPD）：STM32 アプリケーションの動作と性能をリアルタイムで微調整する強力な監視ツール
- STM32Cube MCU パッケージおよび MPU パッケージ：マイクロコントローラおよびマイクロプロセッサの各シリーズに特化した包括的な組込みソフトウェア・プラットフォーム（STM32F7 シリーズの STM32CubeF7 など）：
  - STM32Cube ハードウェア抽象化レイヤ（HAL）：STM32 ポートフォリオの製品間で最大限の移植性を実現
  - STM32Cube 低階層 API：ユーザがハードウェアを高度に制御して最高のパフォーマンスと最小のフットプリントを実現
  - RTOS、USB、TCP/IP、グラフィック、および FAT ファイル・システム などの一貫性のあるミドルウェア・コンポーネント群
  - すべてのペリフェラル群と実用的なサンプルを備えた組込みソフトウェアユーティリティ
- STM32Cube 拡張パッケージ：以下により STM32Cube MCU パッケージおよび MPU パッケージ の機能を補足する組込みソフトウェア・コンポーネントが含まれています。
  - ミドルウェア拡張および実用的なレイヤ
  - 一部の STMicroelectronics 開発ボードで動作するサンプル

1.2 X-Cube-AI が STM32Cube を補完する方法

X-Cube-AI は、計算およびメモリ（RAM および Flash メモリ）で最適化された自動 NN ライブラリ・ジェネレータを提供することで、STM32CubeMX を拡張します。これにより一般的に使用される DL フレームワーク（Keras、TensorFlow™ Lite、ONNX など）の学習済みの ニューラル・ネットワーク がライブラリに変換され、最終ユーザのプロジェクトに自動的に統合されます。プロジェクトが自動的にセットアップされ、STM32 マイクロコントローラでコンパイルや実行ができる状態になります。

また、X-Cube-AI は、ユーザの NN 固有の基準要件（RAM または Flash メモリ のサイズ）に適した正しいデバイスを選択するための固有のマイクロコントローラ・フィルタをプロジェクト作成用に追加することで、STM32CubeMX を拡張します。

X-Cube-AI ツールでは、次の 3 種類のプロジェクトを生成できます。

- STM32 マイクロコントローラで動作するシステム性能プロジェクトにより、NN の推論 CPU 負荷とメモリ使用量を正確に測定することができる
- デスクトップ PC と STM32 Arm® Cortex®-M ベースのマイクロコントローラが組み込まれた環境の両方において、ランダムなテスト·データまたはユーザ·テスト·データによって刺激された、NN によって返された結果を段階的に検証する検証プロジェクト。
- AI ベースのアプリケーションの構築を可能にするアプリケーション·テンプレート·プロジェクト

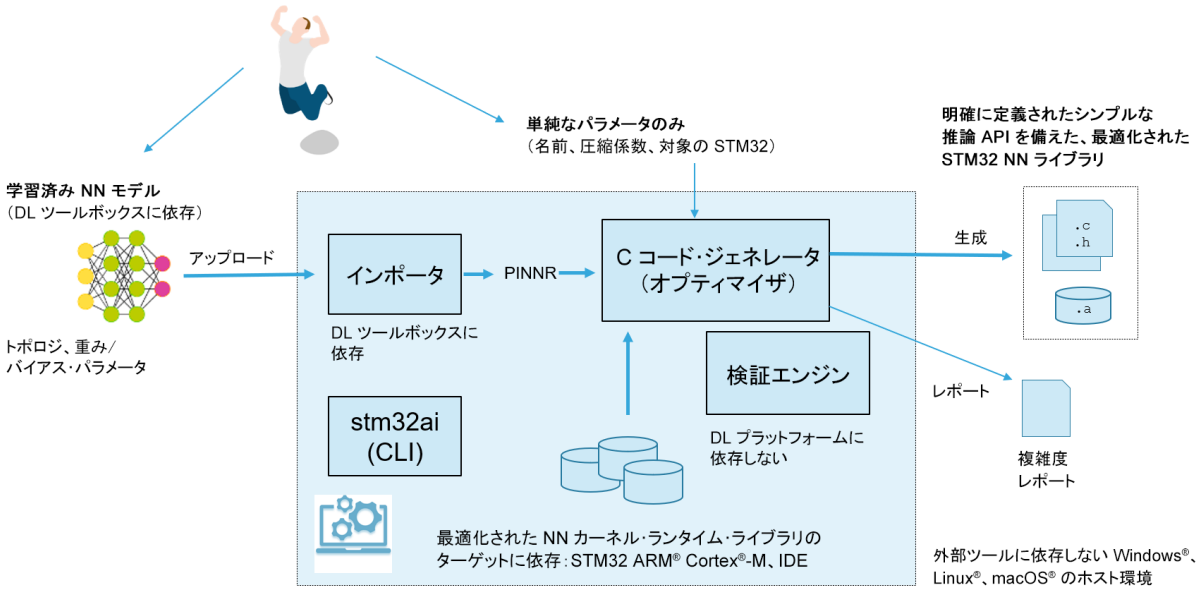
TensorFlow™ Lite モデルを使用する場合、ツールでは STM32Cube.AI ライブラリまたは TensorFlow™ ソース·リポジトリで提供されている マイクロコントローラ用の TensorFlow™ Lite を使用して、コードを生成することができます。

1.3 X-Cube-AI コア·エンジン

図 1 および 図 2 に示す X-Cube-AI コア·エンジンは、セクション 1.4 で後述する X-Cube-AI 拡張パッケージ の一部です。これは、限られた制約のあるハードウェア·リソースを持つ組込みシステム用の自動および高度な NN マッピング·ツールを、学習済みの ニューラル·ネットワーク(DL モデル)の最適化された堅牢な C モデル実装を生成し、展開するために提供します。生成された STM32 NN ライブラリ(特殊部品と汎用部品の両方)は、直接 IDE プロジェクトまたは makefile ベースのビルド·システムに統合できます。明確な特定の推論クライアント API (「セクション 8 組込み推論クライアント API」を参照)も、クライアント AI ベースのアプリケーションを開発するためにエクスポートされます。深層学習 向けの各種フレームワーク(DL ツールボックス)とレイヤがサポートされています(セクション 12 サポートされている 深層学習 のツールボックスおよびレイヤを参照)。

すべての X-Cube-AI コア機能は、完全かつ統合されたコマンド·ライン·インタフェース(コンソール·レベル)から提供され、最適化された STM32 デバイス用の NN C ライブラリを分析、検証、および生成するための主な手順を実行します([6] を参照)。また、Keras モデルの学習後の量子化をサポートします。

図 1. X-CUBE-AI コア·エンジン

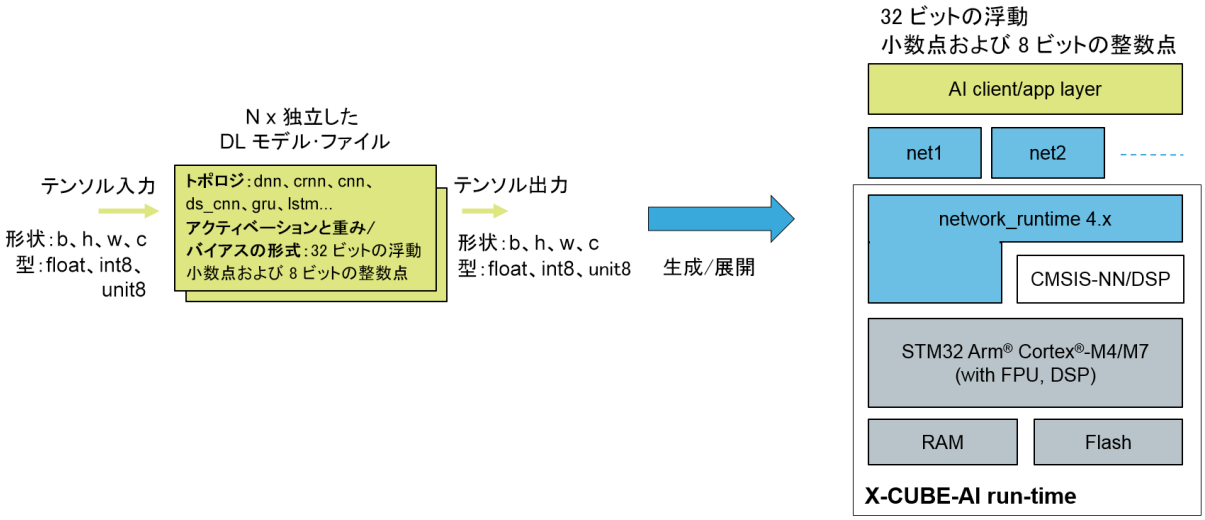


シンプルな設定インタフェースが公開されます。学習済みの DL モデル·ファイルでは、いくつかのパラメータのみが要求されます。

- 名前:生成された C モデルの名前を示します(デフォルト値は network)
- 圧縮:重み/バイアス·パラメータのサイズを低減するための圧縮係数を示します(「セクション 6.1 グラフ·フローとメモリ·レイアウト·オプティマイザ」を参照)。
- STM32 ファミリ:最適化された NN カーネル ランタイム ライブラリを選択します。

図 2 に、アップロードされている DL モデルとターゲット・サブシステム ランタイム でサポートしている主な機能を示します。

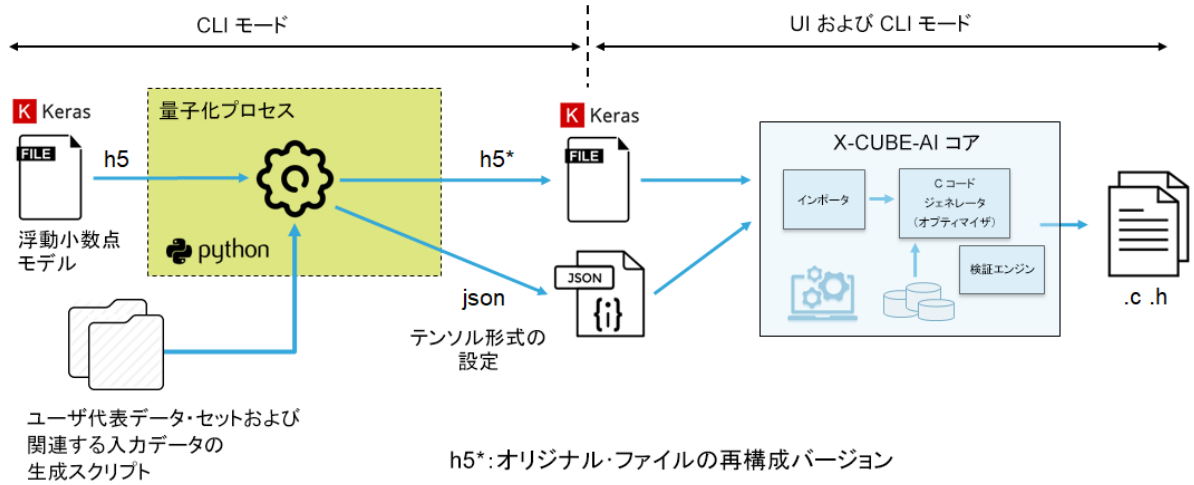
図 2. X-CUBE-AI の概要



- 単純テンソル入力と単純テンソル出力のみがサポートされます。
  - 4 次元形状：バッチ、高さ、幅、チャネル（`channel-last` フォーマット、[10] を参照）
  - 浮動小数点型（32 ビット）および固定小数点型（8 ビット）
- 生成された C モデルは、FPU および DSP 拡張を備えた STM32 Arm® Cortex®-M4/M7/M33 コア用に完全に最適化されています。

X-Cube-AI コード・ジェネレータは、事前に量子化された 8 ビット固定小数点/整数 Keras モデルと量子化された TensorFlow™ Lite モデルを生成して展開するために使用できます。Keras モデルでは、再構成されたモデル・ファイル（`h5*`）および独自のテンソル形式の設定ファイル（`json`）は必須です。

図 3. 量子化フロー



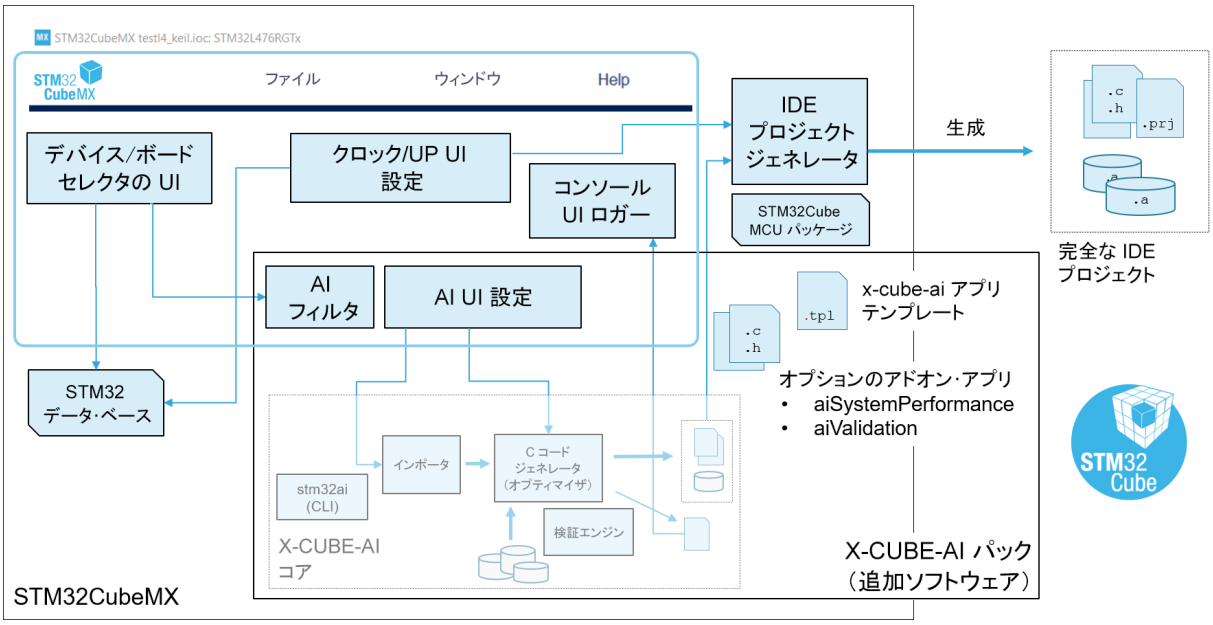
コード・ジェネレータは、重みとバイアス、および関連するアクティベーションを浮動小数点から 8 ビット精度に量子化します。これらは、サポートされているカーネルの最適化された特殊な C の実装に配置されます（[7] を参照）。そうしないと、浮動小数点型の演算子が使用され、浮動小数点から 8 ビットおよび 8 ビットから浮動小数点への変換演算子が自動的に挿入されます。この技術の目的は、モデル・サイズを低減するとともに、モデル精度をほとんど低下させずに CPU およびハードウェア・アクセラレータの遅延（消費電力の側面を含む）を改善することです。

学習済みの浮動小数点 Keras モデルから再構成された Keras モデル·ファイルや関連するテンソル形式の設定ファイルを生成するために、stm32ai アプリケーション(コマンドライン·インタフェース)では学習後の量子化プロセス全体を統合します([12]を参照)。

1.4 STM32CubeMX 拡張

STM32CubeMX は、STM32 マイクロコントローラ向けのソフトウェア設定ツールです。このツールでは、1 回クリックするだけで STM32 の完全な IDE プロジェクトを作成できます。これには、グラフィック·ウィザード(ピン配置競合解決機能、クロック·ツリー設定ヘルパなど)を使用してデバイスやプラットフォーム(ピン、クロック·ツリー、ペリフェラル、ミドルウェア)を設定する C 初期化コードの生成が含まれます。

図 4. STM32CubeMX の X-Cube-AI コア



ユーザの観点から、X-Cube-AI 拡張パッケージ の統合は、ペリフェラルまたはミドルウェア·ソフトウェア·コンポーネントの追加とみなすことができます。X-Cube-AI コアには、次の主な機能が提供されます。

- マイクロコントローラ·フィルタ·セレクタの拡張機能で、十分なメモリを持たないデバイスを削除するための、固有の AI フィルタ·オプション。有効にすると、Arm® Cortex®-M4、-M7、または -M33 コアのない STM32 デバイスは直ちに除外されます。
- 複数の DL モデルをアップロードできる完全な AI UI 設定ウィザード。デスクトップ PC およびターゲット上で生成された C コードの検証プロセスが含まれています。
- 最適化された STM32 NN ライブラリの生成と、選択された STM32 Arm® Cortex®-M コアおよび IDE との統合を支援する、IDE プロジェクト·ジェネレータの拡張機能。
- 生成された NN ライブラリを含む、すぐに使用できる完全な AI テスト·アプリケーション·プロジェクトを生成可能なオプションのアドオン·アプリケーション。ユーザは、ファームウェア·イメージを生成してプログラムするために、任意の IDE 内にこれをインポートしておかなければなりません。エンド·ユーザが追加コードや変更を要求することはありません。
- オンデバイスの AI 検証用ファームウェア(外部メモリのサポートを含む)を自動的に生成、プログラム、実行するワンクリック·サポート。
- ニューラル·ネットワーク ファイルが TensorFlow™ Lite ファイルのときに、STM32Cube.AI ランタイムまたは マイクロコントローラ用の TensorFlow™ Lite を使用した生成。

1.5 略記、略語、および定義

表 1 に、本書で使用される特定の略記と略語について詳しく説明します。

表 1. 本書で使用される用語の定義

AI	人工知能、マシン・インテリジェンスと呼ばれることもあります。通常、AI とは、人間の観点から「スマート」とみなすことができる方法で機械がタスクを実行できることを示す幅広い概念です。これは、知的な存在に関連するタスクを実行するデジタル機器の能力を表します。
DL	深層学習（深層構造学習または階層学習とも呼ばれる）。DL モデルは、生物の神経系における情報処理およびコミュニケーション・パターンから漠然と着想を得ています。
ML	機械学習 は 人工知能（AI）の応用です。明示的にプログラムすることなく、自動的に学習し、経験から改善する機能をシステムに提供します。
MACC	積和演算の計算量は、処理の見地から DL モデルの計算量を示す単位です。
PINNR	プラットフォームに依存しない ニューラル・ネットワーク 表現は、次のステージ（オプティマイザおよび C コード・ジェネレータ）用にアップロードされた DL モデルの共通で移植可能な内部表現を得るために、フロントエンド（X-Cube-AI コア・インポータ）によって生成されたファイルです。

1.6 前提条件

次のパッケージをインストールする必要があります（セクション 2 X-Cube-AI のインストールを参照）。

- STM32CubeMX V5.4.0 以降
- 追加ソフトウェア・パック - STM32CubeMX AI（X-Cube-AI）7.1.0 パック
- STM32CubeProgrammer（STM32CubeProg）バージョン 2.1.0 以降。STM32CubeIDE を使用する場合を除き、ターゲットで自動検証のメリットを得るためには、STM32CubeProgrammer をインストールする必要があります。

STM32 の次の ツールチェーン または IDE のいずれかをインストールする必要があります。

- ST マイクロエレクトロニクス - STM32CubeIDE バージョン 1.0.1 以降
- IAR Systems - IAR Embedded Workbench® IDE - Armv8.x（www.iar.com/iar-embedded-workbench）
- Keil® - MDK-ARM プロフェッショナル版 - µVision® V5.25.2.0（www.keil.com）
- GNU Arm Embedded ツールチェーン（developer.arm.com/open-source/gnu-toolchain/gnu-rm）

X-Cube-AI は、次のオペレーティング・システムに展開できます。

- Windows® 10
- Ubuntu® 18.4
- macOS®（x64）（macOS が明確に関連付けられている場合、macOS® Catalina でテスト）

注 Ubuntu® は Canonical 社の登録商標です。

macOS® および OS X® は、米国内およびその他の国や地域で登録された、Apple 社の商標です。

その他の商標は、それぞれの所有者に帰属します。

1.7 ライセンス

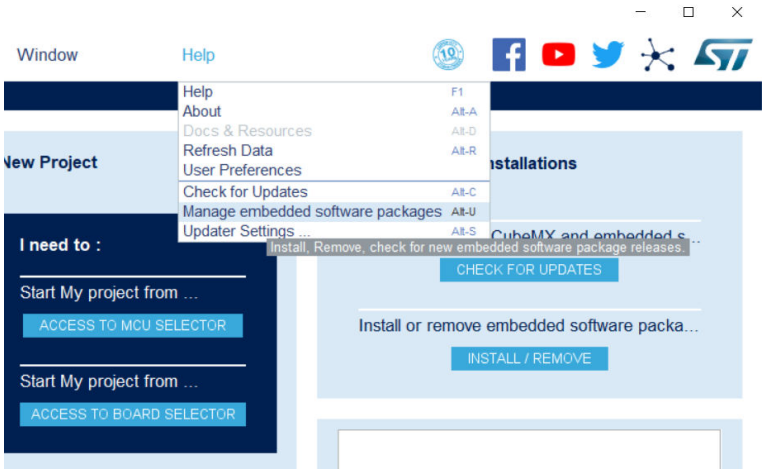
X-Cube-AI は Mix Ultimate Liberty+OSS+3rd パーティ V1 ソフトウェア使用許諾契約（SLA0048）の下で提供されます。

2 X-Cube-AI のインストール

STM32CubeMX（バージョン 5.4.0 以降）をダウンロード、インストール、起動後、いくつかの手順で X-Cube-AI 拡張パッケージ をインストールできます。

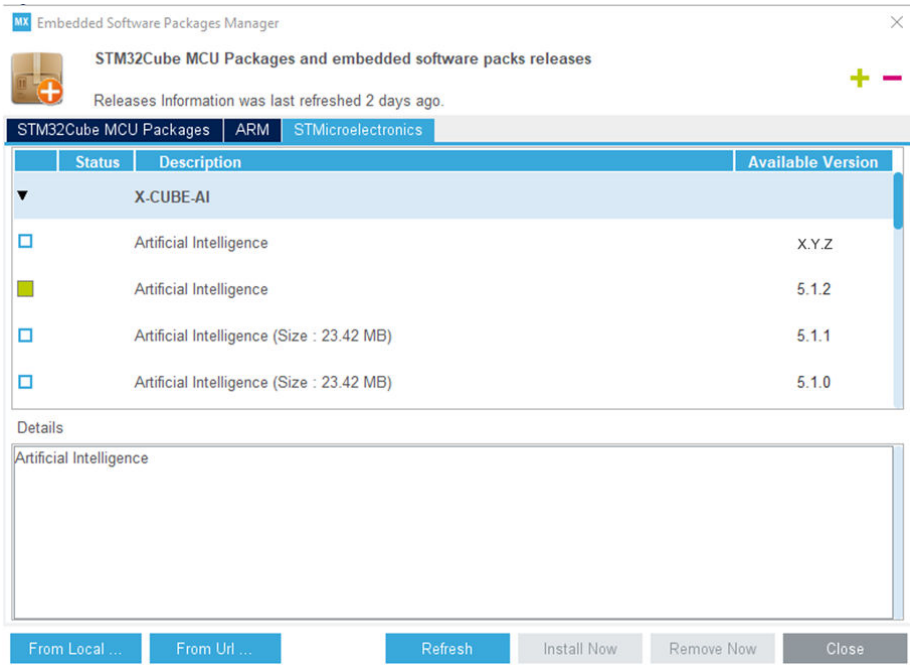
1. メニューから [Help]>[Manage embedded software packages] を選択するか、直接 [INSTALL/ REMOVE] ボタンをクリックします。

図 5. STM32CubeMX での組込みソフトウェア・パックの管理



2. 組込みソフトウェア・パッケージ・マネージャ ウィンドウで、[Refresh] ボタンを押して、アドオン・パックの更新されたリストを取得します。ST マイクロエレクトロニクス タブに移動して X-Cube-AI を検索します。

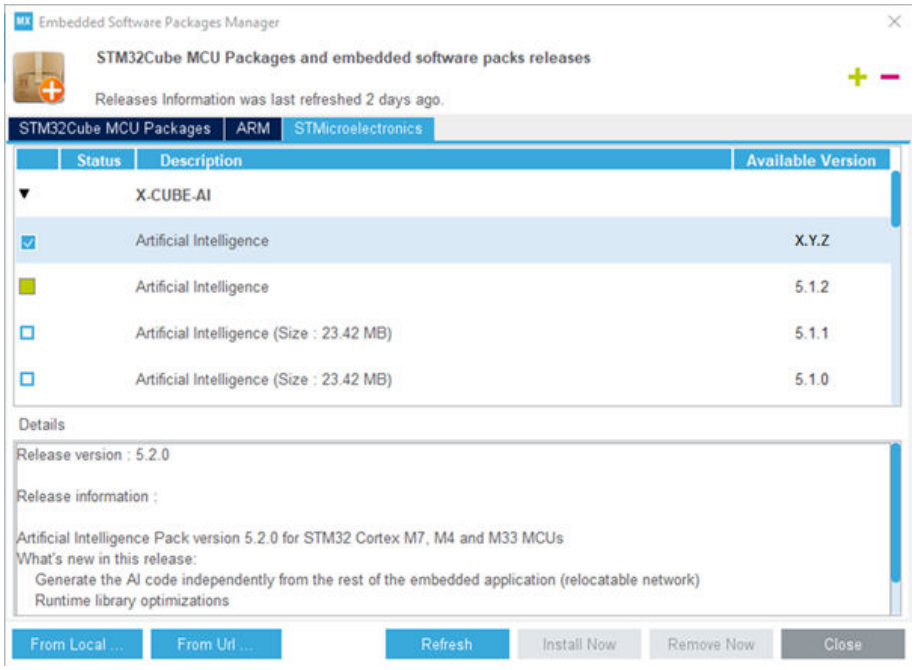
図 6. STM32CubeMX への X-Cube-AI のインストール



注 X.Y.Z は、X-Cube-AI の現在のバージョンを表します。
X-Cube-AI がすでにインストールされている場合は、新規にインストールする前に削除することを推奨します。

