人工智能（AI）X-CUBE-AI 扩展包入门指南

引言

本用户手册指导了基于 IDE 逐步构建用于 STM32 微处理器的完整人工智能（AI）项目，自动转换预训练好的神经网络（NN）并集成所生成的优化库。本手册还介绍了 X-CUBE-AI 扩展包，该扩展包与 STM32CubeMX 工具完全集成。本用户手册还介绍了可选插件式 AI 测试应用程序或者用于 AI 系统性能和验证的实用工具。

本文档的第一部分是快速生成 STM32 AI 项目的实践性学习。采用一个 NUCLEO-F746ZG 开发套件和公开的几个用于深度学习（DL）的模型作为实践范例。任何 STM32 开发套件或者基于 STM32F3、STM32F4、STM32L4、STM32L4+、STM32F7、STM32H7 或 STM32WB 系列微控制器的客户板也可在稍作修改后使用。

本文档的第二部分详细说明了 X-CUBE-AI 自动生成的 NN 库，以及嵌入式客户端推理 API。这里还介绍了使用 X-CUBE-AI 实现 AI 性能和验证，以及各种 DL 工具箱中使用的功能。

1 概述

X-CUBE-AI 扩展包专门用于基于 STM32 Arm® Cortex®-M 的 MCU 上运行的 AI 项目。

当前版本用户手册中的描述基于：

- X-CUBE-AI r3.3.0
- 嵌入式推理客户端 API 1.0.0

用于本文档中各示例的预训练 Keras DL 模型包括：

- https://github.com/Shahnawax/HAR-CNN-Keras：在 Keras 中利用 CNN 实现人类活动识别
- https://github.com/Shahnawax/KWS-ANN-KERAS：KWS-ANN-KERAS - 关键词识别

提示 *Arm 是 Arm Limited（或其子公司）在美国和/或其他地区的注册商标。*

arm

1.1 STM32Cube™是什么？

STM32Cube™源自意法半导体，旨在通过减少开发工作量、时间和成本，明显提高设计人员的生产率。STM32Cube™涵盖整个 STM32 产品系列。

STM32Cube™包括：

- 一套用户友好的软件开发工具，覆盖从构想到实现的整个项目开发过程，其中包括：
  - STM32CubeMX 图形软件配置工具 STM32CubeMX，可通过图形向导自动生成初始化 C 代码。
  - STM32CubeProgrammer（STM32CubeProg），图形版本和命令行版本中可用的编程工具。
  - STM32CubeMonitor-Power（STM32CubeMonPwr），测量并帮助优化 MCU 功耗的监控工具。
- STM32Cube™ MCU 包，针对于每个微控制器系列的综合嵌入式软件平台（例如，STM32F4 系列的 STM32CubeF4），它包括：
  - STM32Cube™硬件抽象层（HAL），确保在 STM32 各个产品之间实现最大限度的可移植性。
  - STM32Cube™底层 API，通过硬件提供高度用户控制，确保最佳性能和内存开销
  - 一套一致的中间件，比如 RTOS、USB、TCP/IP 和图形。
  - 配备完整外设和应用示例的全部嵌入式软件实用工具

1.2 X-CUBE-AI 如何补充 STM32Cube™？

X-CUBE-AI 通过提供在计算和存储器（RAM 和闪存）方面均已优化的自动神经网络库生成器扩展 STM32CubeMX，该生成器将预训练的神经网络从最常用的 DL 框架（如 Caffe、Keras、Lasagne 和 ConvnetJS）转换为自动集成到最终用户项目的库。项目自动完成设置，准备好在 STM32 微控制器上进行编译和执行。

X-CUBE-AI 还为项目创建添加特定 MCU 过滤，从而选择符合用户 NN 特定标准要求（如 RAM 或闪存大小）的正确设备，进而扩展 STM32CubeMX。