3. 対応するボックスをチェックして選択し、[Install Now] ボタンを押してインストールします。インストールが完了すると、対応するボックスが緑色になり、[Close] ボタンが押せるようになります。

図 7. STM32CubeMX の X-Cube-AI



X-Cube-AI 拡張パッケージ には、パックを初めて使用するときにダウンロードされる、OS 固有の部分が含まれています。

ダウンロードをトリガするには、次のいずれかを行います。

- マイクロコントローラ·セレクタの AI フィルタを選択します。
- STM32CubeMX プロジェクト X-Cube-AI の拡張機能を追加します。

または、https://sw-center.st.com/packs/x-cube-ai/stm32ai-<OS>-<version>.zip から OS 固有の部分を直接ダウンロードすることもできます。ここで、

- OS は windows、linux、または mac
- version は現在の 3 桁のバージョン(X.Y.Z)

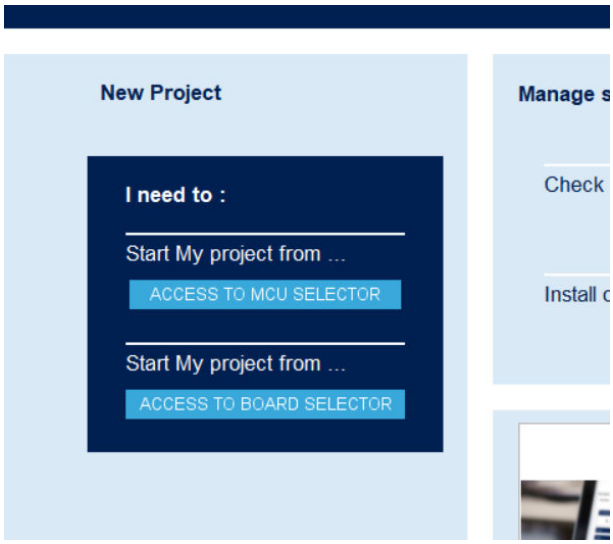
例：X-Cube-AI 7.1.0 の Windows® 固有の部分は、https://sw-center.st.com/packs/x-cube-ai/stm32ai-windows-7.1.0.zip からダウンロードできます。

3 新しい STM32 AI プロジェクトの開始

3.1 マイクロコントローラ·セレクタとボード·セレクタ

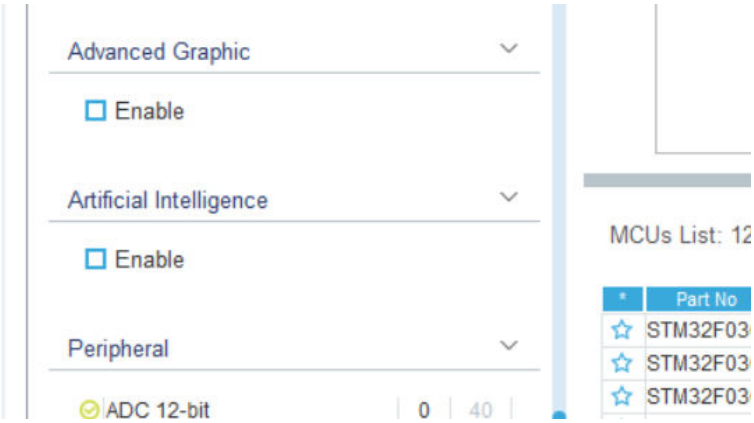
STM32CubeMX アプリケーションを起動後、[ACCESS TO MCU SELECTOR] または [ACCESS TO BOARD SELECTOR] ボタンをクリックします。または、[File]>[New Project...] を選択するか、CTRL-N ショートカットを選択します。

図 8. 新規プロジェクトの作成



ここでは、標準の STM32CubeMX フローを使用して、特定のマイクロコントローラまたはボードを選択できます。オプションのマイクロコントローラ·フィルタ·エントリにより、内蔵メモリ（RAM、Flash メモリ、または両方）の不足により最適化された STM32 NN ライブラリを保存できないマイクロコントローラを除外できます。この特定の AI フィルタを図 9 に示します。

図 9. AI フィルタ



注 この機能は、ボード·セレクタでは使用できず、1 つの NN モデルでのみ使用できます。

図 10 は、DL モデルがアップロードされ、デフォルトのオプションで解析された場合を示しています。パブリック·ドメインの学習済みの NN モデル（Keras タイプ）が使用されます。Keras で CNN を使用した人間行動認識の例を示しています。

図 10. デフォルト·オプションのある AI フィルタ

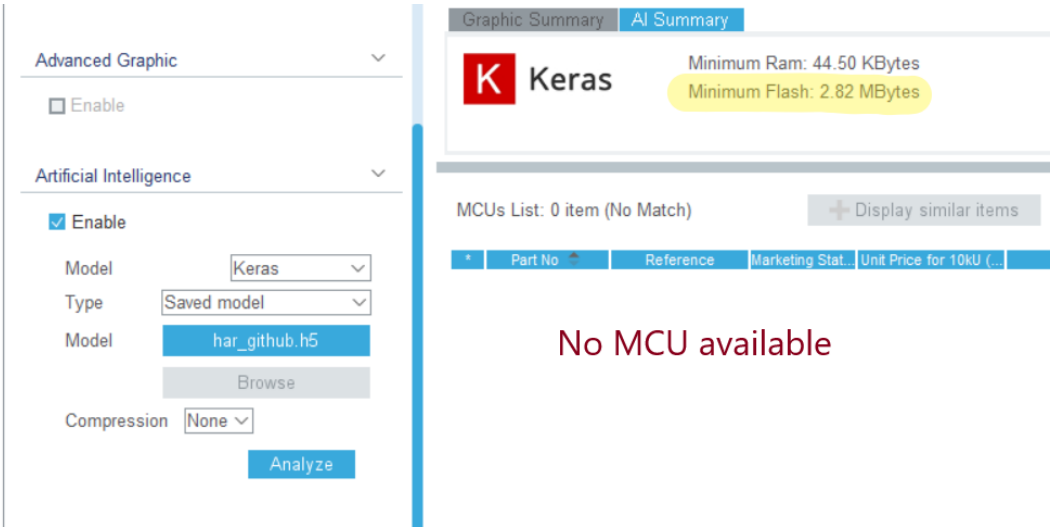
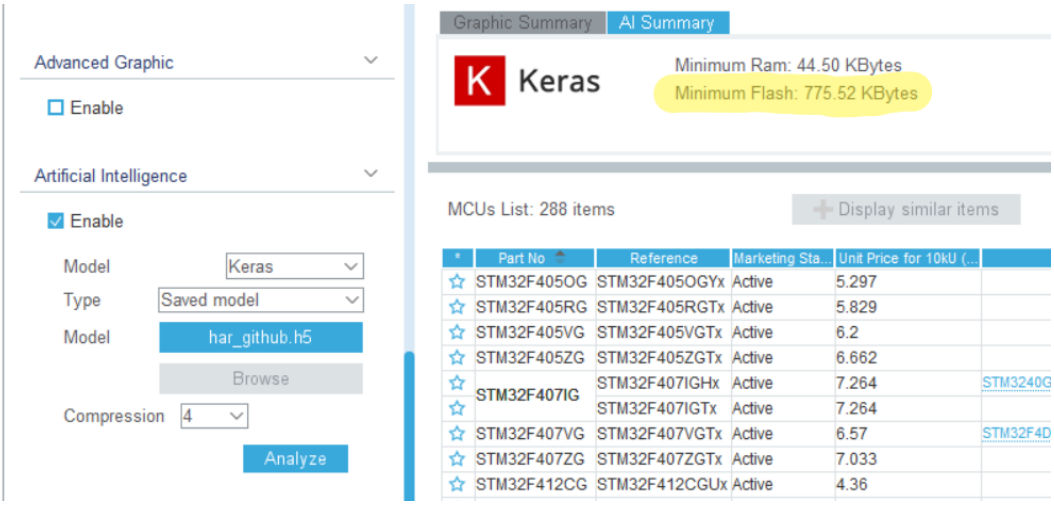


図 11 は、圧縮係数 4 が適用された場合を示しています。

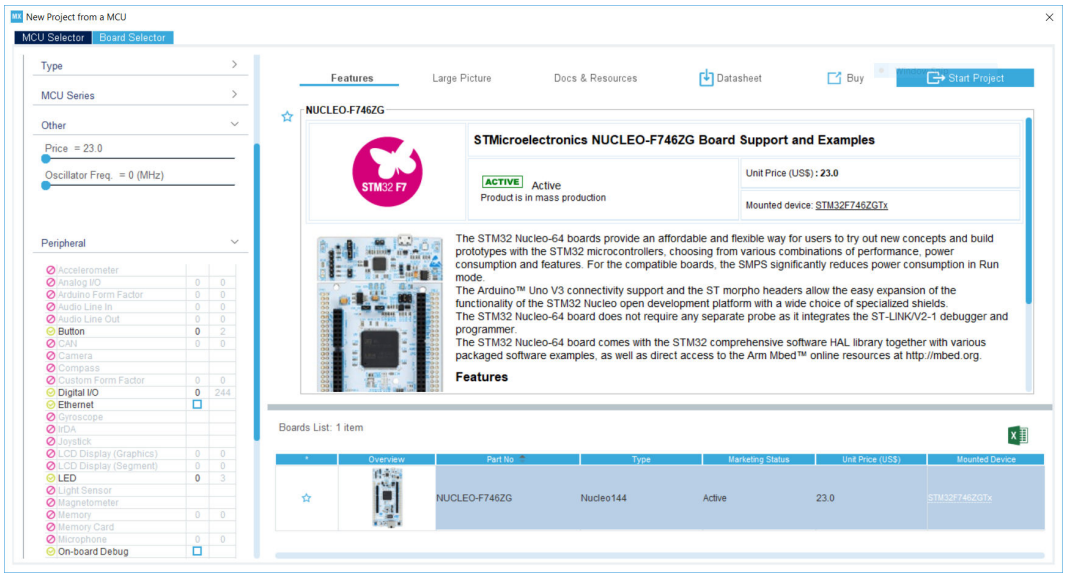
図 11. 4 倍圧縮機能付き AI フィルタ



注　メモリ·サイズは、NN ライブラリの生成中にオプティマイザでもチェックされます。これは、選択したマイクロコントローラに応じて最小 RAM と Flash メモリ のサイズの制約が無視される場合に、ユーザに通知するためです。

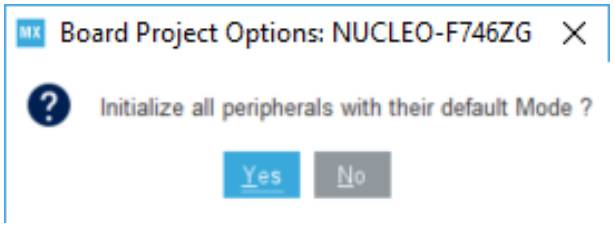
続行するには、図 12 に示すように、NUCLEO-F746ZG 開発キットを選択します。

図 12. NUCLEO-F746ZG ボードの選択



[Start Project] ボタンをクリックして続行し、すべてのペリフェラルをデフォルト·モードで初期化する必要があることを確認します。

図 13. すべてのペリフェラルの初期化

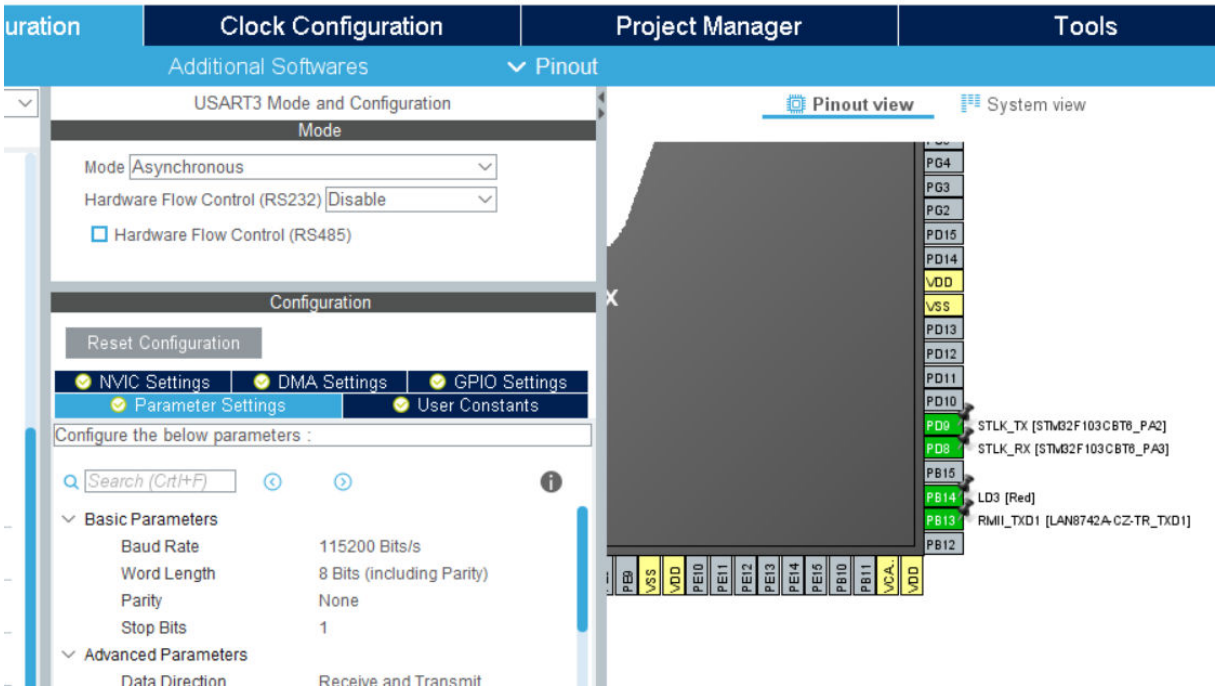


3.2 ハードウェアおよびソフトウェアのプラットフォーム設定

マイクロコントローラまたはボードを選択すると、関連する STM32 のピン配置が表示されます。このウィンドウで、ユーザは 1 つ以上のソフトウェアやペリフェラルを追加し、またクロックを設定することで、プロジェクトをセットアップすることができます。

アドオン AI アプリケーション（セクション 4.1 X-Cube-AI コンポーネントの追加 を参照）を使用している場合は、ホスト開発システムとの USART ベースのリンクが期待されます。STM32 Nucleo-144 開発ボード では、ピン PD9 TX および PD8 RX は 仮想 COM ポート をサポートする ST-LINK ペリフェラルに接続されます（図 14 を参照）。

図 14. USART3 設定



NUCLEO-F746ZG では、クロックやメモリサブシステムの構成を追加することで、より高性能になることが期待されます。

3.2.1 CPU およびシステム・クロック周波数の増加または設定

1. Clock Configuration タブをクリックします。
   デフォルトで、このラボ（実験）では、システム・クロック（SYSCLK、HCLK）は 72 MHz です。
2. PLL ペリフェラル（および関連するクロック・ツリー）を自動的に設定するクロック・ウィザードを呼び出すために、HCLK（MHz）入力の青色のボックスに 216 と入力します（図 16 を参照）。図 15 に示すようにクロック・ウィザードのポップアップが表示された場合は、[OK] ボタンをクリックして続行します。

図 15. クロック・ウィザードのポップアップ

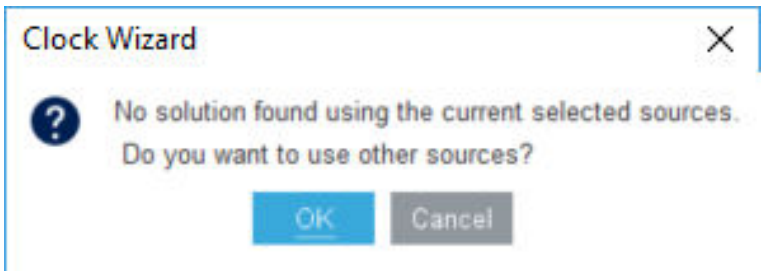
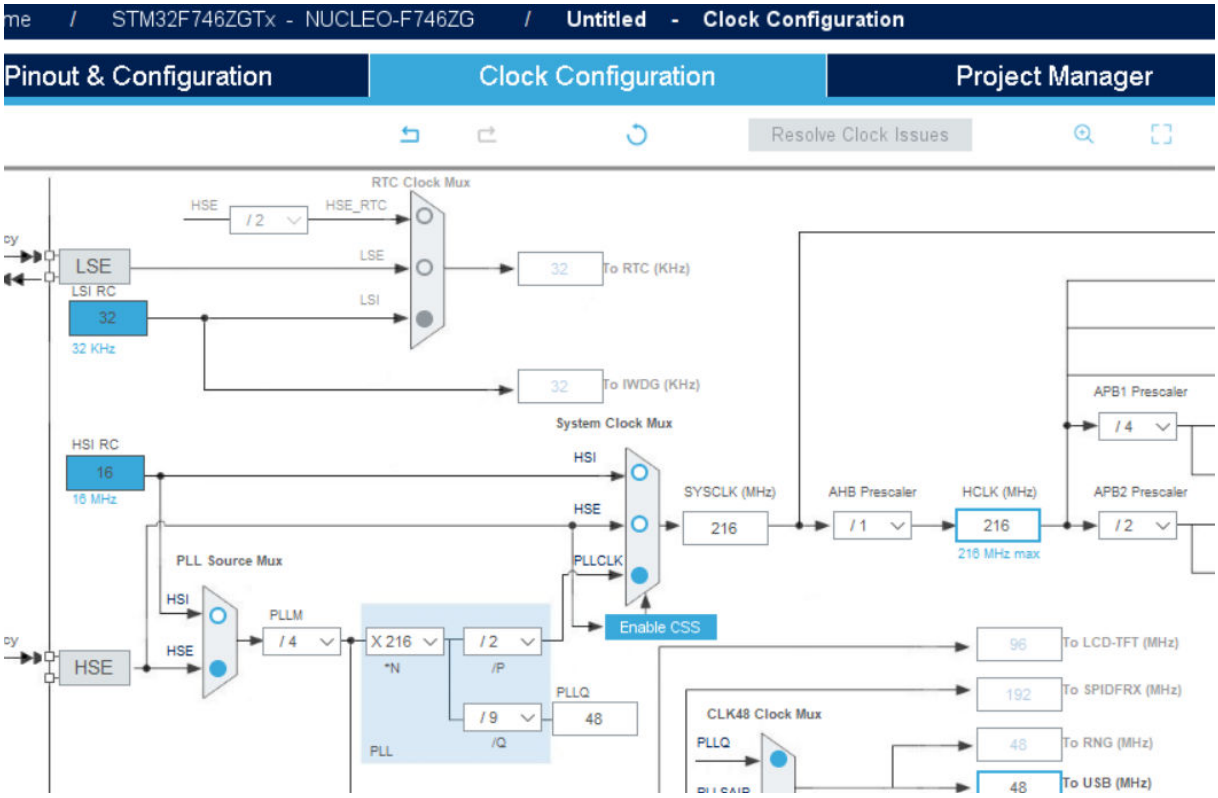


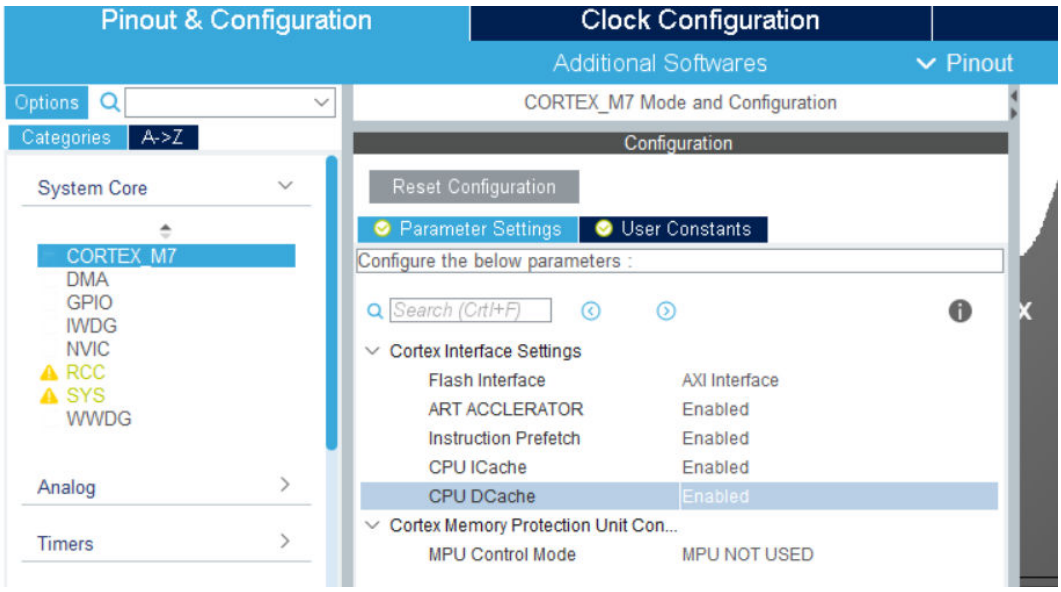
図 16. システム·クロック設定



3.2.2 マイクロコントローラ·メモリ·サブシステムの設定

- Pinout & Configuration タブ（図 17 を参照）から、Cortex®-M7 設定ウィザードを開くための [System Core]>[CORTEX_M7] エントリをクリックします。
  コアの命令キャッシュとデータ·キャッシュ、および ART アクセラレータ·サブシステムを有効にする必要があります。

図 17. マイクロコントローラ·メモリ·サブシステム（パラメータ設定）



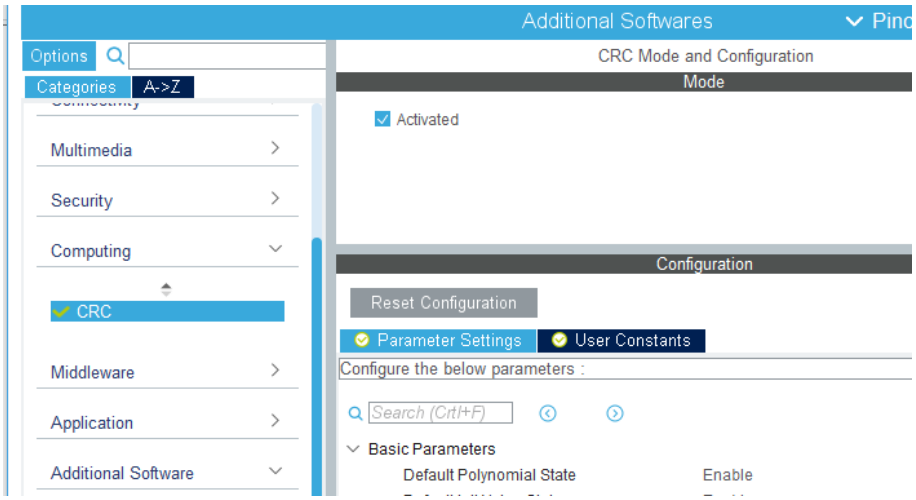
注　マイクロコントローラ·クロックの最大値の設定は必須ではありません。最終デザインで使用される設定に合わせる必要があります。Flash メモリ のウェイトステートの設定は、STM32CubeMX プラットフォーム·コード·ジェネレータによって自動的に調整されます。

3.2.3 CRC

CRC ペリフェラルは、NN ライブラリ ランタイム で保護されたメカニズムをサポートするように要求されており、有効にする必要があります。

注　これはツールによって自動的に行われるため、手動で行う必要はありません。

図 18. CRC ペリフェラルの有効化

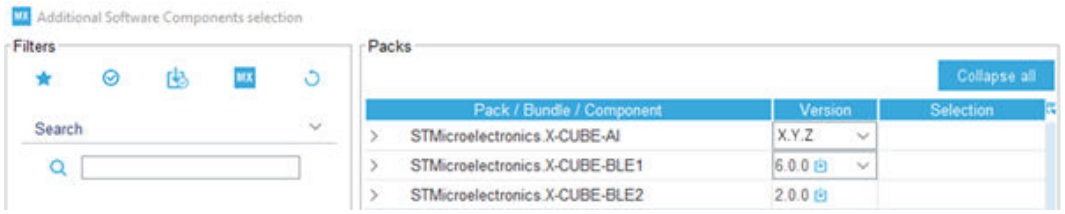


4 X-CUBE-AI 設定ウィザード

4.1 X-Cube-AI コンポーネントの追加

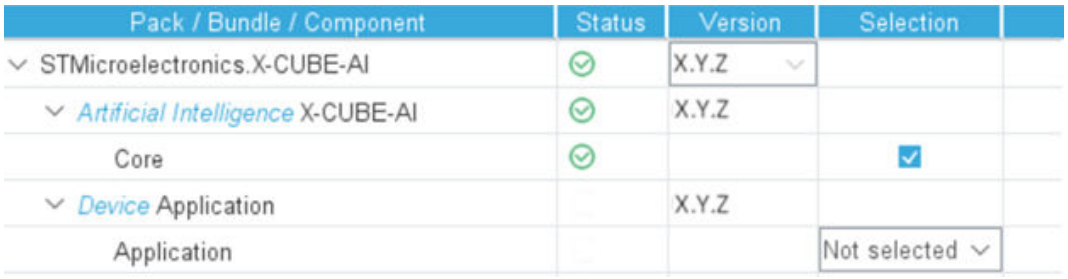
1. [Additional Softwares] ボタンをクリックしてプロジェクトに X-Cube-AI の追加ソフトウェアを追加します（図 19 を参照）。

図 19. 追加ソフトウェア・ボタン



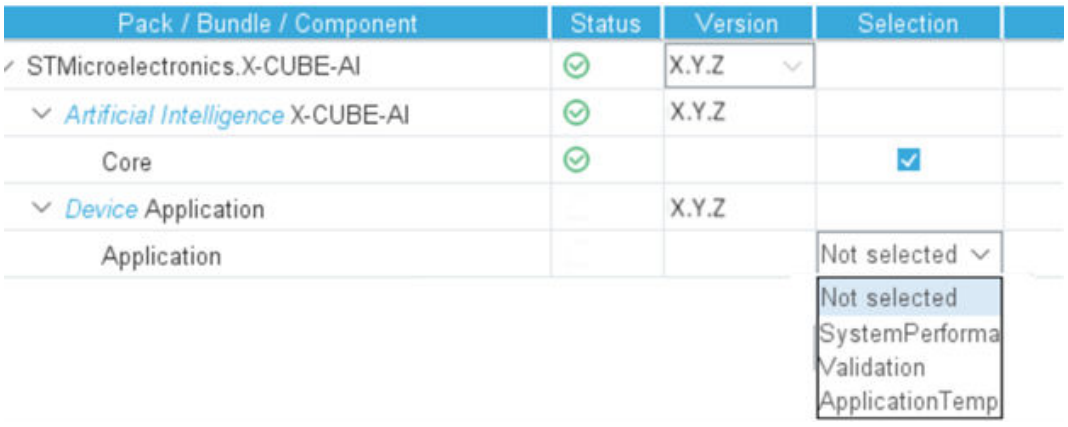
2. Additional Software Component Selection ウィンドウで、NN モデルをアップロードし、関連する STM32 NN ライブラリを生成できるよう、X-Cube-AI/コア・バンドル（図 20 を参照）を確認する必要があります。この場合、ライブラリはスタティック・ライブラリとして完全に統合されているため、ユーザは AI ベースのアプリケーション/ミドルウェアを、生成された明確に定義された NN API [10] の後に実装するだけです。

図 20. X-Cube-AI コア・コンポーネントの追加



3. オプションで、X-Cube-AI/アプリケーション・バンドルからアドオン X-Cube-AI アプリケーション（図 21 を参照）の 1 つを選択できます。
   - システム性能：性能を目的としたスタンドアロン AI テスト・アプリケーション
   - 検証：検証を目的とした AI テスト・アプリケーション
   - テンプレート・アプリケーション：AI アプリケーションの基本アプリケーション・テンプレート

図 21. アドオン X-CUBE-AI アプリケーション



4. [OK] をクリックし、選択を確定します。

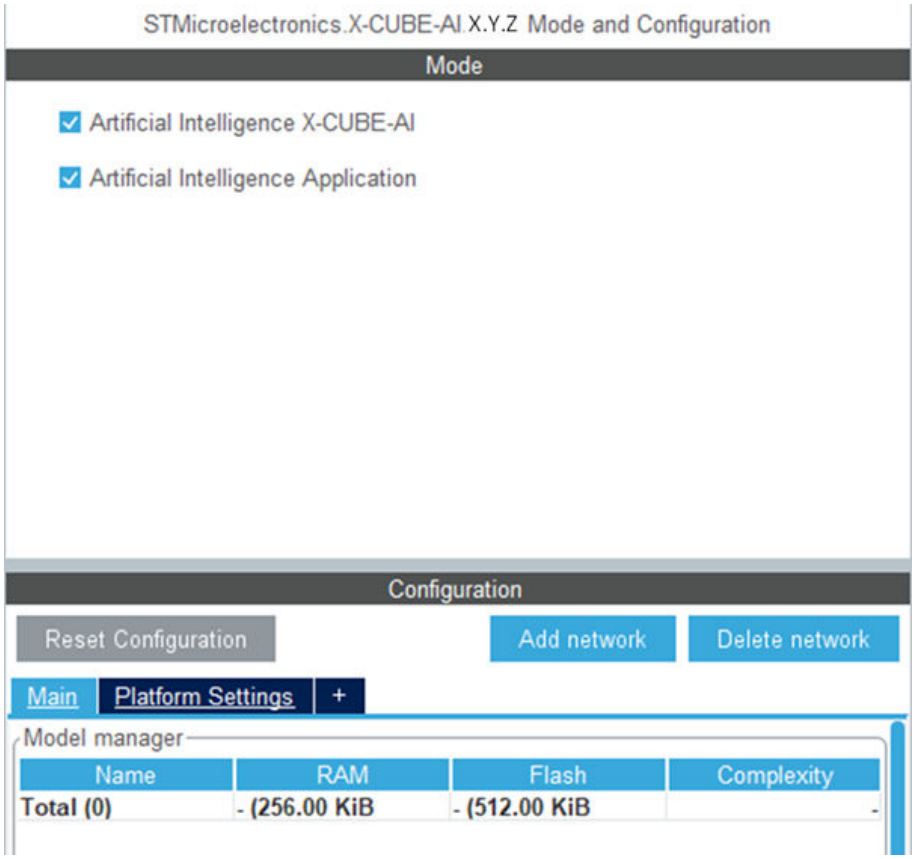
4.2 X-Cube-AI コンポーネントの有効化

X-Cube-AI コンポーネントを有効にして設定するには、次の追加手順が必要です。

1. Pinout & Configuration タブで、[Additional Software] セレクタをクリックして、追加のソフトウェアを検出します。
   [STMicroelectronics X-CUBE-AI X.Y.Z] をクリックして、AI の初期設定ウィンドウを開きます。

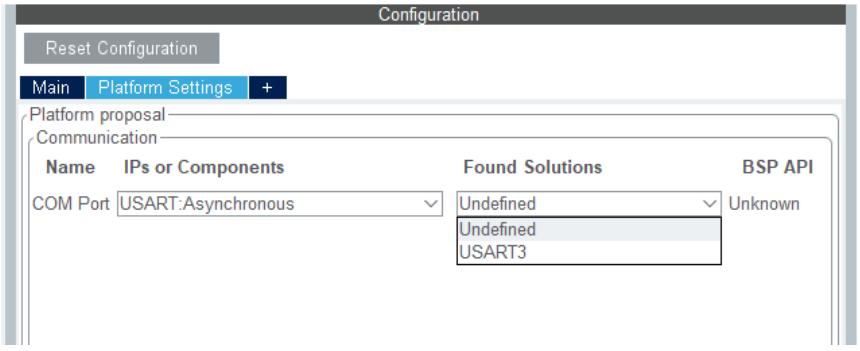
2. [Artificial Intelligence Core] にチェックを付けて、X-Cube-AI コア·コンポーネントを有効にします。アドオン AI アプリケーションを追加するには、[Artificial Intelligence Application] にもチェックを付ける必要があります。
ここで選択する AI アプリケーションは、前の手順で有効にしたアプリケーションに対応します（図 21 を参照してください）。

図 22. メイン X-CUBE-AI 設定パネル



– Main タブには、概要と、ネットワークを追加または削除するためのエントリ·ポイントがあります（それぞれ [Add model] および [Delete model] ボタン）。[+] を直接使用して、ネットワークを追加することもできます。
– Platform Settings タブは、情報のレポートに使用される USART ペリフェラル（AI システム·パフォーマンス·アプリケーション）またはホストとの通信（AI 検証アプリケーション）の取扱いを示します。

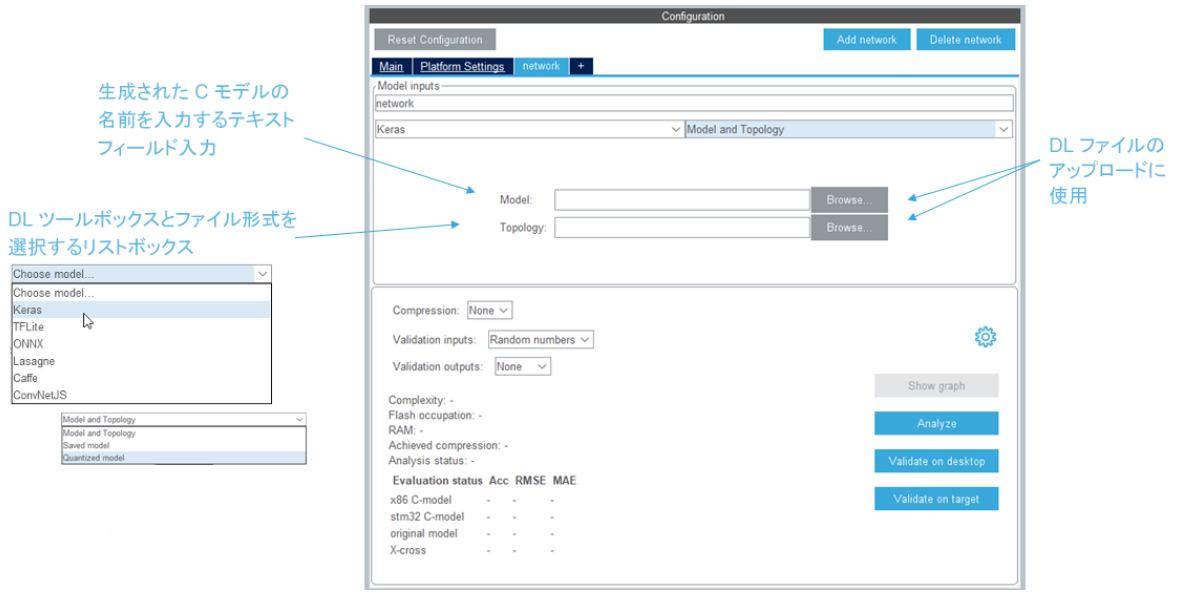
図 23. X-CUBE-AI プラットフォーム設定パネル



4.3 学習済みの DL モデル·ファイルのアップロード

メイン タブで、[Add model] または直接 [+] をクリックし、新しい専用の <model_name> 設定ウィザードを開きます。または、モデルが以前に MCU フィルタを通じて提供された場合は、ネットワーク タブを直接クリックして NN 設定 ペインを開きます。

図 24. NN 設定ウィザード



1. テキスト·フィールド入力は、ネットワークの 名前（最大 32 文字）を定義するために使用します。この文字列は、組込みクライアント推論 API の名前を生成するために直接使用されます（[10] を参照）。期待されるネットワークが 1 つのみである場合は、デフォルトのネットワーク文字列名を維持できます。
2. リスト·ボックスのエントリでは、DL モデル·ファイルおよび関連するファイル形式のエクスポートに使用する DL ツールボックスを指定します（詳細については、セクション 12 サポートされている 深層学習 のツールボックスおよびレイヤ を参照）。
   – [Browse..] ボタンをクリックして、ホスト·ファイル·システムから DL ファイルをアップロードします。このハンズオン·ラボでは、公開 Keras HAR モデル·ファイルがアップロードされます（保存されたモデル形式）。
3. [Analyze..] ボタンをクリックして、寸法情報をレポートするネットワークの事前解析をトリガします（システム統合の観点から）。圧縮係数はあらかじめ 4 に設定されています。そうでない場合は、図 25 に示すような警告メッセージのポップアップが表示されます。無効なネットワーク メッセージ·ボックスがポップアップ表示された場合は、ログ·コンソールで [Window]>[Outputs] を選択し、詳細を確認することができます（セクション 13 エラー処理 を参照）。最小 RAM、Flash メモリ の占有量、およびオリジナルの DL モデルの複雑度が更新されます（セクション 4.5 を参照）。
   報告された Flash メモリ と RAM のサイズは、ネットワークおよび関連する組込みライブラリのカーネルが使用するサイズの合計に対応します。

図 25. 不十分な RAM/Flash メモリ のメッセージ·ボックス

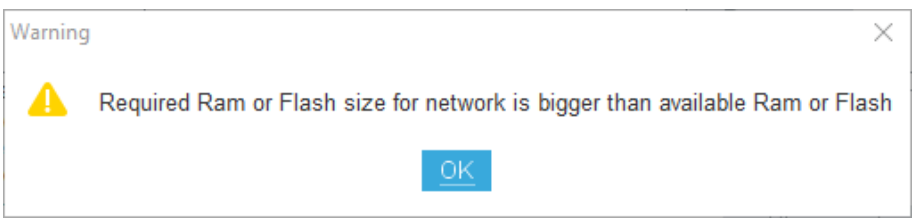
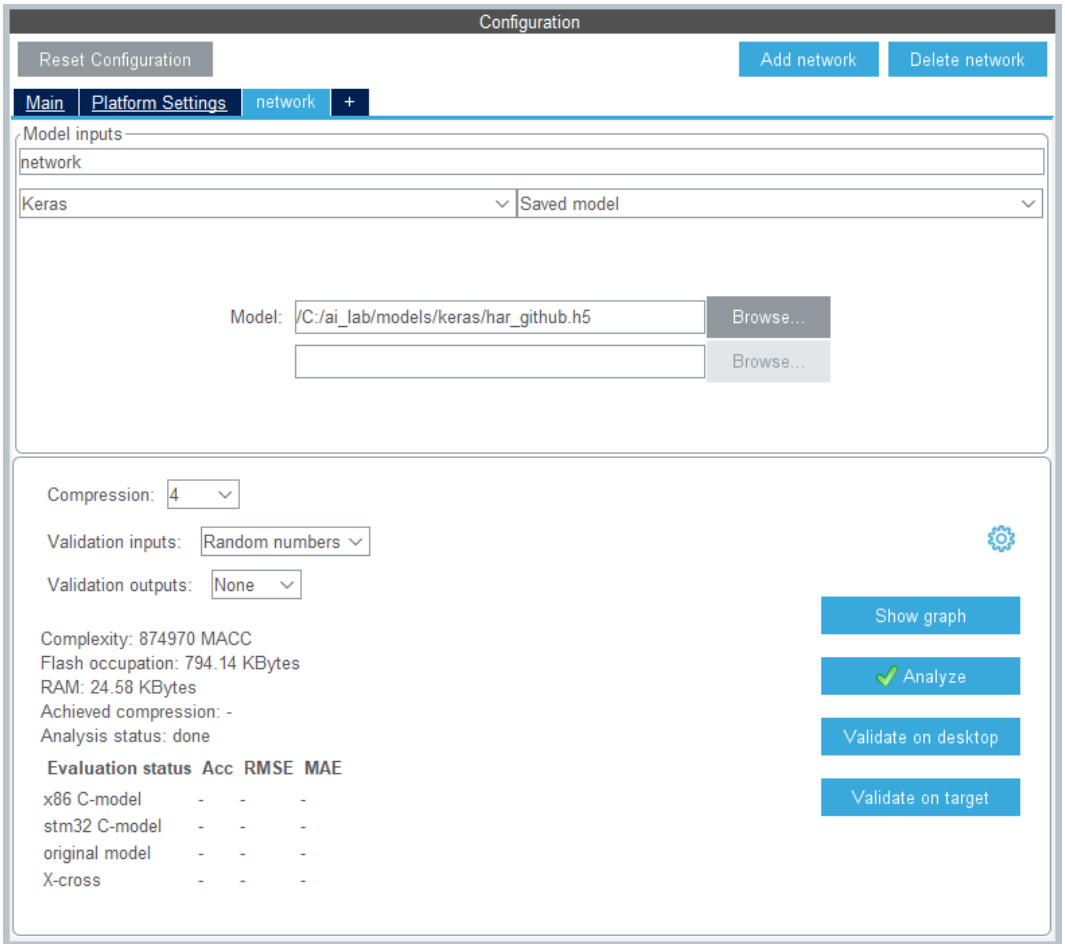


図 26. アップロードされ、解析された DL モデル



注 追加のデバッグ/ログ情報は、`C:\Users\<username> \.stm32cubemx\STM32CubeMX.log` または `$HOME/.stm32cubemx/STM32CubeMX.log` のファイルにあります。

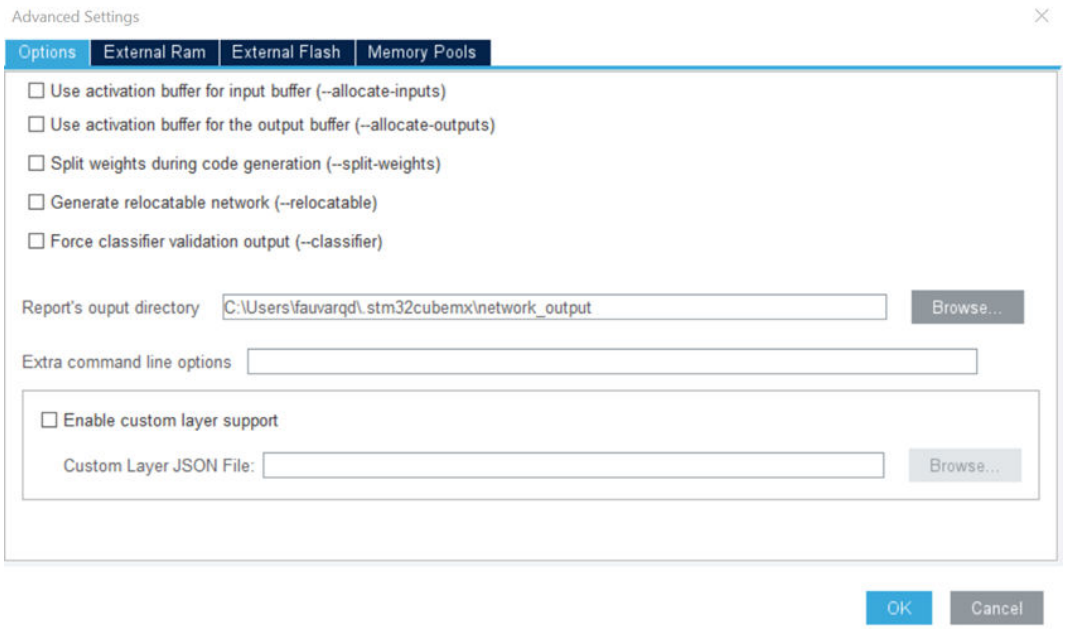
[Advanced Settings] ボタン( )をクリックすることで、拡張オプションを選択したり、カスタム·レイヤを設定するために、重みに外部 Flash メモリ、または重みやアクティベーションに外部 RAM を使用するよう、ネットワークを設定することができます。詳細については、詳細設定を参照してください。

外部 Flash メモリ または RAM が搭載された STM32 ST マイクロエレクトロニクス·ボードからプロジェクトを開始している場合、コード生成中の外部 Flash メモリ または RAM の設定は自動的に行われます。Flash メモリ または RAM を正しく初期化するために、STM32Cube MCU パッケージ で提供しているボードの BSP が使用されます。外部 Flash メモリ はメモリマップド·モードで使用されます。

4.4 詳細設定

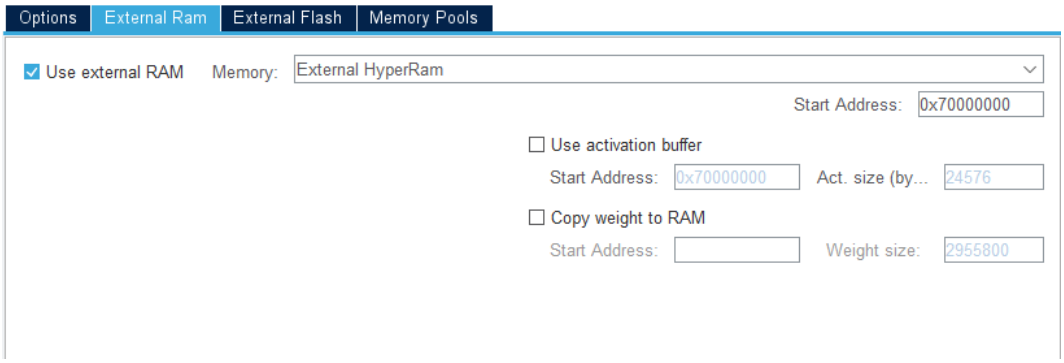
図 27 に、Advanced Settings ウィンドウの内容を示します。

図 27. 詳細設定



外部 RAM の使用

図 28. 外部 RAM の設定



外部 RAM は、以下の用途で使用できます。

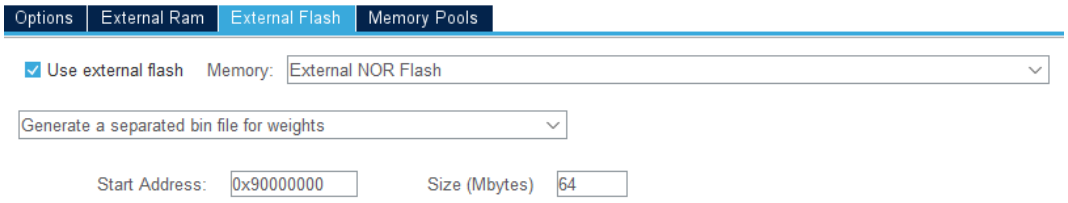
- アクティベーション・バッファ（ Use activation buffer option ）
- 重み（ Copy weight to RAM ）
- または、アクティベーション・バッファが物理メモリのいくつかのセグメントに分割されている場合に、メモリ・プールのプールの 1 つとして

外部 Flash メモリ の使用

[Use external flash] を選択すると、以下を行うことができます。

- 別の `network_data.bin` ファイルに重みを生成します。コードが、外部 Flash メモリ の始まりのアドレスをポイントするように生成されます。

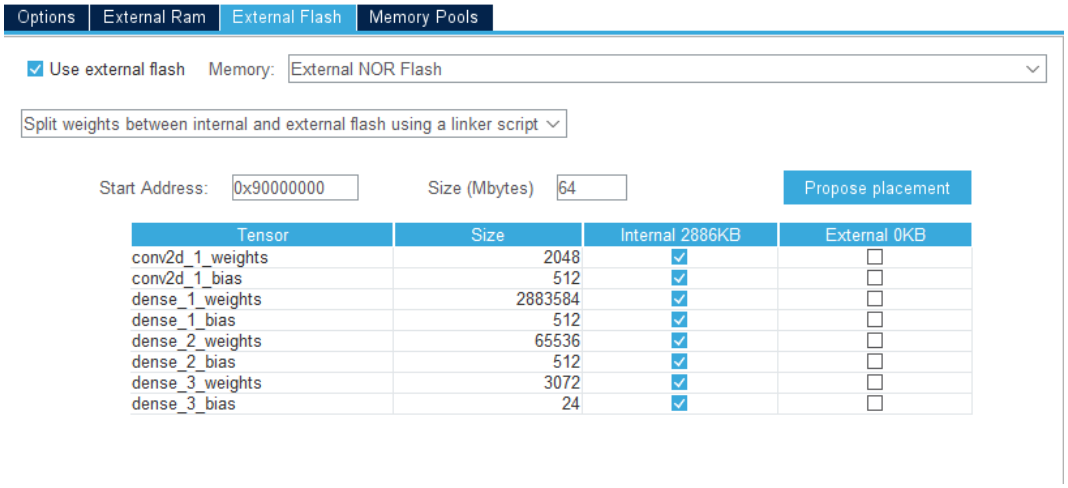
図 29. 別の重みファイル



`network_data.bin` ファイルは、STM32CubeProgrammer（STM32CubeProg）などのツールを使用して、ボードの外部 Flash メモリ に手動でプログラムする必要があります。
注：ターゲットで自動検証を使用すると、ファイル `network_data.bin` がボードの外部 Flash メモリ で自動的にプログラムされます。

- リンカ·スクリプトを使用して、重みを内部と外部の Flash メモリ に分割します。

図 30. 重みメモリ分割

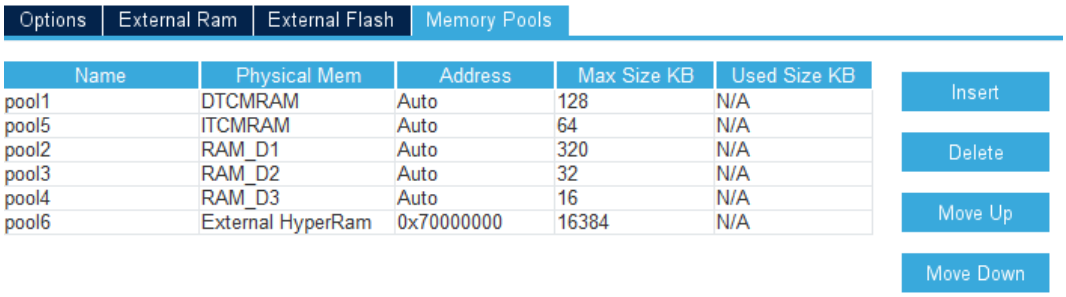


テンソル表で、テンソルを内部メモリに配置するか外部メモリに配置するかを選択します。
[Propose placement] ボタンは、内部メモリと外部メモリの間で重みを自動的に調整し、ライブラリ·カーネル空間とプログラム空間を確保しながら、内部メモリに最初に最大の MACC を持つ重みレイヤを配置することを試みます。

メモリ·プールの使用

コード·ジェネレータは、複数のメモリ·プール内にアクティベーション·バッファを生成できます。このツールは、External RAM タブで選択している場合、外部 RAM を含めた使用可能なすべての物理メモリを提案できます。

図 31. メモリ·プール設定



プールの割り当ては、表に示されている順に実行されます。異なるプールを上下に移動して、独自の割り当て優先順位を指定します。また、各物理メモリで使用できる最大メモリ·サイズを制限することもできます。

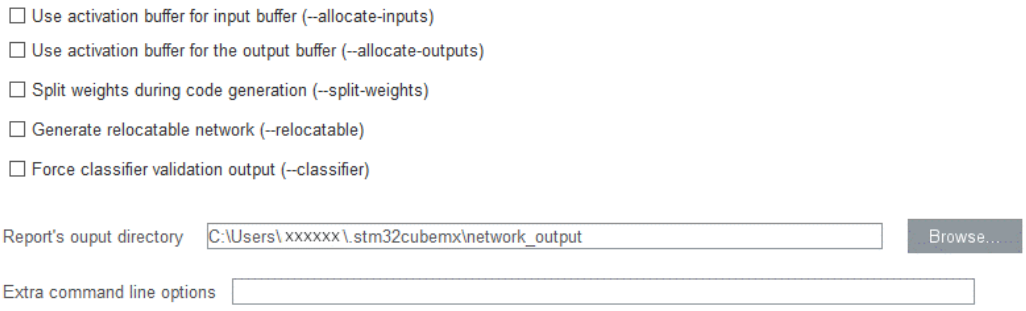
解析、検証、または生成の実行時に、各プールの実際の使用サイズが報告されます。このツールは、使用されていないプールがある場合も報告します。

メモリ·グラフには、各プールでのバッファ割り当ても表示されます。メモリ·プールはバッファ空間を再利用しない場合があるため、メモリ·プールを使用すると、全体の使用メモリ·サイズが大きくなることがあります。

Cortex®-M7 ベースの STM32 マイクロコントローラの外部 Flash メモリ または RAM を使用する場合、CORTEX_M7 CPU の ICACHE および DCACHE は自動的に有効になり、メモリ保護ユニットは自動的に外部 RAM または Flash メモリ へのアクセスを許可するように設定されます。

拡張オプション

図 32. 拡張オプション



- [Use activation buffer] チェックボックスを選択すると、[Start Address] フィールドで指定したアドレスにある外部 RAM に、アクティベーション·バッファが配置されます。
  オプションで、起動時に外部 RAM に重みをコピーすることができます。この場合、重みのコピー先のアドレスが要求されます。
- [Use activation buffer for input buffer] チェックボックスを選択すると、ユーザは個別の入力バッファを割り当てる必要はありません。事前に割り当てられたアクティベーション·バッファの空間に入力データを配置することができます。入力データのサイズによっては、アクティベーション·バッファの方が大きくなることがありますが、全体としては、アクティベーション·バッファと入力バッファを個別に合わせた合計よりも小さくなります。
- [Use activation buffer for the output buffer] チェックボックスを選択すると、ユーザは個別の出力バッファを割り当てる必要はありません。アクティベーション·バッファに自動的に割り当てられるため、全体の使用メモリを節約できます。
- [Split weights during code generation] を選択すると、生成された `<network name> _data.c` ファイルには重みの配列がレイヤごとに 1 つ作成されます。その後、必要に応じて、専用のリンカ·スクリプトを使用して、各配列を異なるメモリ·セクションに配置できます。

- [Generate relocatable network] チェックボックスを選択すると、ネットワークと組込み ニューラル·ネットワーク ライブラリに個別のバイナリを生成できるようになります。このバイナリは、ターゲットの任意の場所に配置でき、アドレスは初期化関数に渡されます。これにより、アプリケーション全体を再プログラムせずに、ネットワーク、重み、およびトポロジ全体を更新できます。組込み ニューラル·ネットワーク ライブラリをネットワークにリンクすると、最終バイナリには使用したカーネルのみが含まれます。このオプションは、単一ネットワークの場合にのみ使用できます。
- [Force classifier validation output] では、ニューラル·ネットワーク は分類器であり、出力時に結果のクロス·マトリックスを生成する必要があることを stm32ai 検証 に通知します。
- [Extra command line options] は、グラフィカル·ユーザ·インタフェースでは直接考慮されない任意の `stm32ai` コマンドライン·オプションを追加するためのフリーテキスト·フィールドです。

カスタム·レイヤ

図 33. カスタム·レイヤ

カスタム·レイヤは、現在の X-Cube-AI 機能を拡張する方法の 1 つです。そのパネルで、カスタム·レイヤ設定用の json ファイルを指定します。実装の詳細については、カスタム·レイヤに関する内部ドキュメントを参照してください。

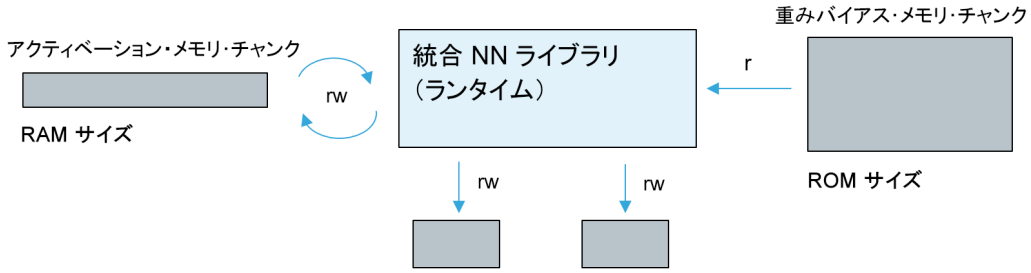
4.5 寸法情報レポート

DL モデルが処理されると、表 2 および 図 34 に示されている寸法システム情報が報告されます。

表 2. システム情報レポート

レポート情報	説明
RAM	中間推論計算値(.data または .bss セクション)の格納に使用される予期された RW メモリ·チャンクのサイズ(バイト単位)を示します。
ROM/Flash	要求された場合に、圧縮後に重み/バイアス·パラメータを格納するために生成された RO メモリ·チャンク(.rodata セクション)のサイズ(バイト単位)を示します。
Complexity	積和演算(MACC)におけるインポートされた DL モデルの機能的複雑性を示します。これには、アクティベーション関数の近似も含まれます(同じ 1 で表される)。

図 34. 統合 C モデル(ランタイムビュー)



注 AI の概要に記載されている RAM および Flash の最小サイズ要件は、ユーザ·アプリケーション(入力および出力テンソルを格納する RAM を含む)のメモリ制約を考慮しません。ここでは、DL モデルの重み/バイアスとアクティベーションメモリ要件についてのみ考慮されます。NN カーネル関数と特殊なモデル·コード(最小スタック/ヒープ·サイズなど)についても考慮されません。

4.5.1 CPU サイクル/MACC とは?

報告されている複雑性と、生成される NN C ライブラリの実際のパフォーマンスとの間に、理論的な関係は定義されていません（CPU サイクル/MACC）。ターゲットとなる環境（Arm® ツールチェーン、マイクロコントローラ、および基礎となるサブシステムのメモリ設定、NN トポロジとレイヤ、および適用された最適化などを含む）はさまざまです。そのため、オフラインで正確な CPU サイクル/MACC と STM32 のシステム設定を推定することは困難です。ただし、同梱サンプルで、次の概算値を使用できます（32 ビット浮動小数点 C モデルの場合）。

- STM32 Arm® Cortex®-M4/M33: ~9 サイクル/MACC
- STM32 Arm® Cortex®-M7:- ~6 サイクル/MACC

アドオンの AI システムのパフォーマンス テスト·アプリケーションは、実際のオンデバイス·パフォーマンスを報告するために特別に設計されています（セクション 9 AI システムのパフォーマンス·アプリケーション を参照）。

4.5.2 生成された C モデルのグラフ表現

[Show graph] ボタンをクリックすると、C コード·ジェネレータによって考慮されるアップロードされた DL モデルの主な構造情報を表示します。次の 3 つのグラフを使用できます。

1. 図 35 に示すような最適化を適用する前にインポートされた DL モデルの内部表現
2. 図 36 に示すようなすべての最適化後の生成された C コードの表現
3. レイヤをクリックして、図 37 に示すような、レイヤの追加情報取得

図 35. 最適化前のネットワーク

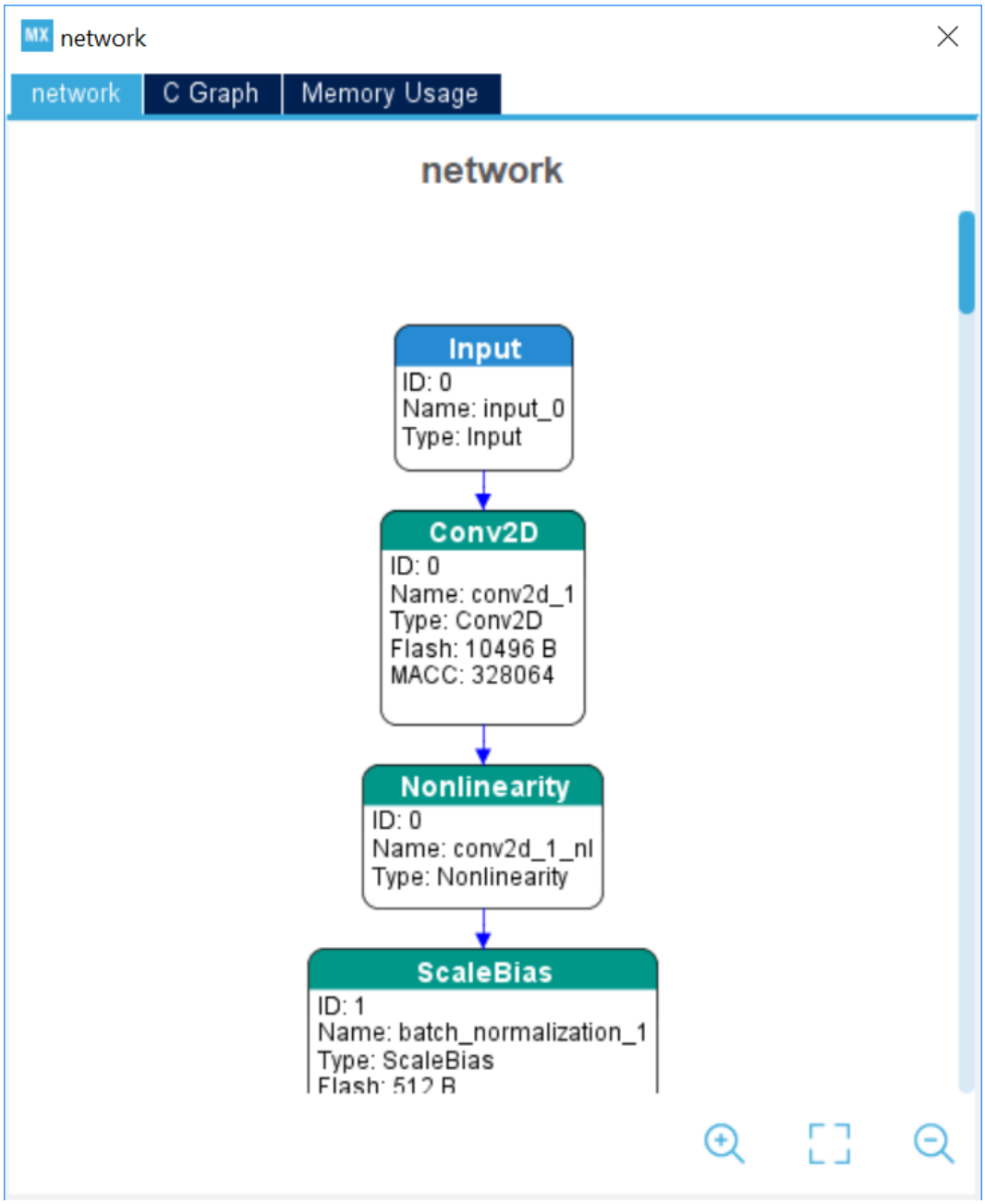
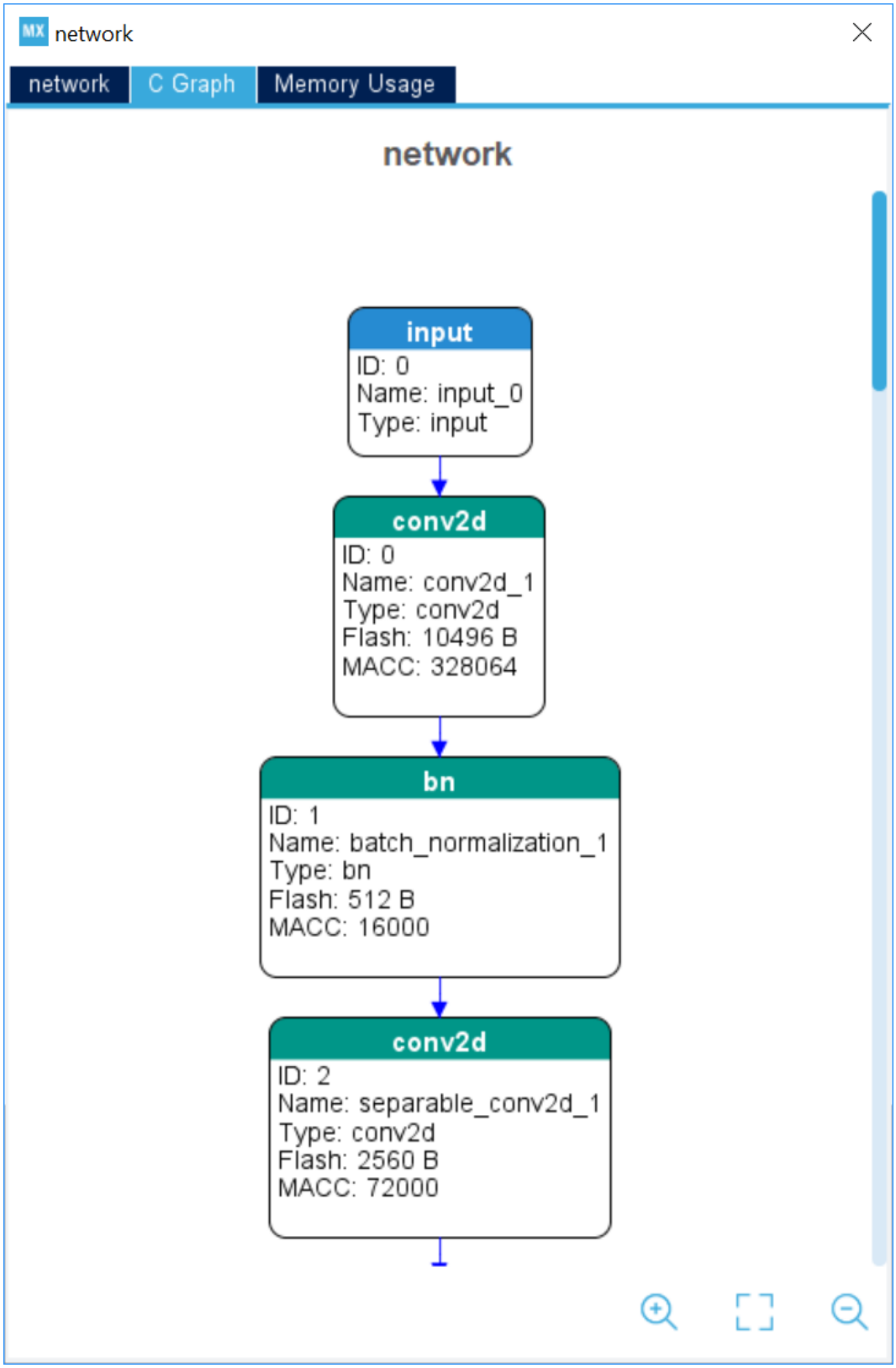
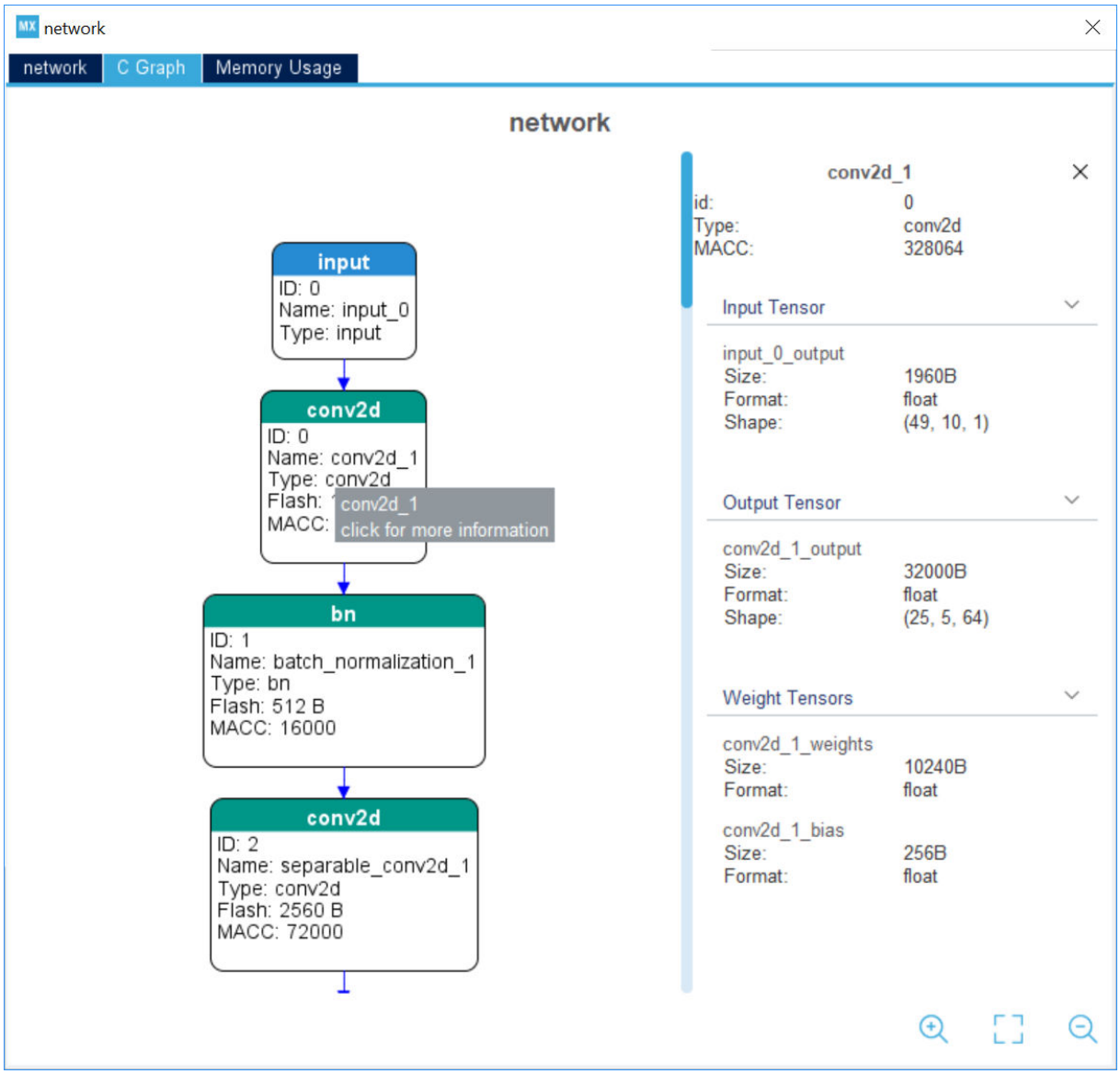


図 36. 生成コードの C グラフ



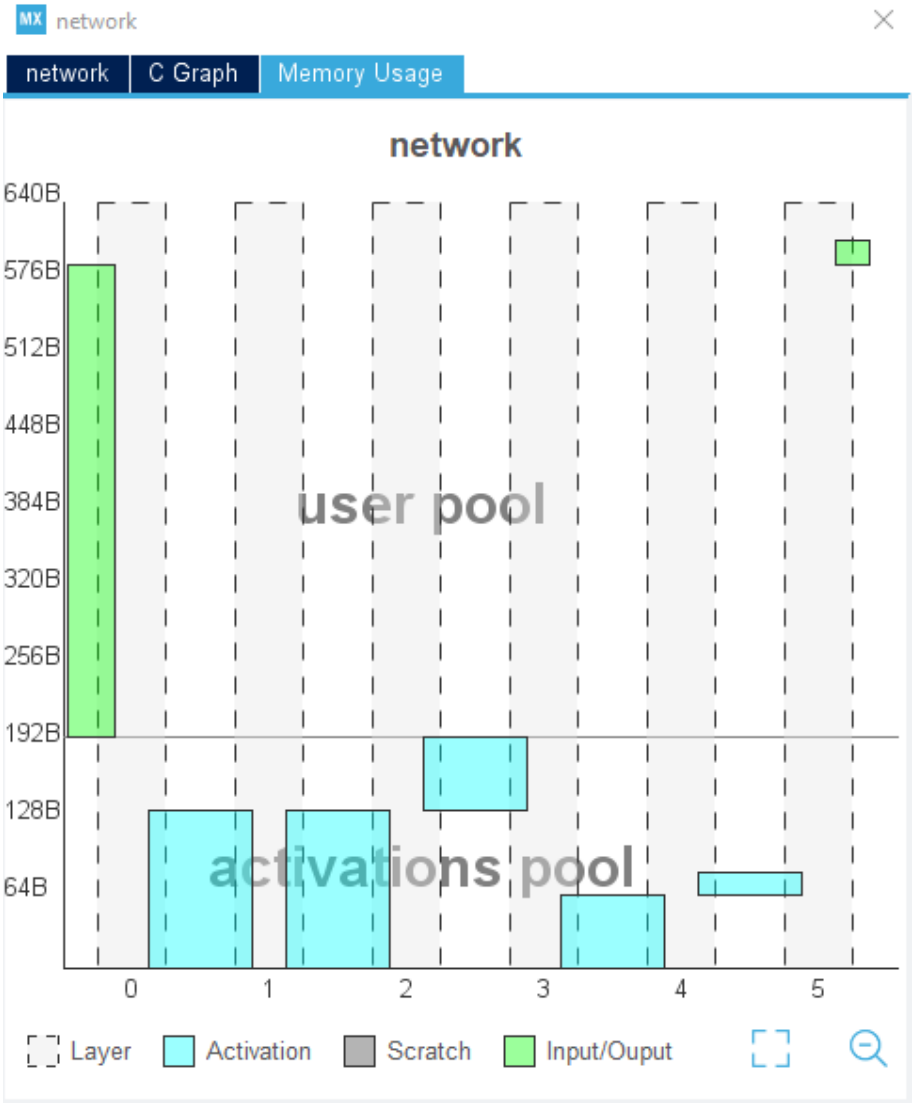
レイヤをクリックすると、そのレイヤの詳細が表示されます。

図 37. レイヤ情報



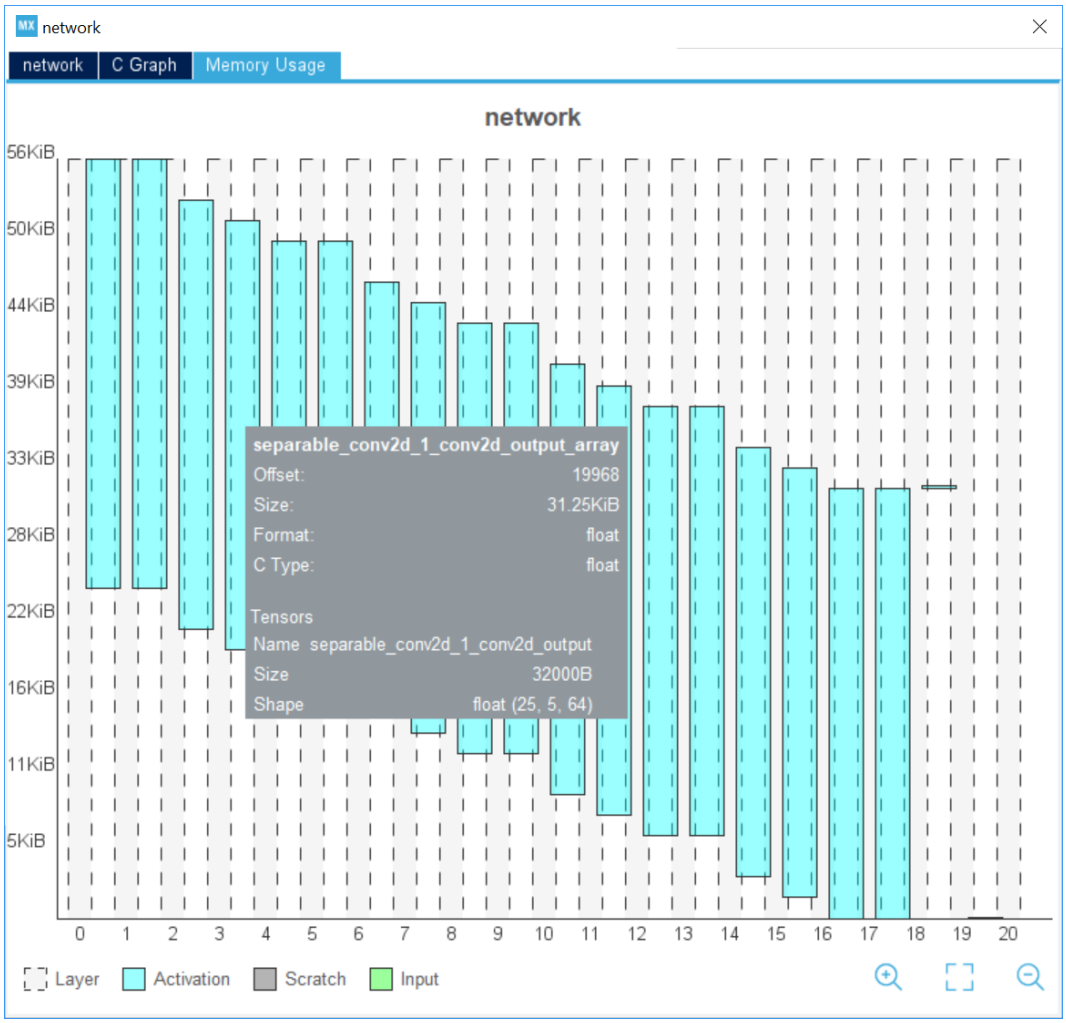
アクティベーション·バッファと内部バッファ用のメモリ使用量を 図 38 に示します。このビューには、ネットワークの実行に必要なメモリの総量が表示されます。入力バッファまたは出力バッファがアクティベーション·バッファに配置されると、ユーザ·プールには表示されなくなります。

図 38. メモリ使用量



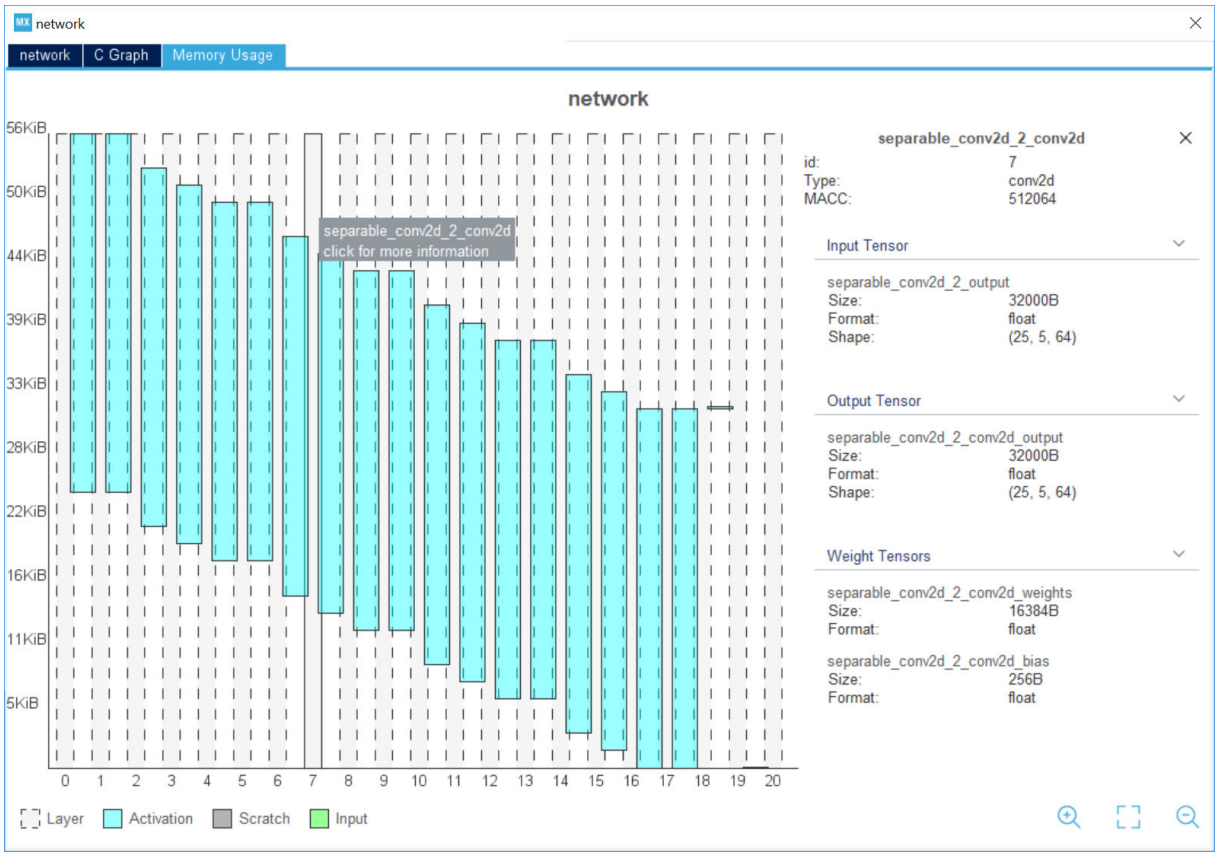
メモリ·グラフでは、バッファのツールヒントにバッファの内容が表示されます。

図 39. ツールヒントとしてのバッファの内容



同じビューで、レイヤをクリックすると詳細情報が表示されます。

図 40. レイヤの詳細情報



4.6 生成された C モデルの検証

[Validate on desktop] ボタンをクリックして、生成された C モデルの検証プロセスを開始します。使用可能なカスタム・データに応じて、さまざまなメトリックが計算されます（セクション 6.2 検証エンジン を参照）。この手順はオプションですが、特にインスタンスの圧縮係数が適用される場合に推奨します（セクション 6.1 グラフ・フローとメモリ・レイアウト・オプティマイザ を参照）。リファレンスまたはグラウンドトゥルースの出力値が関連する入力サンプルとともに提供された場合、表 3 にリストされたメトリックを計算するために予測値が使用されます（[11] を参照）。

表 3. メトリック

メトリック	説明
ACC	分類精度
RMSE	二乗平均平方根誤差（分類精度）
MAE	平均絶対誤差
L2r	L2 相対誤差

図 41. 検証ステータス·フィールド

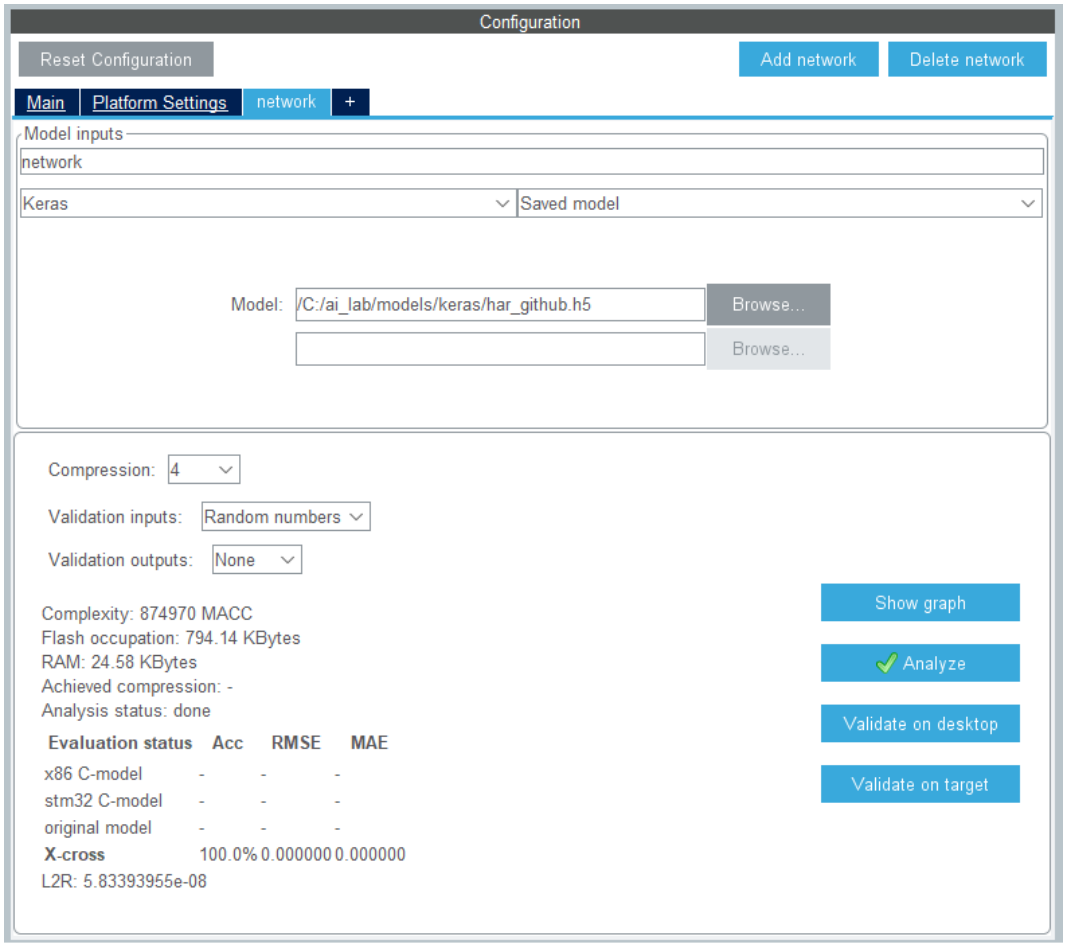
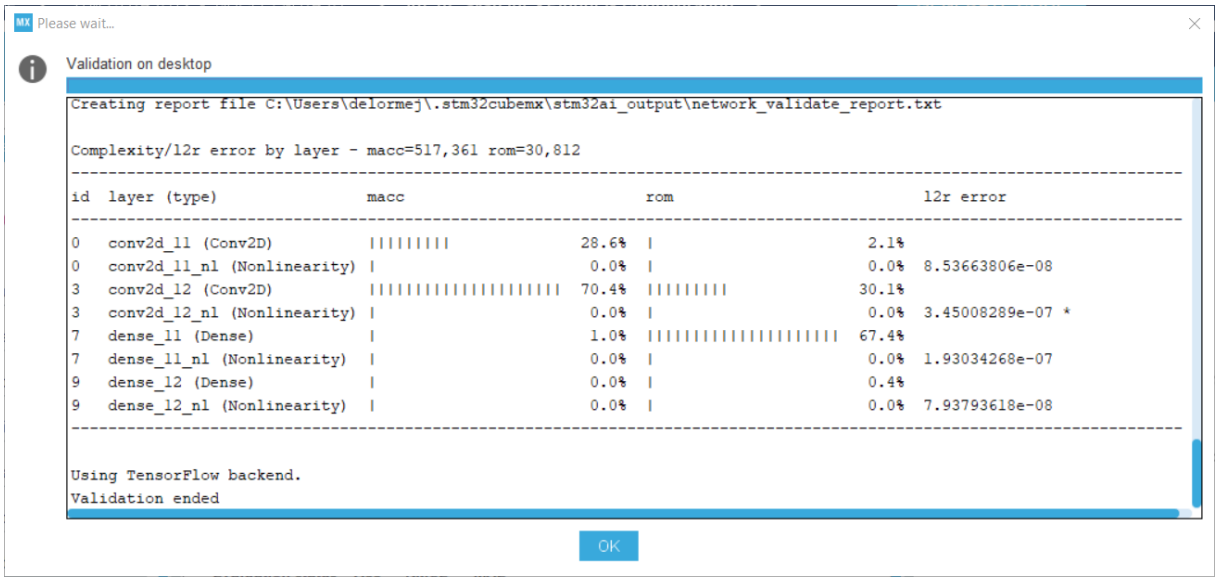


図 42 に示すように、より詳細な情報が UI ログ·コンソールに報告されます。特に、元のレイヤと一致する生成された C レイヤごとに L2r エラーも報告されます。

図 42. デスクトップで検証 - ログ·レポート

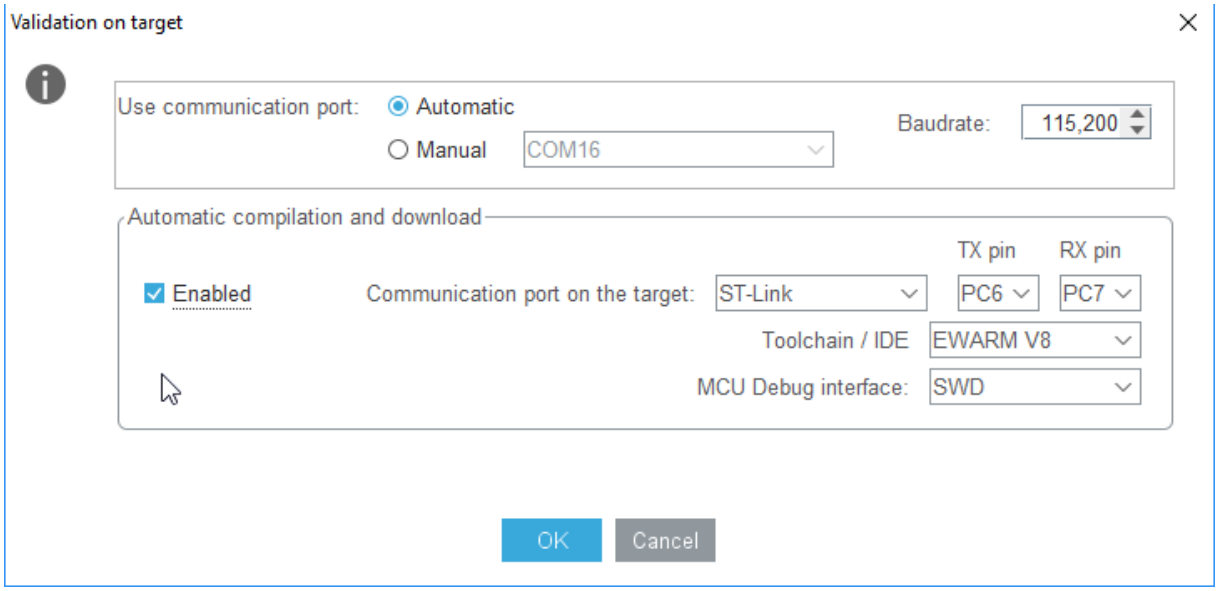


注 この手順では、アップロードされた DL モデルを生成された IDE プロジェクトに統合できるようになります。

ターゲット・デバイスを特殊なテスト・アプリケーション AI 検証 でプログラムしている場合、[Validate on target] オプションは必ず後で使用してください。これは、前の手順（セクション 4.1 X-Cube-AI コンポーネントの追加 の手順 3）で選択する必要があります。レポートされる情報と使用方法については、セクション 10 AI 検証アプリケーション に詳しく説明されています。

[Validate on target] ボタンを使用すると、ユーザはターゲットの検証を実行し、現在のネットワークに対応する一時プロジェクトをオプションで自動的に生成、コンパイル、プログラムして実行することができます。

図 43. ターゲットの検証



自動コンパイル、プログラミング、および実行を機能させるには、ターゲット上の提案された通信ポートが、仮想 COM ポート の ST-LINK に接続された USART（UART、LPUART）に対応していることを確認します。オプションで、ペリフェラル・インスタンスに、送受信信号に使用したピンを強制的に使用することができます。

デフォルトでは、プロジェクトに提案されたツールチェーンが選択されます。必要に応じて更新できます。

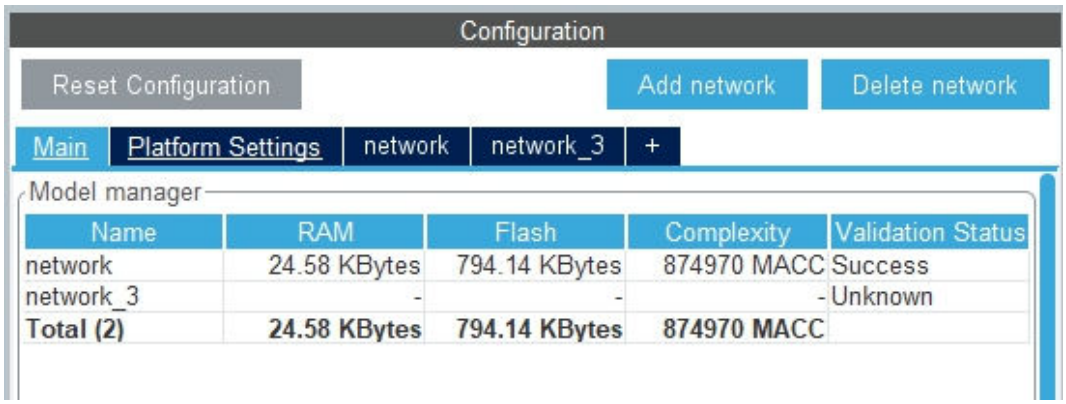
JTAG インタフェースを使用してマイクロコントローラのプログラミングを行う場合、デフォルトのデバッグ・インタフェースを変更する必要があります。

[OK] を押すと、一時プロジェクトが生成、コンパイル、プログラムされ、ターゲット上で開始されます。その後、ネットワークの定期的な検証が行われます。

4.7 新しい DL モデルの追加

複数の DL モデルをインポートできます。ウィザードが合計数を制限することはありませんが、初期制限値は主に、選択した STM32 マイクロコントローラ·デバイスで利用可能な RAM と Flash メモリのサイズに関連しています。[+] ボタンをクリックして新しい DL モデルをインポートし、前と同じ手順を適用します。メイン ビューに、RAM と Flash メモリの総占有率の要約を示します。

図 44. 複数ネットワークのあるメイン·ビュー



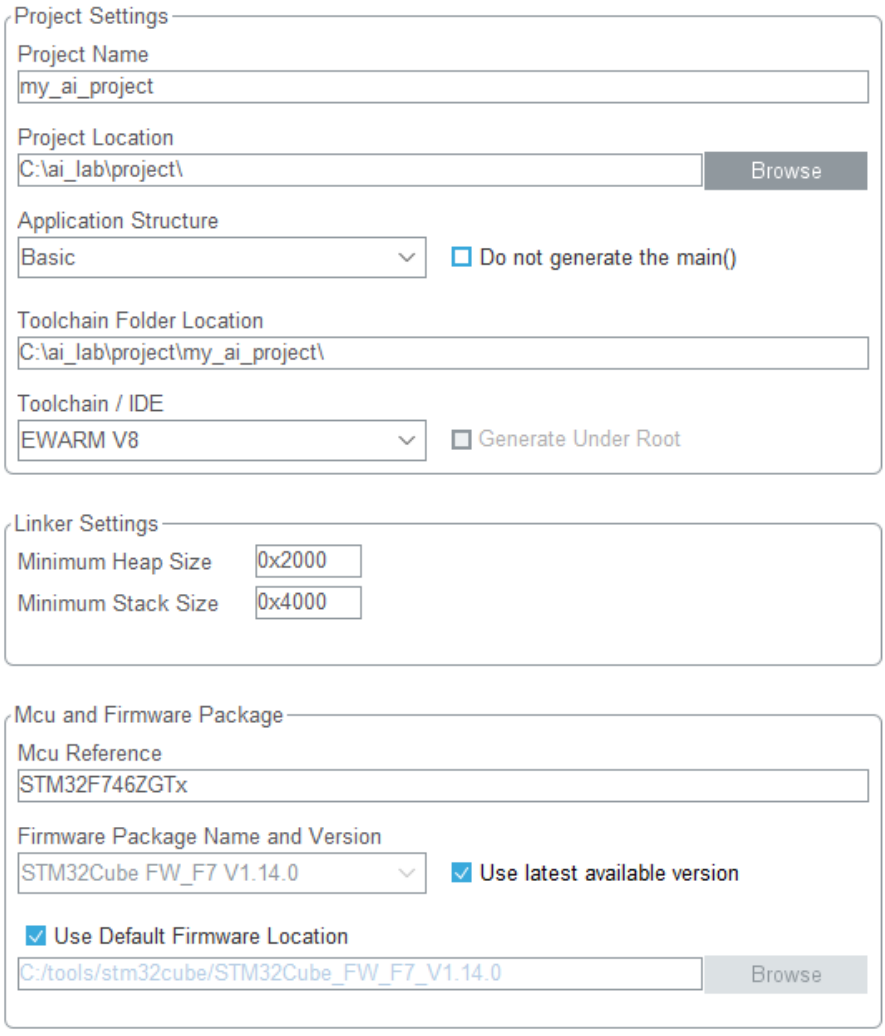
5 生成、構築、およびプログラミング

5.1 IDE プロジェクトの生成

次の手順では、特定の AI 拡張を追加せずに IDE プロジェクトを生成するために、従来の STM32CubeMX プロセスを順番に示します。

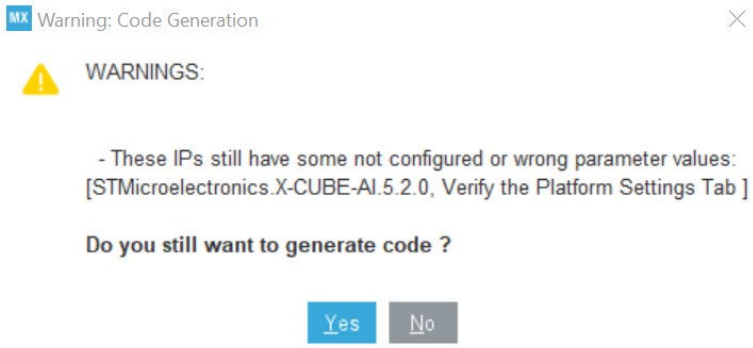
1. Project Manager ビューをクリックします。
2. プロジェクトの場所と名前を設定します。
3. 1 つのツールチェーンと IDE（IAR Systems 社の IAR Embedded Workbench® 用の EWARM、Keil® MDK-ARM、または STM32CubeIDE など）を選択します。
4. 初回でオーバーフローの可能性を最小化するために、ヒープ/スタック・サイズが正しく設定されていることを確認します（ AI Validation テスト・アプリケーションでは最小 2 KB のヒープを期待）。

図 45. IDE コード・ジェネレータのプロジェクト設定ビュー



5. [GENERATE CODE] ボタンをクリックして、現在のプロジェクト設定（IDE プロジェクト・ファイルを含む）に対応するコードを生成します。
ユーザがアドオン AI アプリケーションを選択して有効にし、予期されるプラットフォーム依存性（USART ハンドルなど）を設定し忘れた場合、IDE プロジェクトの生成中に、図 46 に示すメッセージ・ボックスがポップアップ表示されることがあります。詳細については、セクション 4.1 X-Cube-AI コンポーネントの追加を参照してください。

図 46. 完全には設定されていない AI ペリフェラル



この段階で、STM32CubeMX UI アプリケーションを終了できます。後で `<project_name> .ioc` ファイルを使用して再度開き、新しいペリフェラル、ミドルウェア・コンポーネント、またはその両方を有効にするか、ターゲットの検証 プロセスを実行することができます。

5.2 構築とプログラミング

STM32CubeMX ツールによって IDE プロジェクトが正常に生成されると、STM32 ボード開発キットまたはカスタマ・ボードのビルドとフラッシュに、標準ビルド・プロセスが使用されます。

1. IDE アプリケーションを起動し、生成したプロジェクト・ファイルを開きます。
2. ファームウェア・イメージをビルドしてフラッシュします。AI テスト・アプリケーションが選択された場合、コードの変更や更新は期待されていません。そうでない場合、生成された推論 C API を使用するには、ユーザの AI ベースのアプリケーション・コードを追加する必要があります。

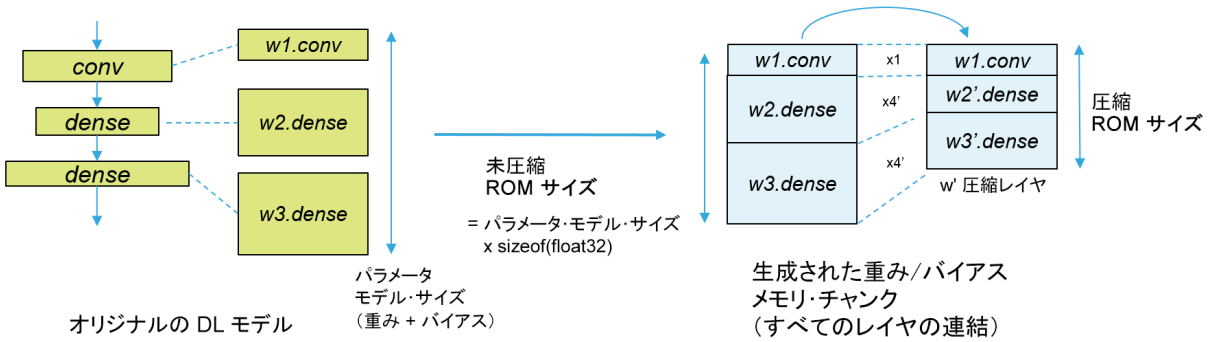
6 X-CUBE-AI の内部

6.1 グラフ·フローとメモリ·レイアウト·オプティマイザ

C コード·ジェネレータ(オプティマイザ)エンジンは、推論計算時間に対してメモリ使用量(RAM & ROM)を最適化します(消費電力も考慮される)。これはデータセットなしのアプローチに基づいています。つまり、圧縮および最適化アルゴリズムを適用するために、学習済みの有効なデータセットやテスト·データセットは要求されません(初期モデルの精度を保持するための再度学習/改良された重み/バイアスの段階は期待されていない)。

- 重み/バイアス圧縮(対象係数:なし、x4、x8)
  - 密(または全接続)レイヤ·タイプにのみ適用可能
  - 重み共有ベースのアルゴリズムを適用(K 平均法クラスタリング)
  - なし の場合、初期 DL モデルの精度は保証されます。残留誤差(~10^{-08})は、使用する 32 ビット C 浮動小数点のサイズに対するネイティブモデルの浮動小数点 64 ビットのサイズに関連しています。ただし、大規模なネットワークの場合は 10^{-06} の方が一般的です。

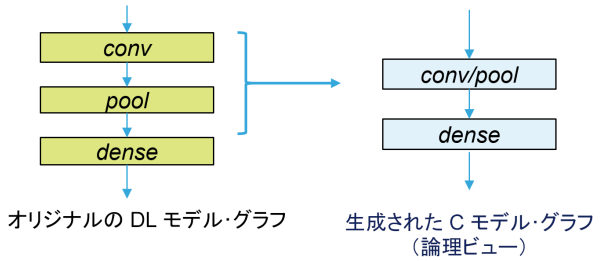
図 47. 重み/バイアス圧縮



この手法の利点は、圧縮プロセスが高速になることですが、最終結果は可逆的ではなく、グローバルな精度に影響する可能性があります。生成された C モデルの 検証 プロセスは、生成されたエラーを評価するための緩和策として、提供されています(セクション 6.2 を参照)。

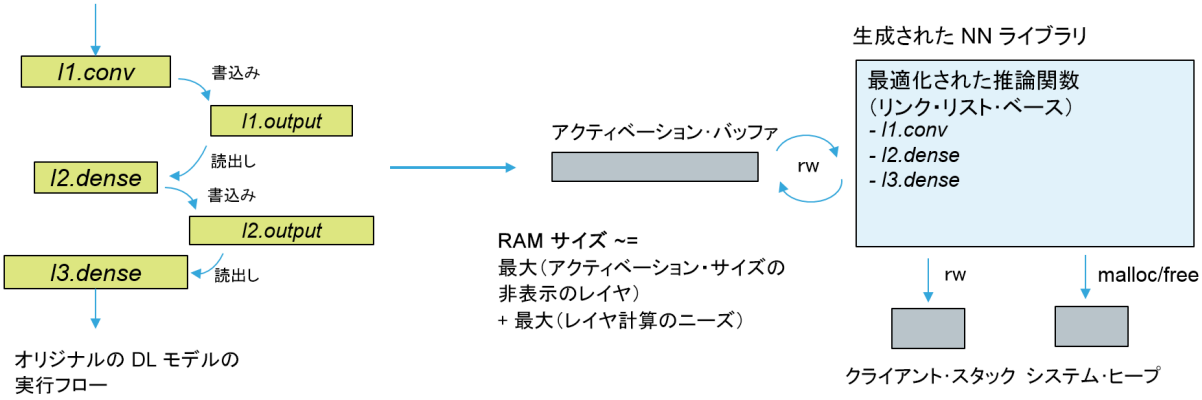
- 操作の融合
  - 2 つのレイヤを結合して、データ配置と関連する計算カーネルを最適化します。一部のレイヤ(低損失、再構成)は変換または最適化中に削除され、その他(畳み込み層後の非線形性やプーリングなど)は前のレイヤで融合されます。このため、変換されたネットワークは、元のネットワークと比較して層数が少ないことがしばしばあります。

図 48. 操作の融合



- 最適なアクティベーション/作業メモリ:R/W チャンクは、一時的な非表示層の値(アクティベーション演算子の出力)を格納するために定義されます。これは、推論機能によって使用されるスクラッチ・バッファとみなすことができます。アクティベーションメモリは、さまざまな層にわたって再利用されます。このため、アクティベーション・バッファ・サイズは、連続する 2 つのレイヤの最大メモリ要件によって定義されます。

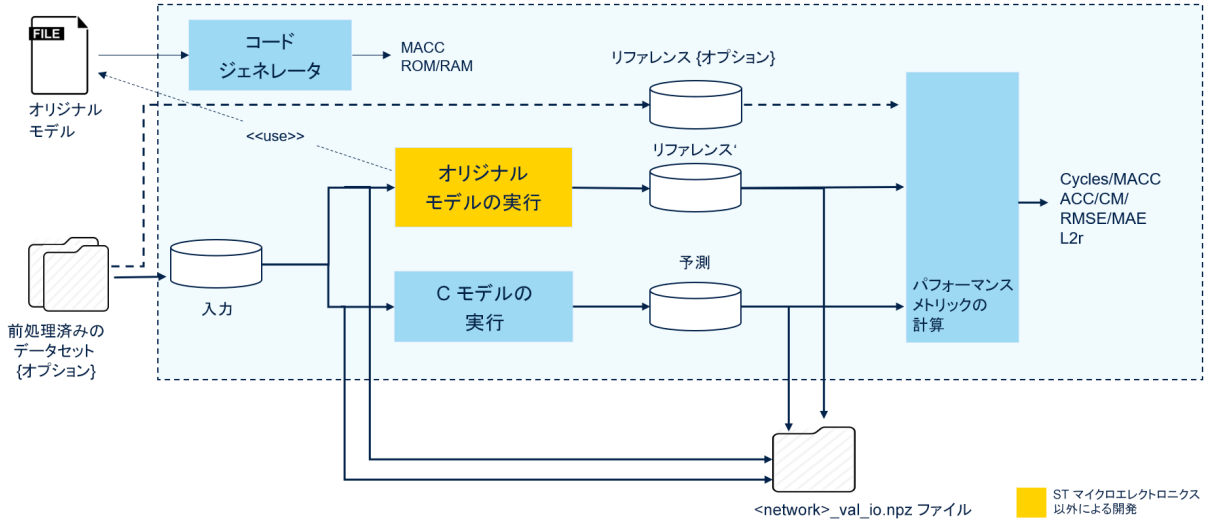
図 49. 最適なアクティベーション/作業バッファ



6.2 検証エンジン

数値の観点から生成されたモデルとアップロードされた DL モデルの精度を比較するための、シンプルで迅速な検証メカニズムが用意されています。両方のモデルには同じ入力テンソル(固定ランダム入力またはカスタム・データセット、[11] を参照)が供給されます。より正確には、[11] で詳しく説明されているように、生成された C モデルを評価するための追加のメトリックが報告されています。X-Cube-AI 拡張パッケージ では、サポートされているすべての DL フレームワークについて、推論 DL を実行するエンジンを提供します。ツールによって検証の失敗が報告されている場合でも、まだ生成された最適な C モデルを考慮することは可能であることに注意してください。パフォーマンスは元の Python™ モデルのようにはならない場合がありますが、C モデルは引き続き使用できます。カスタム・データセットと NN 出力トレースなどを使用した詳細な点検が必要です。

図 50. 検証フローの概要



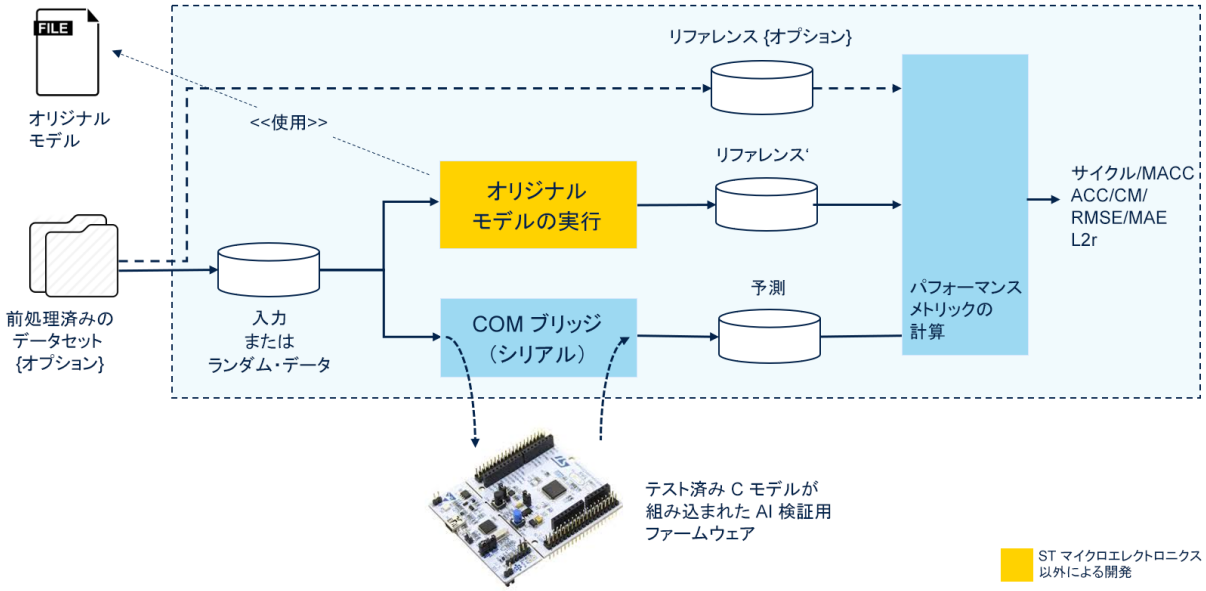
2 つの実行モードがあります。

- デスクトップでの検証：このモードでは、DL モデルを生成された X86 C モデルと比較することができます。ホスト上で実行されます。関連する出力は、セクション 4.6 生成された C モデルの検証 に示されています。
- ターゲットでの検証：このモードでは、DL モデルとターゲット・デバイス上で実行される C モデルを比較します。ホスト・システムと通信するための生成された NN ライブラリと COM エージェントが組み込まれた、特殊な AI テスト・アプリケーションが必要です。出力と使用方法の例を セクション 10 AI 検証アプリケーション に示します。

ターゲットでの検証 機能：

- 接続された STM32 ボードの自動検出
- 生成された組込み C モデルの署名を検証された DL モデルで確認する
- L2 エラーは、最後の出力レイヤでのみ計算される（データ・アップロード時の COM 速度のため）
- レイヤごとの推論実行時間などの追加情報が報告されます（セクション 10 AI 検証アプリケーション を参照）。

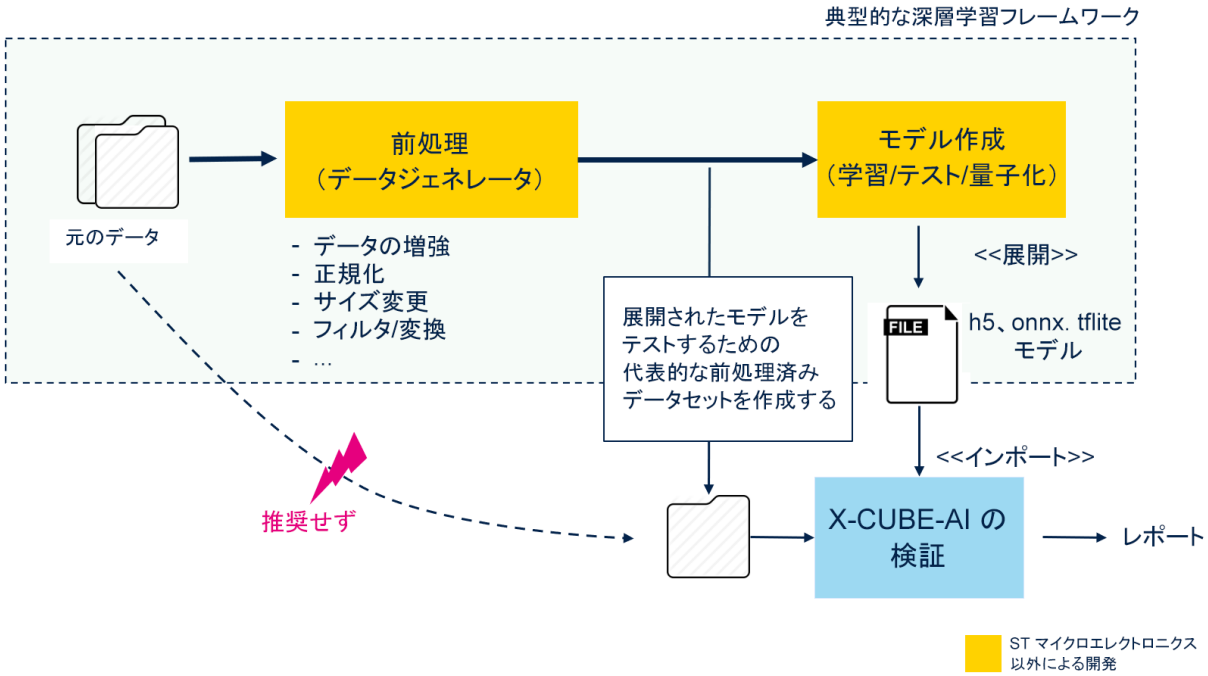
図 51. ターゲットの検証



カスタム·データへの特別な注意

正確な検証プロセスや、より有効なメトリックを得るには、元のモデルと生成された C モデルに、元のモデルのテストに使用された検証データセットに可能な限り近いデータを供給することが重要です。生のデータセットが前処理されている場合、ユーザはこれらの前処理されたデータのサブセットを使用して、代表的なデータセットを構築する必要があります。

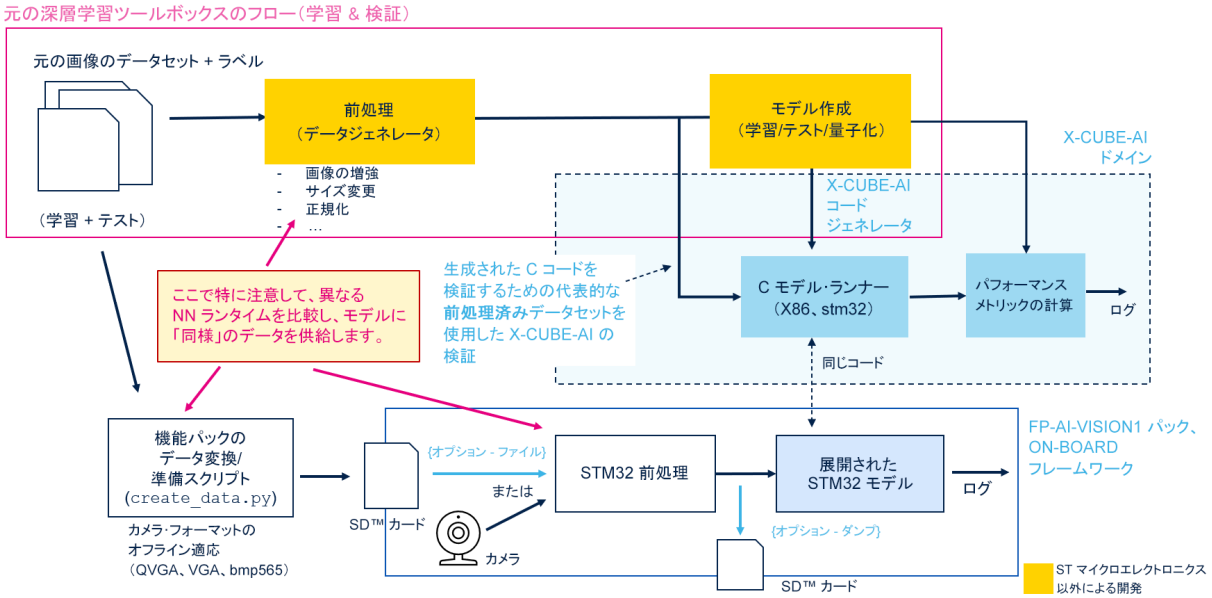
図 52. 代表的なデータセット



この推奨事項は、出力データにも適用されます。メトリックの計算は、提供されたリファレンスと予測値の間の要素ごとの演算に基づいて行われます。たとえば、1 つの分類器（1 つまたは複数の分類）には、ワン·ホット·エンコーディング·データを提供できます。整数エンコードはサポートされていません。

特に注意が必要な典型的な例を 図 53 に示します。コンピュータ·ビジョン（FP-AI-VISION1）用の STM32Cube 機能パック には高度なデバッグ·モードがあり、イメージの注入またはダンプを行うことができます。ユーザが学習済みモデルを X-Cube-AI 検証エンジンで検証して展開されたモデルと比較したい場合は、より正確な検証メトリックを得るために、形式と適用される前処理が同じであることを確認する必要があります。この使用例では、パックによって実行された前処理を考慮するため、一連の代表的な前処理済みイメージを作成して、さまざまなモデルをオフラインでテストしたり、既存の学習済みモデルを改良したりすることが推奨されています。

図 53. コンピュータ·ビジョン用の機能パックの例



7 生成された STM32 NN ライブラリ

インポートした DL モデルごとに、専用（DL モデルによって異なる）の C ファイルのみが生成されます。これらのファイルの名前の先頭には、ユーザが指定したネットワーク名が付きます（セクション 4.3 学習済みの DL モデル・ファイルのアップロード を参照）。これらは、network_runtime.a ライブラリによって実装された内部 API およびプライベートの API に基づいています。

- トポロジ用の <name>.c および <name>.h ファイル
- 重み/バイアス用の <name>_data.c および <name>_data.h ファイル

注 生成された専用データとネットワーク・ファイルは、すべてのツールチェーンおよび STM32 MCU シリーズに共通です。

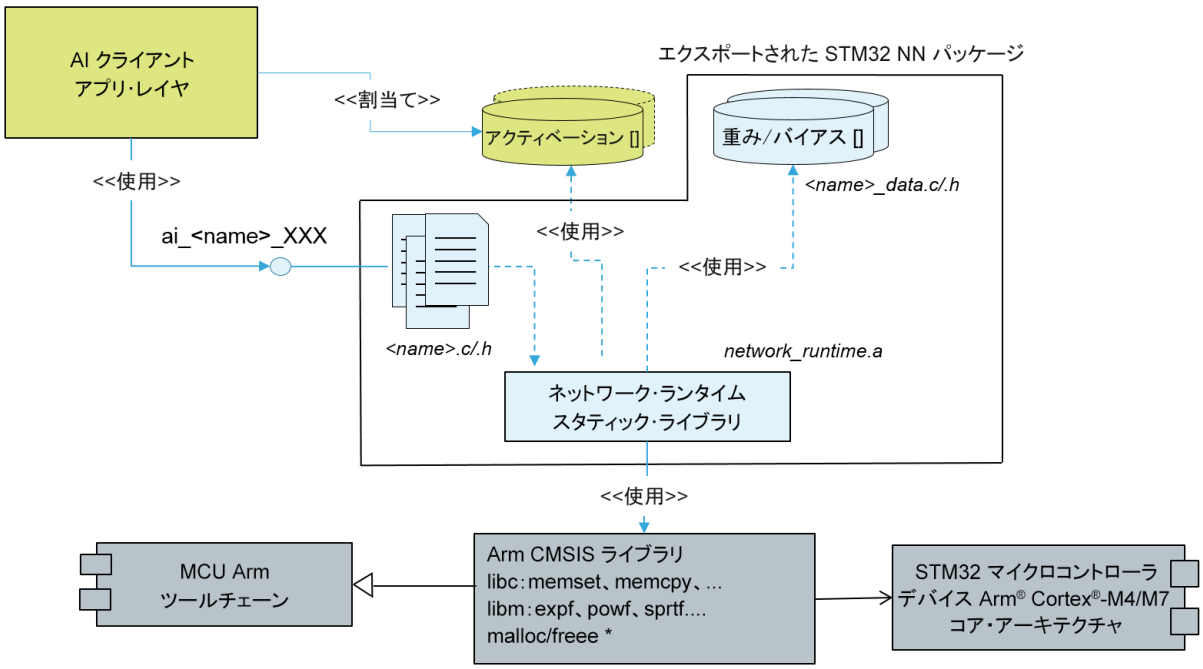
network_runtime.a ライブラリまたは NN 計算カーネル・ライブラリは、スタティック・ライブラリとして提供されます。

- 未使用のシンボルとメソッドはすべてリンク段階で削除されます。
- メモリ・リソースが限られているデバイスには適していないため、ネットワーク-グラフ・インタプリタの手法には基づいていません（ARM-NN のように、Lite 展開環境）。

7.1 ファームウェアの統合

図 54 に生成された STM32 NN パッケージのマイクロコントローラ統合モデルとビュー（ランタイムの依存関係を含む）を示します。

図 54. マイクロコントローラ統合モデルとビュー



アプリケーション層の場合、エクスポートされた NN ライブラリは、「ブラック・ボックス」または自己完結型オブジェクトとみなされます。特化したファイルとネットワーク・トポロジ（<name> .c および <name>.h ファイル）、重み/バイアスパラメータ（<name> _data.c および <name>_data.h ファイル）がソース・ファイルとして提供されています。これらは、共通ネットワーク・ランタイム・ライブラリ（network_runtime.a）に基づいています。システム・ランタイムとの依存関係は、最小限です。

- 標準 libc メモリ操作関数（`memcpy`、`memset`）。通常、これらはマイクロコントローラ・ツールチェーンによって提供されます。
- Cortex®-M の最適化された動作（FPU および DSP 命令）をサポートする CMSIS ライブラリこれは STM32 HAL パッケージの一部です。
- `malloc/free` は、回帰型 層（GRU および LSTM 層）をサポートすることが現在期待されています。将来のリリースでは、代わりに静的なバッファ割り当て手法を使用する予定です。パフォーマンスへの影響は、再帰セル・ユニットの数と関連する処理時間によって軽減されます。

- `expf`、`powf`、`tanhf`、および `sqrtf` 関数をサポートするには、演算ライブラリ(DSP/FPU をサポート)も必要です。
- 最小スタックが要求されます(実際の値は AI システムの性能 アプリケーションで測定可能、セクション 9 AI システムのパフォーマンス·アプリケーションを参照)。

注 外部依存関係はすべて、エンドユーザのリンク段階(ファームウェア·イメージの生成)で解決する必要があります。

アクティベーション·メモリ·バッファは、ヒープ内に動的に割り当てるか、グローバル配列(.bss および .data セクション)として割り当てることができます。`ai_<name>_init()` 関数を参照して、重み/バイアス·バッファとアクティベーション·メモリ·バッファを NN コア·ライブラリに渡す方法を示します。

7.2 ライブラリ·ソース·ツリー·ビュー

IDE プロジェクトが作成されると、ネットワーク·ランタイム·ライブラリはサブフォルダ `<project_name>/Middlewares/ST/AI/` にエクスポートされます。専門または専用の NN ファイルが標準の STM32CubeMX `Inc` および `Src` サブフォルダに格納されます。CMSIS-DSP ライブラリからの要求された特定のファイルも追加されます。

```
```
<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 applica
tion */
    | |- 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
```
```

ファイル名 `network_runtime.a` は、`NetworkRuntime410_CM4_GCC.a` または `NetworkRuntime410_CM4_IAR.a` などのように、使用した X-Cube-AI バージョンと IDE によって異なります。

7.3 マルチネットワーク推論 API

app_x-cube-ai.c および app_x-cube-ai.h ファイルでは、AI クライアント·アプリケーションで使用できる汎用マルチネットワーク推論 API も提供しています。ネイティブの組込み推論クライアント API に非常に近く([10] を参照)、`create()` 関数のみが異なります。基礎となるネットワークのインスタンスを作成するには、ネットワークの C 名文字列を渡す必要があります。このインタフェースは、さまざまな組込みネットワークのアドレスを指定する一般的な方法を得るために、主にアドオンの AI テスト·アプリケーションで使用されます。

```
/* @file - app_x-cube-ai.h/.c - GENERATED CODE by STM32Cube MX */
 ...
const char* ai_mnetwork_find(const char *name, ai_int idx);

ai_error ai_mnetwork_create(const char *name, ai_handle* network, const ai_buffer*
 network_config);

ai_bool ai_mnetwork_get_info(ai_handle network, ai_network_report* report);
ai_error ai_mnetwork_get_error(ai_handle network);
ai_handle ai_mnetwork_destroy(ai_handle network);
ai_bool ai_mnetwork_init(ai_handle network, const ai_network_params* params);
ai_i32 ai_mnetwork_run(ai_handle network, const ai_buffer* input, ai_buffer* outpu
t);
```

7.4 リエントラント（再入可能）およびスレッドの安全性に関する考慮事項

エントリ・ポイントを同時アクセスから保護するための内部同期メカニズムは実装されていません。API がマルチスレッドのコンテキストで使用される場合、インスタンス化された NN の保護は、アプリケーション層自体によって保証されなければなりません。

RAM の使用量を最小限に抑えるため、同じアクティベーションメモリ・チャンク（`SizeSHARED`）を使用して、複数ネットワークをサポートできます。この場合、ユーザは、進行中の推論の実行が別のネットワークの実行によって横取りされないことを保証する必要があります。

```
SizeSHARED = MAX(AI_<name>_DATA_ACTIVATIONS_SIZE) for name = “net1” … “net2”
```

注 リアルタイムの制約または遅延の理由として横取りが予想される場合、各ネットワーク・インスタンスには独自のプライベートなアクティベーション・バッファが必要です。

7.5 コードとデータの配置に関する考慮事項

現在の STM32 メモリ・アーキテクチャ（STM32L4/STM32F4/STM32F3 ベースおよび STM32F7/STM32H7 ベース）では、パフォーマンス上の理由から、特定のデータまたはコードの配置は見込まれません。Flash ART ペリフェラルと Arm® コア・サブシステム・キャッシュ（Cortex®- M7 ベースのアーキテクチャ）は、メモリ遅延の副作用を効率的に制限します。NN コード（.text セクション）および RO データ（.rodata セクション）は、内部 Flash メモリ領域に配置できます。RW データ（.data および .bss セクション）は、内蔵 SRAM に配置する必要があります。クライアント・スタックが使用されている場合は、0 ウェイトステート・メモリに配置する必要があります。

注 アクティベーション・バッファにメモリ保持要件はありません。これは、実際にはスクラッチ・バッファまたは作業バッファとみなすことができます。2 つの推論の間で、たとえばバッファを前処理の目的などで再利用したり、システムがディープ・スリープになったときに関連するメモリ・デバイスをオフにしたりすることができます。

7.6 デバッグに関する考慮事項

ライブラリは、バイナリ形式の最適化されたブラック・ボックスとみなす必要があります（ソース・ファイルは供給されない）。ランタイムの内部データまたは状態のイントロスペクションはサポートされません。NN のマッピングと移植は、X-Cube-AI ジェネレータによって保証されています。統合の一部の問題は、`ai_<name>_get_error()` 関数によってハイライトされます。

8 組込み推論クライアント API

生成された NN コードを使用するために、単純な組込み推論クライアント API が生成されます（図 54 の `ai_<name>_XX()` の機能を参照）。これは、`<project_name>/Src/<name>.h` ファイルの一部です。すべての関数とマクロは、指定された C ネットワーク名に従って生成されます。使用方法と詳細な説明については、[10] を参照してください。

9 AI システムのパフォーマンス·アプリケーション

AI システムのパフォーマンス·アプリケーションは、セルフおよびベアメタルのオンデバイス·アプリケーションで、生成された NN の重要なシステム·インテグレーションの側面について、出荷時状態で測定することができます。精度パフォーマンスの側面について、ここで考慮することはできません。報告される測定値は次のとおりです。

- 推論による CPU サイクル(ms 単位の時間、CPU サイクル、CPU の負荷)
- 使用スタックおよび使用ヒープ(バイト単位)

アプリケーションを実行するために、次の手順を順番に実行します。

1. COM ポートを介して接続されたホスト·シリアル·ターミナル·コンソールを開いて設定します(ST-LINK/V2 機能など、通常は USB 接続による仮想 COM ポートでサポートされます)。
2. COM 設定をセットします。STM32 USART の設定に合わせる必要があります(セクション 3.2 ハードウェアおよびソフトウェアのプラットフォーム設定 を参照)。
   - 115200 ボー
   - 8 ビット
   - 1 個のストップビット
   - パリティなし
3. アプリケーションを起動するためにボードをリセットします。

アプリケーションの実行中に、p または P をコンソールに入力すると、メイン·ループを中断できます。アプリケーションには最低限のインタラクティブ·コンソールが組み込まれており、次のコマンドに対応しています。

```
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 システム·ランタイム情報

図 55 および 図 56 に、ログの最初の部分が表示されます。それぞれ Keil® および Atollic IDE の STM32 ランタイムまたは実行環境の有用な情報を示します。これには、デバイス ID、システム·クロック値、使用しているツールチェーンなどが含まれます。

図 55. システム·ランタイム情報 - Keil® IDE

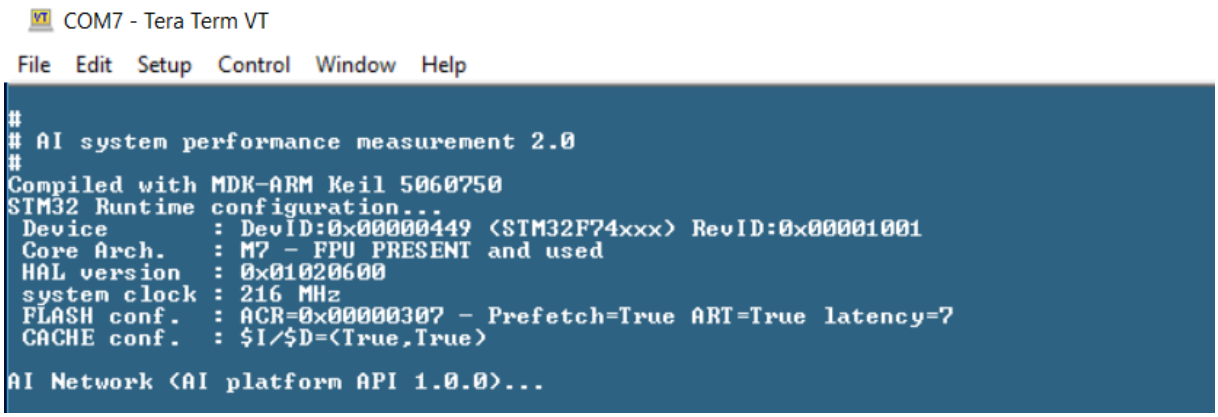
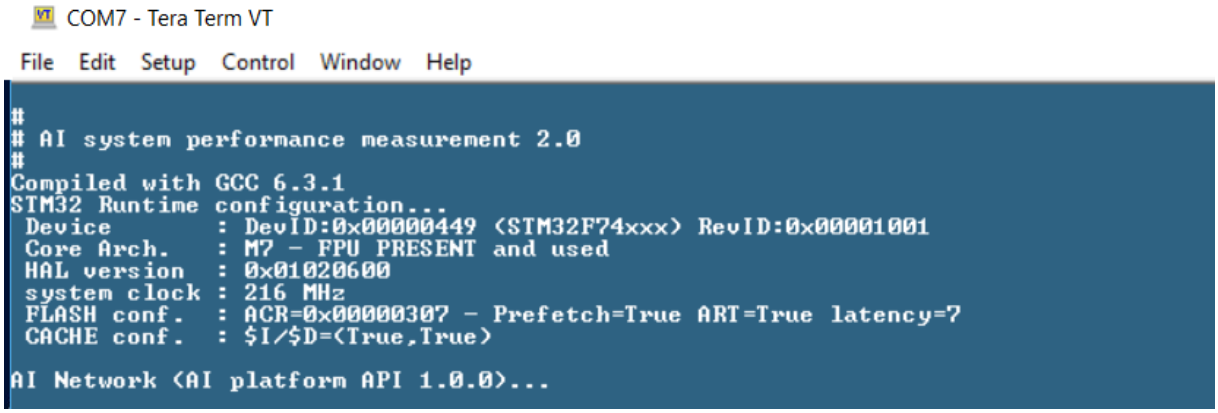


図 56. システム·ランタイム情報 - AtollicIDE

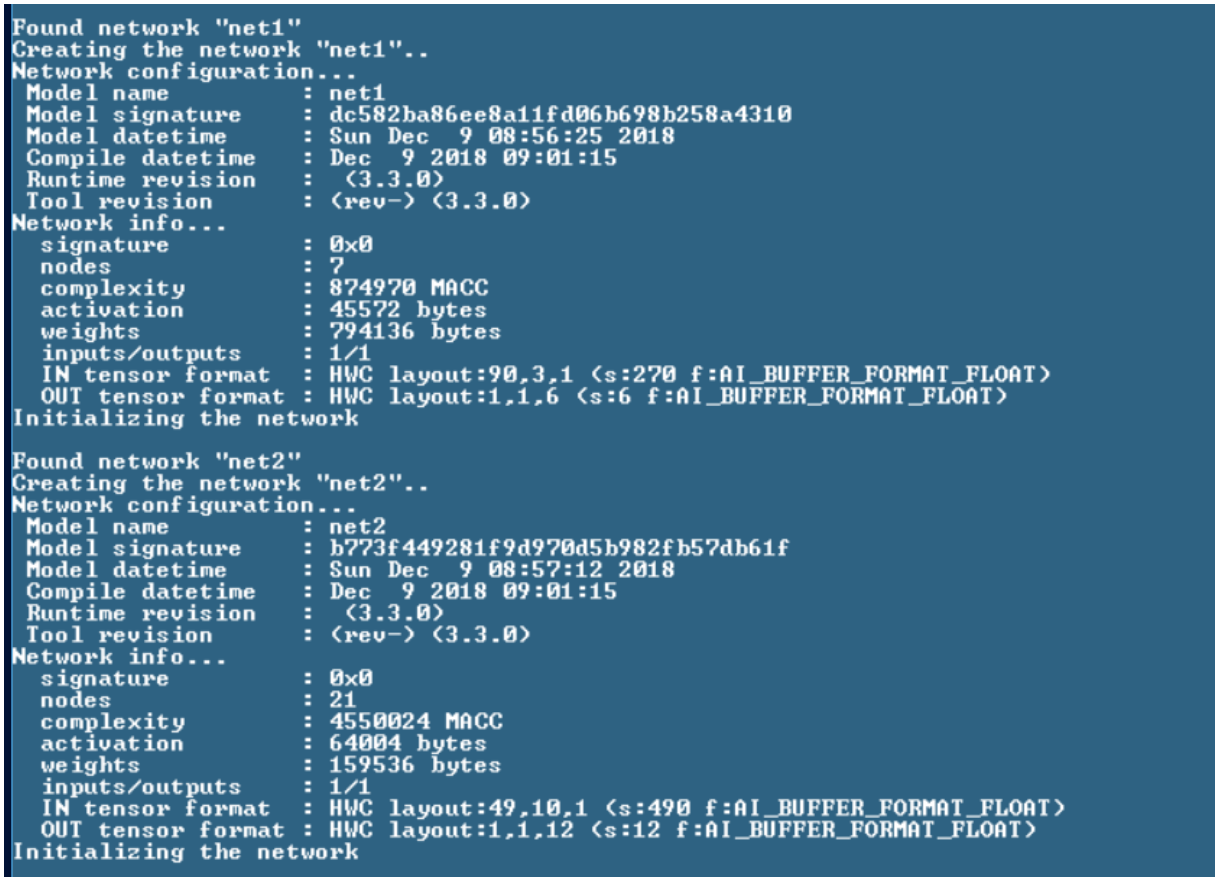


注　これらの情報をログから取得するには、メイン·ループの実行中にコンソールに r または R と入力します。

9.2 組込み C モデル·ネットワーク情報

図 57 の後半部分には、生成された NN の主なスタティック特性を示します。特に、RAM/Flash サイズ（アクティベーション/重みフィールド、それぞれバイト単位）と論理計算量（MACC、計算量フィールド）が提供されています。入力および出力テンソルの形状定義も報告されます。これらの情報は、ai_<name>_get_info() クライアント API 関数によってクライアント·アプリケーション·コードから入手することもできます。

図 57. C モデル·ネットワーク情報



注　これらの情報をログから取得するには、メイン·ループの実行中にコンソールに r または R と入力します。

9.3 組込み C モデルのランタイム・パフォーマンス

図 58 および 図 59 に示すように、ログの最後の部分（メイン・ループ）には、出荷時のシステム・パフォーマンスの測定値が報告されます。推論によって CPU サイクル数を測定するために、ランダム入力がネットワークに供給されます（CPU cycles）。CPU workload および cycles/MACC は、この値から推測されます。測定中、IRQ はマスクされます。

- duration は、1 回の推論の時間を ms 単位で示します。
- CPU cycles は、1 回の推論あたりの CPU サイクル数を示します。
- CPU workload インジケータは、1 秒間の関連する CPU の負荷に対応します。
- cycles/MACC は MACC 操作による CPU サイクル数です。

図 58. C モデルのランタイム・パフォーマンス

```
Running PerfTest on "net1" with random inputs (16 iterations)...
---------------

Results for "net1", 16 inferences @216MHz/216MHz (complexity: 874970 MACC)
 duration      : 28.047 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

Running PerfTest on "net2" with random inputs (16 iterations)...
---------------

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
```

図 59. ヒープおよびスタック・チェック機能付きの C モデルのランタイム・パフォーマンス

```
Running PerfTest on "network" with random inputs (16 iterations)...
---------------

Results for "network", 16 inferences @80MHz/80MHz (complexity: 3729752 MACC)
 duration      : 569.124 ms (average)
 CPU cycles    : 45529988 -134210/+211019 (average,-/+)
 CPU Workload : 56%
 cycles/MACC   : 12 (average for all layers)
 used stack    : 284 bytes
 used heap     : 16:29568 16:29568 (req:allocated,req:released) cfg=3
```

注　「used heap」は、すべての推論の実行中にリクエストされた malloc() の数と累積割り当てサイズ（それぞれ free()）を示します。仮想メモリ・リークを検出するために、カウンタは 2 つの推論の間またはテストの繰り返しの間ではリセットされません。この場合、最小ヒープ・サイズは 29568 / #iter = ~2Kbytes です。

注意　現在、ヒープ監視は GCC ベースの環境でのみサポートされています。

10 AI 検証アプリケーション

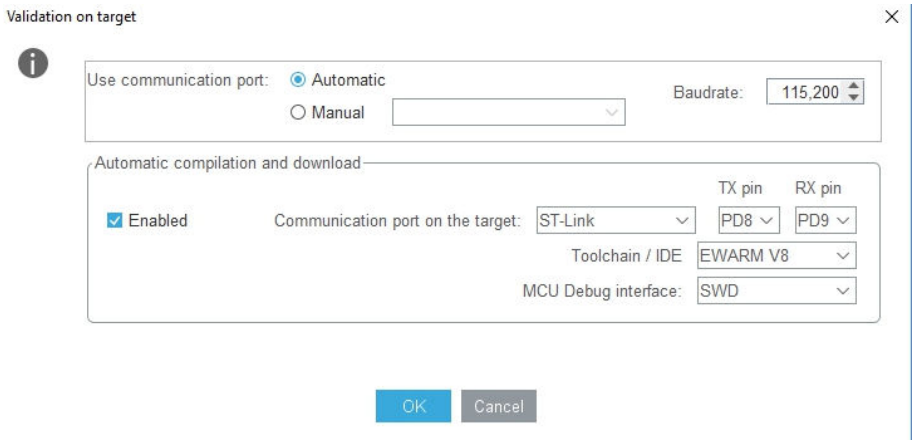
AI 検証 アプリケーションは、セクション 6.2 検証エンジン に示すように、デバイスでの検証 をサポートするセルフおよびベアメタルのオンデバイス·アプリケーションです。推論 API をエクスポートするために、ホストとの USART ベースのインタフェースを提供します。

AI 検証アプリケーション全体は、通常の IDE プロジェクトとして生成およびプログラムされるか、自動的にコンパイルされ、STM32CubeMX 自体からダウンロードされます（図 60 を参照）。

このアプリケーションが自動的に生成されない場合、以下の手順に従います。

1. ボードをプログラムした場合は、リセットまたは再起動します。
2. ファームウェアの生成に使用された `ioc` ファイルを再度開きます（閉じられていた場合）。
3. AI 設定パネルに移動し、<network_name> タブを開いて必ず検証を行います。
4. [Validation on target] ボタンをクリックして、検証プロセスを開始します。[OK] ボタンをクリックする前に、ユーザは 図 60 に示すように使用されるホストの COM ポートを表示することができます。そうしないと、使用可能なすべての COM ポートが検出され、接続された有効な STM32 ボードを検出します（最初に検出されたボードが使用されます）。

図 60. デバイスでの検証用のホスト COM ポート·セレクタ



デスクトップでの検証 の場合、最終結果は [Validation status] フィールドに報告されます。より詳細な情報は、UI ログ·コンソールに報告されます。

10.1 システム·ランタイム情報

図 61 に示す報告されたログの最初の部分には、デバイス ID、クロック周波数、メモリ·サブシステム設定、組込みネットワークのリストなどのメイン·システム情報が示されています。検証済みのネットワークには、shape がある場合とない場合のテンソルの説明と、使用している AI ツールのバージョンが提供されています。

図 61. システム·ランタイム情報

MCUs Selection | Output

```
ON-DEVICE STM32 execution ("net1", default, 115200)..

<Stm32com id=0x1bfd54ec0f0 - CONNECTED(COM7/115200) devid=0x449/STM32F74xxx msg=1.0>
 0x449/STM32F74xxx @72MHz/72MHz (FPU is present) lat=2 Core:I$/D$ ART: PRFTen ARTen
 found network(s): ['net1', 'net2']
 description    : 'net1' (90, 3, 1)-[7]->(1, 1, 6) macc=874970 rom=775.52KiB ram=44.50KiB
 tools versions : rt=(3, 3, 0) tool=(3, 3, 0)/(1, 1, 0) api=(1, 0, 0) "Sat Dec  8 23:55:41 2018"

Running with inputs=(10, 90, 3, 1)..
....... 1/10
....... 2/10
```

10.2 組込み C モデルのランタイム・パフォーマンス

図 62 に示すログの後半部分には、出荷時のシステム・パフォーマンスの測定値（推論による平均実行時間 duration）が報告されます。cycles/MACC は duration の値から差し引かれます。測定中、IRQ はマスクされます。

- duration は、1 回の推論の時間を ms 単位で示します。
- CPU cycles は、1 回の推論あたりの CPU サイクル数を示します。
- cycles/MACC は MACC 操作による CPU サイクル数です。

図 62. C モデルのランタイム・パフォーマンス

```
....... 9/10
....... 10/10
 RUN Stats    : batches=10 dur=24.266s tfx=21.957s 0.491KiB/s (wb=10.547KiB,rb=240B)

Results for 10 inference(s) @72/72MHz (macc:874970)
 duration    : 78.846 ms (average)
 CPU cycles  : 5676924 (average)
 cycles/MACC : 6.49 (average for all layers)
```

10.3 レイヤごとのランタイム・パフォーマンス

図 63 に示すログの次の部分には、生成された C モデルに関する追加情報が提供されています。これには、実装されている C レイヤの名前とタイプ（Clayer / id / desc）、出力波形（oshape）、および推論による平均実行時間（ms）が含まれます。

図 63. レイヤごとの結果 - ターゲットの検証

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

Clayer  id  desc                            oshape            ms
-------------------------------------------------------------------------------
0       0   10011/(Merged Conv2d / Pool)    (10, 44, 1, 128)  24.277
1       4   10005/(Dense)                   (10, 1, 1, 128)   53.165
2       4   10009/(Nonlinearity)            (10, 1, 1, 128)   0.010
3       5   10005/(Dense)                   (10, 1, 1, 128)   1.303
4       5   10009/(Nonlinearity)            (10, 1, 1, 128)   0.010
5       6   10005/(Dense)                   (10, 1, 1, 6)     0.066
6       6   10014/(Softmax)                 (10, 1, 1, 6)     0.015
                                                              78.846 (total)
```

10.4 ターゲットの検証の最終結果

図 64 に示すログの最後の部分には、検証プロセスの最終結果が提供されています。デスクトップでの検証結果と似ていますが、最終レイヤの L2 エラーのみが報告されます（セクション 6.2 検証エンジン を参照）。

図 64. ターゲットの検証の最終報告

```
  MACC / frame: 874970
  ROM size:     775.52 KBytes
  RAM size:     44.50 KBytes (Minimum: 44.50 KBytes)


Matching criteria: L2 error < 0.01 on the output tensor

  Ref layer 6 matched with C layer 6, error: 0.0010825497

Validation: OK
```

10.5 接続中に返されたエラー

デフォルトでは、次の USART 設定が使用されます。

- 8 ビット
- 1 個のストップビット
- パリティなし
- 115200 ボー

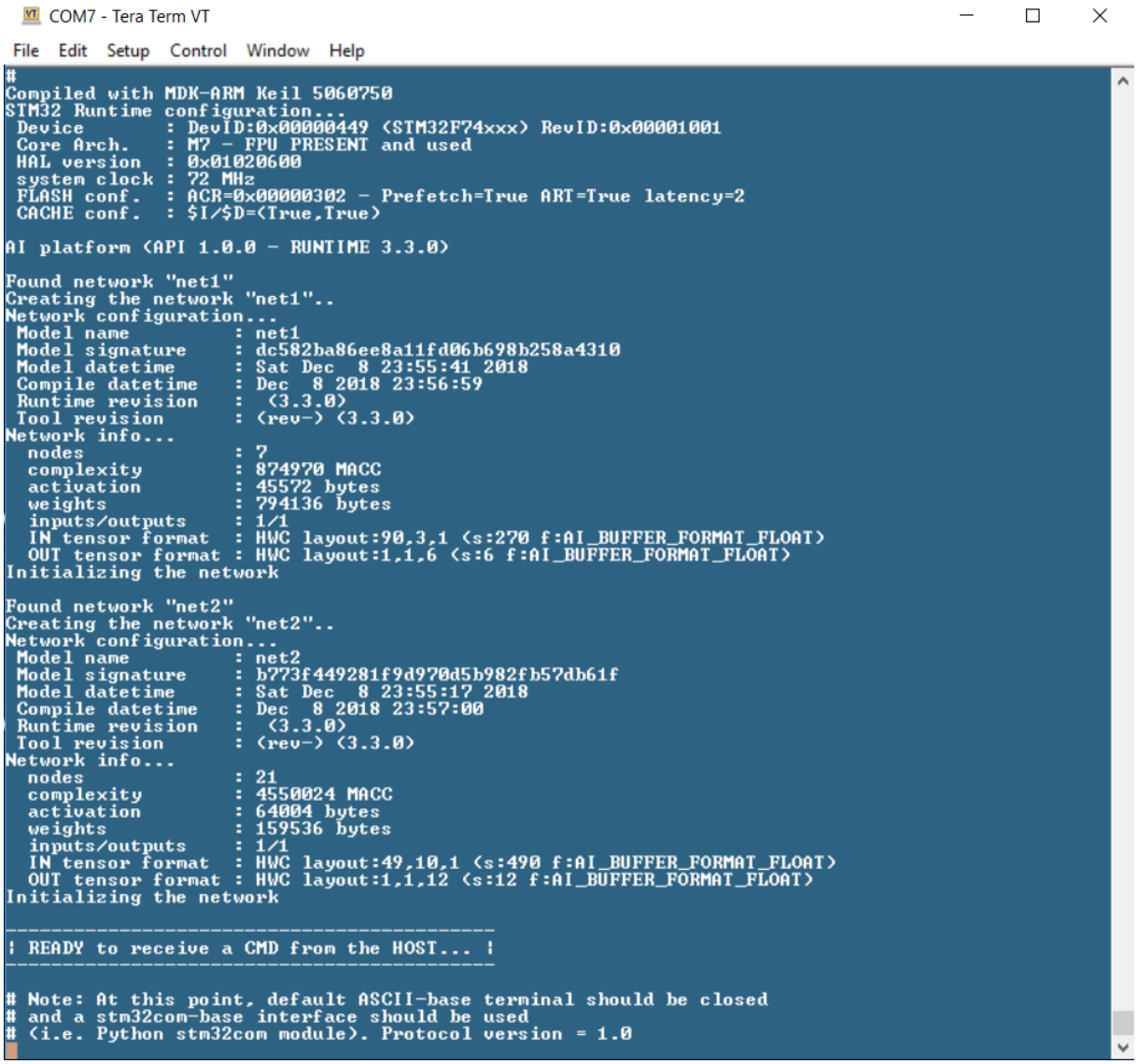
ユーザが再定義した場合は、設定全体でボー・レートの整合性を取る必要があります（セクション 3.2 ハードウェアおよびソフトウェアのプラットフォーム設定 および 図 60. デバイスでの検証用のホスト COM ポート・セレクタ を参照）。

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

ボードが接続されていないか、ボードが見つからないか、またはファームウェアが期待された AI Validation ファームウェアではないことを示します。このエラーは、不整合なファームウェア状態を示している場合もあり、その場合はボードを再起動する必要があります。

ファームウェアが正しくプログラムされたことを確認するには、ブート時に ASCII ベースのログを生成するホスト・シリアル・ターミナル・コンソールを開きます。Validation on target プロセスを再度起動する前に、必ず接続を閉じてください。

図 65. AI 有効 - 初期ログ



10.5.2 Error: network_name is not a valid network

名前で識別された予測された C モデルが、接続されたボードでは使用できないことを示します。UI ログ·コンソール（Outputs ウィンドウ）を参照してください。

10.5.3 Error: the embedded STM32 model does not match the C model

生成された C モデルの署名が期待されたモデルと一致しないことを示します。署名の確認に使用されるパラメータは、次のとおりです。

- RAM/ROM サイズ
- MACC
- ノード数
- ツールのバージョン

UI ログ·コンソール（Outputs ウィンドウ）を参照してください。

11 AI テンプレート・アプリケーション

選択された場合、生成された IDE プロジェクトは、生成された C モデル使用するための基本的な AI アプリケーションの例を含む完全な AI テンプレート・アプリケーションとは言えません。aiSystemPerformance.h および aiSystemPerformance.c ファイルがこの目的の良い例です。特定の生成ファイルのみが生成されます。これは、2 つの単純なエントリ・ポイント（`init` および `process` 関数）で最初のベア・メタル・アプリケーションを開発する起点として使用できます。詳細については、[10]を参照してください。

12 サポートされている 深層学習 のツールボックスおよびレイヤ

X-Cube-AI コアは、現在次の DL ツールボックスをサポートしています。

- Keras：//keras.io/
- TensorFlow™ Lite：www.tensorflow.org/lite/
- ONNX：//onnx.ai/

注 TensorFlow は Google 社の商標です。

ネットワーク C API の表現力と、特定のツールボックスのパーサーに応じて、ツールボックスごとに、考えられるすべてのレイヤとレイヤ・パラメータのサブセットのみがサポートされます。サポートされる設定の詳細については、[7]、[8]、および [9] を参照してください。

13 エラー処理

X-Cube-AI コアは、さまざまなエラーを処理して、ユーザに報告します（[6] を参照）。

14 よくある質問

14.1 デバッグ目的でログ·ファイルを取得しますか?

ターゲットの検証 プロセスが実行されると、ターゲットとやりとりしたすべてのメッセージ(データを含む)が専用のログ·ファイルに格納されます。

`C:\Users\<username>\.stm32cubemx\ai_stm32_msg.log`

検証または生成プロセスが失敗した場合、追加のデバッグ/ログ情報を次のファイルで確認できます。

`C:\Users\<username>\.stm32cubemx\STM32CubeMX.log`

14.2 arm_dot_prod_f32.c ファイルをコンパイルできません。

IDE プロジェクト·ファイルを再生成すると、`arm_dot_prod_f32.c` のコンパイルに失敗することがあります。

```
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_CM
4, 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
```

対象の STM32 デバイスによっては、プロジェクト設定で次の C 定義を再定義する必要があります。

```
ARM_MATH_CM7,__FPU_PRESENT=1
```

14.3 使用ヒープまたは使用スタック: 無効または未対応

このログは、使用されているツールチェーンで、スタック(またはヒープ)監視が明示的に無効化されているか、実装されていないことを示します。

表 4. ヒープとスタックの監視のサポート

ツールチェーン	スタック監視	ヒープ監視	注
GCC	サポートあり	サポートあり	STM32CubeIDE
IAR Systems EWARM 8.x	サポートあり	実装なし	-
Keil® MDK-ARM	実装なし	実装なし	-

ヒープとスタックの監視を有効化する例:

```
/* @file: aiSystemPerformance.c */
...
#if defined(__GNUC__)
#define _APP_STACK_MONITOR_ 1
#define _APP_HEAP_MONITOR_ 1
#elif defined (__ICCARM__)
#define _APP_STACK_MONITOR_ 1
#define _APP_HEAP_MONITOR_ 0
#else
#define _APP_STACK_MONITOR_ 0
#define _APP_HEAP_MONITOR_ 0
#endif
...
```

14.4 使用ヒープ が常にゼロである理由

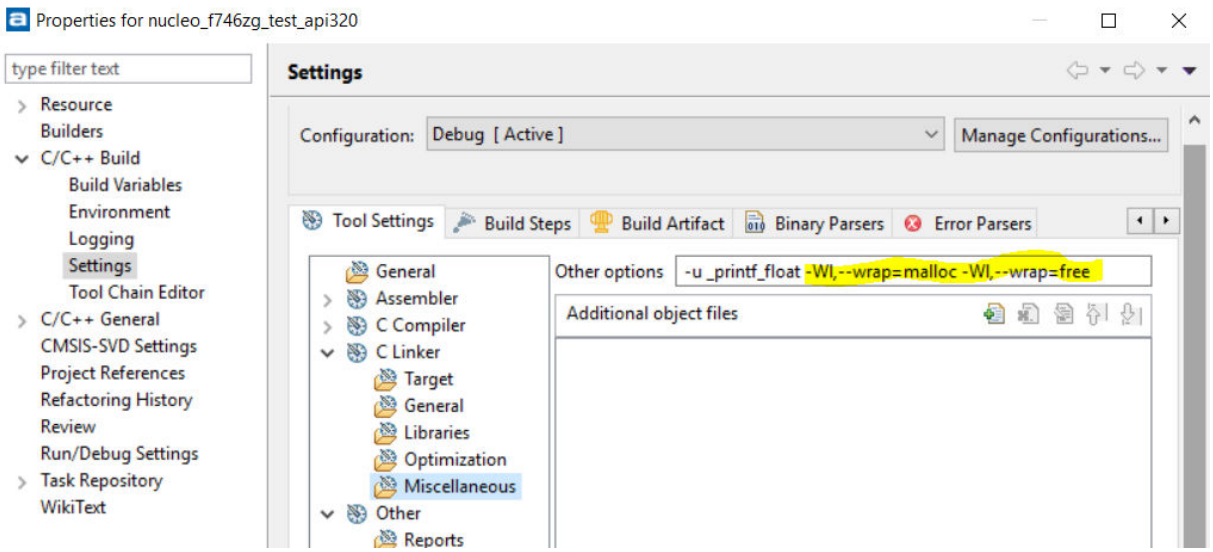
GCC ベースのプロジェクトの場合のみ:

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

このような結果は、必ずしも問題ではありません。ほとんどの network_runtime.a ライブラリは、事前に割り当てられた R/W バッファ構成（アクティベーション・バッファ）に基づいています。一部の特定の層（回帰型層）について、現在の実装は `malloc()` 関数を通してこれらの作業バッファの一部を動的に割り当てるよう要求します。

ヒープ監視は、ツールチェーン固有のメカニズムに基づいており、システムの `malloc()` および `free()` 関数の折り返しが可能です。この折り返しを有効にするには、図 66 に示すように、`-Wl,--wrap=malloc -Wl,--wrap=free` のその他のリンカ・オプションをビルド・システムに設定する必要があります。

図 66. ヒープ監視を有効にするためのリンカ・オプション



14.5 フォーマットされた浮動小数点数が GCC ベースのプロジェクトで空の場合

フォーマットされた浮動小数点数を出力するには、次のリンク・オプションを追加する必要があります。

```
-u _printf_float
```

14.6 CPU サイクル/MACC とは?

セクション 4.5.1 CPU サイクル/MACC とは? および セクション 9 AI システムのパフォーマンス・アプリケーションを参照してください。

14.7 TIMER ペリフェラルを有効にしたり設定したりする必要はありますか?

必要ありません。推論で CPU サイクル数を測定するメカニズムでは、専用の Arm® Cortex®-M デバッグ・ユニット（DWT：Data Watch-point and Trace（データ・ウォッチポイントおよびトレース）ユニット）を使用しています。これはサポートされているすべての STM32 デバイスで利用可能です。このメカニズムでは、CPU クロック（HCLK システム・クロック）によってクロック供給された、フリーランニング・カウンタを使用しています。

14.8 生成したプロジェクトでエクスポートされた NN ライブラリのみを更新する方法

エクスポートおよび生成されたファイルへの変更に STM32CubeMX の設計ガイドラインを適用すると、簡単に更新できます（`/* USER CODE BEGIN*/, /*... END*/` タグ）。`<project_name>.ioc` ファイルを直接再度開いて、新しい NN モデルをアップロードし、IDE プロジェクトを更新することができます。

14.9 非 STM32CubeMX ベースのプロジェクト用に NN ライブラリをエクスポートできますか?

エクスポートされた NN ライブラリが明確に定義された自己完結型サブフォルダにあるため（セクション 7 生成された STM32 NN ライブラリ を参照）、この AI サブフォルダは転送先プロジェクトのソース・ツリーに完全にコピーすることができます。

1. ユーザの STM32 MCU デバイス用に新しいダミー STM32CubeMX プロジェクトを作成します。

2. ユーザのツールチェーン/IDE の IDE プロジェクトを生成します。この手順は、ツールチェーンおよび Arm®-Cortex®-M-アーキテクチャに依存した、正しい `network_runtime.a` ライブラリを含めるために要求されます。
3. 新しいプロジェクトのソース・ツリーに、生成された AI サブフォルダをコピーします。
4. 構築されたシステムに `network.c` および `network_data.c` ファイル、および `network_runtime.a` ライブラリを追加し、セクション 7 生成された STM32 NN ライブラリ に従って、C/C++ コンパイラおよびリンカのオプションを更新します。
5. 手順 1 に戻って、修正された NN モデルを更新して評価します。

14.10 コマンドライン・インタフェースとは?

完全なコマンドライン・インタフェースを、X-Cube-AI 拡張パッケージ で提供しています（[6] を参照）。

15 リファレンスとドキュメント

15.1 このユーザ·マニュアルで使用されているリファレンス

表 5. 参照

ID	説明	Link
[1]	NUCLEO-F746ZG 開発キット	www.st.com/en/product/nucle-f746zg
[2]	STM32CubeIDE	www.st.com/stm32cubeide
[3]	IAR Embedded Workbench® IDE - ARM v8.x または v7.x	www.iar.com/iar-embedded-workbench
[4]	µVision® V5.25.2.0 - Keil® MDK-ARM プロフェッショナル版	www.keil.com
[5]	STM32CubeMX - 初期化コード·ジェネレータ	www.st.com/en/product/stm32cubemx
[6]	X-Cube-AI `stm32ai` コマンドライン·インタフェース	`Documentation/command_line_interface.html` の X-Cube-AI 拡張パッケージ に付属のドキュメント(1)
[7]	X-Cube-AI Keras ツールボックスのサポート	`Documentation/supported_ops_keras.html` の X-Cube-AI 拡張パッケージ に付属のドキュメント(1)
[8]	X-Cube-AI ONNX ツールボックスのサポート	`Documentation/supported_ops_onnx.html` の X-Cube-AI 拡張パッケージ に付属のドキュメント(1)
[9]	X-Cube-AI TensorFlow™ Lite ツールボックスのサポート	`Documentation/supported_ops_tflite.html` の X-Cube-AI 拡張パッケージ に付属のドキュメント(1)
[10]	X-Cube-AI 組込み推論クライアント API	`Documentation/embedded_client_api.html` の X-Cube-AI 拡張パッケージ に付属のドキュメント(1)
[11]	X-Cube-AI 評価レポートとメトリック	`Documentation/evaluation_metrics.html` の X-Cube-AI 拡張パッケージ に付属のドキュメント(1)
[12]	X-Cube-AI 量子化モデルと量子化コマンド	`Documentation/quantization.html` の X-Cube-AI 拡張パッケージ に付属のドキュメント(1)

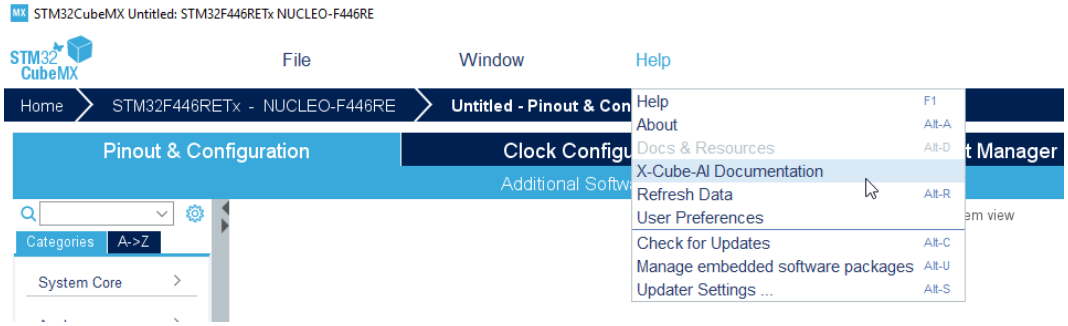
1. セクション 15.2 X-Cube-AI 同梱マニュアルへのアクセスを参照。

15.2 X-Cube-AI 同梱マニュアルへのアクセス

以下に提案された 2 つの解決策のうちの 1 つに従って、利用可能な X-Cube-AI 拡張パッケージ のドキュメントにアクセスします。

1. メニューから直接アクセスする場合:
   a. 図 67 に示すように、[Help]>[X-CUBE-AI Documentation] をクリックします。

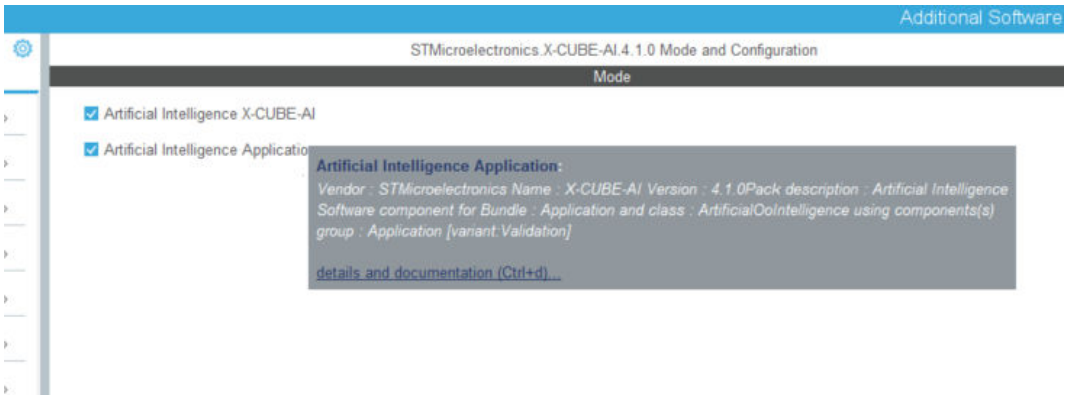
図 67. パッケージ付属のドキュメントに直接メニューからアクセスする



2. X-Cube-AI モードからアクセスする場合：

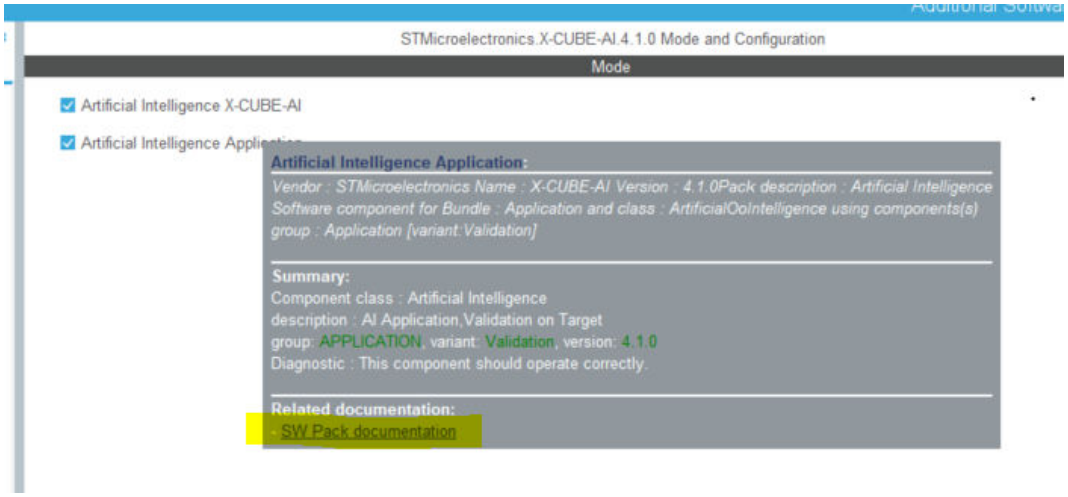
   a. 図 68 に示すように、X-Cube-AI モードのいずれかにカーソルを合わせ、[details and documentation] をクリックします。

図 68. パッケージ付属のドキュメントに X-Cube-AI モードでアクセスする（1/2）



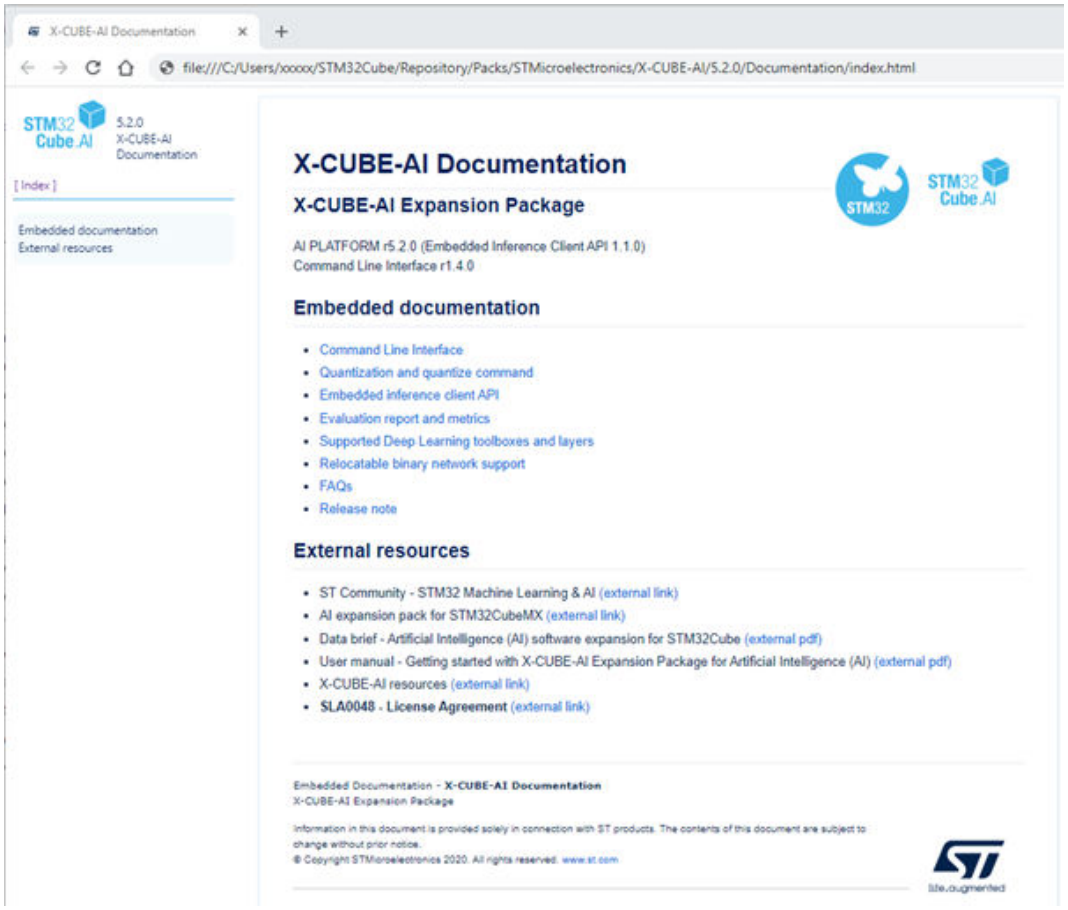
   b. 図 69 に示すように、[SW Pack documentation] をクリックします。

図 69. パッケージ付属のドキュメントに X-Cube-AI モードでアクセスする（2/2）



   c. 図 70 に示すように、利用可能な X-Cube-AI ドキュメントが一覧表示されたブラウザ·ウィンドウが開きます。

図 70. パッケージ付属のドキュメントの索引



改版履歴

表 6. 文書改版履歴

日付	版	変更内容
2019 年 1 月 15 日	1	初版発行
2019 年 7 月 19 日	2	X-CUBE-AI 4.0.0 用に更新: • 量子化を追加 • コマンドライン・インタフェースを追加 • TensorFlow™ Lite のサポートを追加 次の拡張パッケージのドキュメントを参照するユーザ・マニュアルを簡略化: • セクション 8 組込み推論クライアント API • セクション 11 AI テンプレート・アプリケーション • セクション 12 深層学習でサポートされているツールボックスとレイヤ • セクション 13 エラー処理
2019 年 10 月 11 日	3	X-CUBE-AI 4.1.0 用に更新: • TensorFlow™ Lite の量子化モデルのサポートを追加 • 検証用の外部メモリのサポートを追加 ユーザ・マニュアルを簡略化: • セクション 14.2 カスタム・データ・セットのファイル・フォーマット? を削除 • パッケージ内のドキュメントへのアクセス方法を説明した セクション 15.2 を追加
2020 年 1 月 6 日	4	X-CUBE-AI 5.0.0 用に更新: • ONNX の量子化モデルのサポートを追加 • 入力バッファのアクティベーション・バッファの用途を追加 • パッケージ内のドキュメントへのダイレクト・メニュー・アクセスを追加
2020 年 6 月 8 日	5	X-CUBE-AI 5.1.0 用に更新: • スクリーンショットを更新 • セクション 4.3 学習済みの DL モデル・ファイルのアップロード にスプリットウェイト・オプションを追加 • 次のセクションにメモリ使用量のグラフとレイヤ情報を追加: – セクション 4.4.2 生成された C モデルのグラフ表現 • 次のセクションのツールボックスを更新: – セクション 1.2 X-CUBE-AI が STM32Cube を補完する方法 – セクション 14.4 使用ヒープまたは使用スタック:無効または未対応
2020 年 10 月 1 日	6	X-CUBE-AI 5.2.0 用に更新: • スクリーンショットを更新 • Cortex®-M33 コア・ベースの STM32L5 シリーズのサポート • セクション 4.3 学習済みの DL モデル・ファイルのアップロード の外部メモリの詳細設定を更新
2021 年 3 月 4 日	7	X-CUBE-AI 6.0.0 用に更新: • スクリーンショットを更新 • STM32WL シリーズの互換性を追加 • セクション 4.3 学習済みの DL モデル・ファイルのアップロード を更新 • セクション 4.4 詳細設定 を作成 • セクション 6.2 検証エンジン を更新し、カスタム・データへの特別な注意 を追加 • セクション 15.1 このユーザ・マニュアルで使用されているリファレンス を更新 • セクション 14.2 マルチネットワークの制限事項 を削除
2022 年 1 月 4 日	8	X-CUBE-AI 7.1.0 用に更新: • STM32F0 シリーズ、STM32G0 シリーズ、STM32L0 シリーズ、および STM32U5 シリーズを追加 • セクション 1.6 前提条件を更新。 • メモリ設定/拡張オプション用の セクション 4.4 詳細設定 を更新

目次

表一覧

図一覧

重要なお知らせ（よくお読み下さい）

STMicroelectronics NV およびその子会社（以下、ST）は、ST 製品及び本書の内容をいつでも予告なく変更、修正、改善、改定及び改良する権利を留保します。購入される方は、発注前に ST 製品に関する最新の関連情報を必ず入手してください。ST 製品は、注文請書発行時点で有効な ST の販売条件に従って販売されます。

ST 製品の選択並びに使用については購入される方が全ての責任を負うものとします。購入される方の製品上の操作や設計に関して ST は一切の責任を負いません。

明示又は黙示を問わず、ST は本書においていかなる知的財産権の実施権も許諾致しません。

本書で説明されている情報とは異なる条件で ST 製品が再販された場合、その製品について ST が与えたいかなる保証も無効となります。

ST および ST ロゴは STMicroelectronics の商標です。ST の登録商標については ST ウェブサイトをご覧ください。www.st.com/trademarks.その他の製品またはサービスの名称は、それぞれの所有者に帰属します。

本書の情報は本書の以前のバージョンで提供された全ての情報に優先し、これに代わるものです。