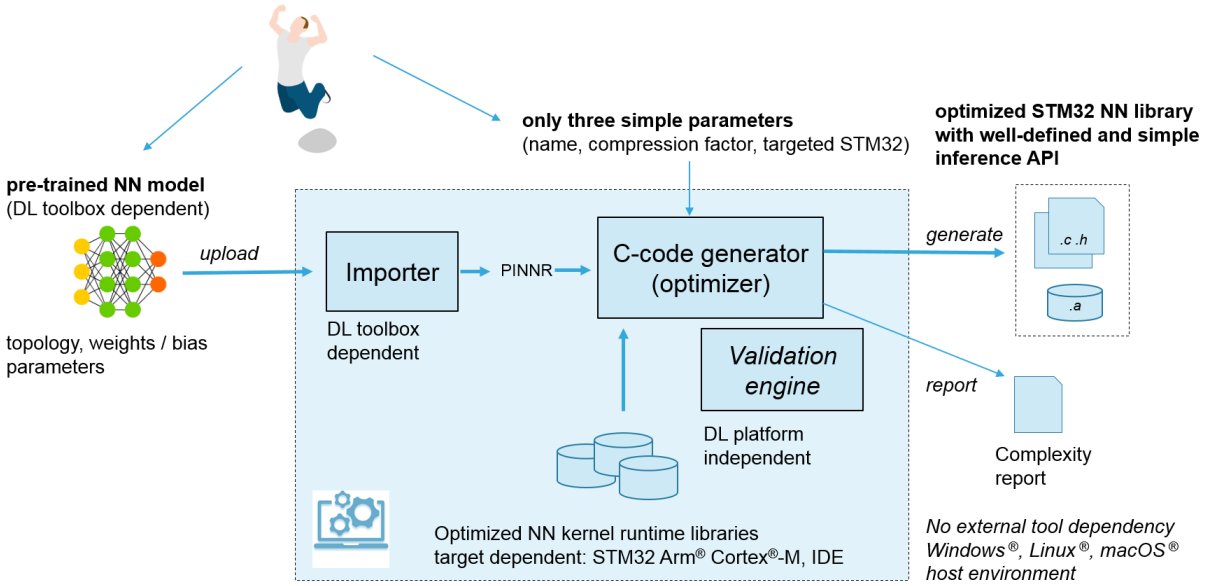
X-CUBE-AI 工具可生成三种项目：

- STM32 MCU 上运行的系统性能项目，可精确测量 NN 推理 CPU 负载和存储器使用情况
- 验证项目，在桌面 PC 和 STM32 Arm® Cortex®-M 的 MCU 嵌入式环境下递增地验证 NN 在随机或用户测试数据激励下返回的结果。
- 应用模板项目，可构建包括多网络支持的应用程序

1.3 X-CUBE-AI 内核引擎

X-CUBE-AI 内核引擎，如图 1 和图 2 所示，是 X-CUBE-AI 扩展包的一部分，之后将在第 1.3 节 介绍。它提供一个自动且先进的 NN 映射工具，利用有限并受约束的硬件资源为嵌入式系统的预训练神经网络（DL 模型）生成并部署优化且稳定的 C 模型。生成的 STM32 NN 库（专用和通用部分）可直接集成到 IDE 项目或者基于 makefile 的构建系统。还可导出定义明确且特定的推理客户端 API（参考第 8 节 嵌入式推理客户端 API），用于开发客户端基于 AI 的应用程序。支持各种用于深入学习的框架（DL 工具箱）和层（参考第 12 节 支持的深度学习工具箱和层深度学习）。

图 1. **X-CUBE-AI 内核引擎**

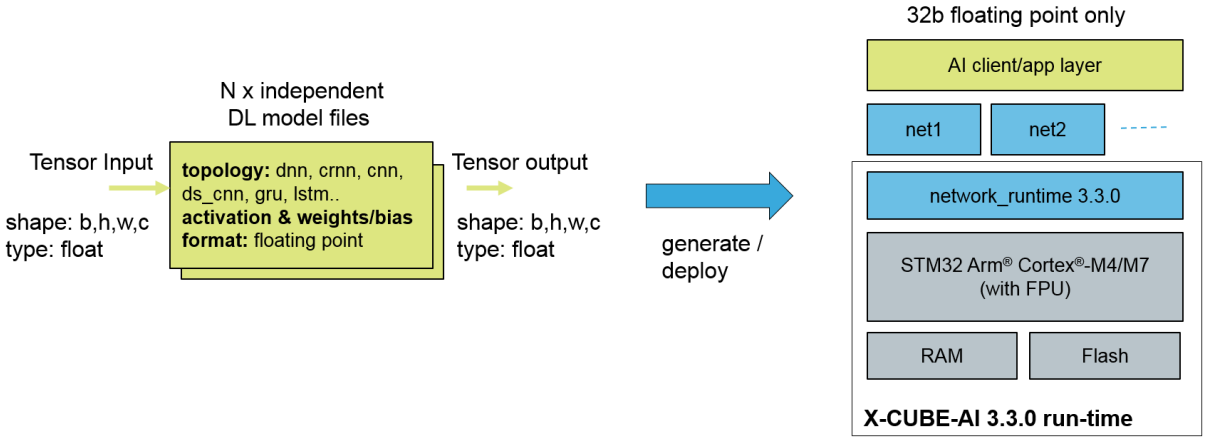


配备一个简单的配置接口。使用预训练的 DL 模型文件时，仅需要较少的参数：

- 名称：表示生成的 C 模型的名称（默认值为*"network"*）
- 压缩：表示用于减小权重/偏差参数大小的压缩系数（参考第 6.1 节 图形流和存储器布局优化器）
- STM32 系列：选择优化的 NN 内核运行时库

图 2 图 2 总结了上传的 DL 模型和目标子系统运行时的主要支持功能。

图 2. **X-CUBE-AI 3.3.0 概述**

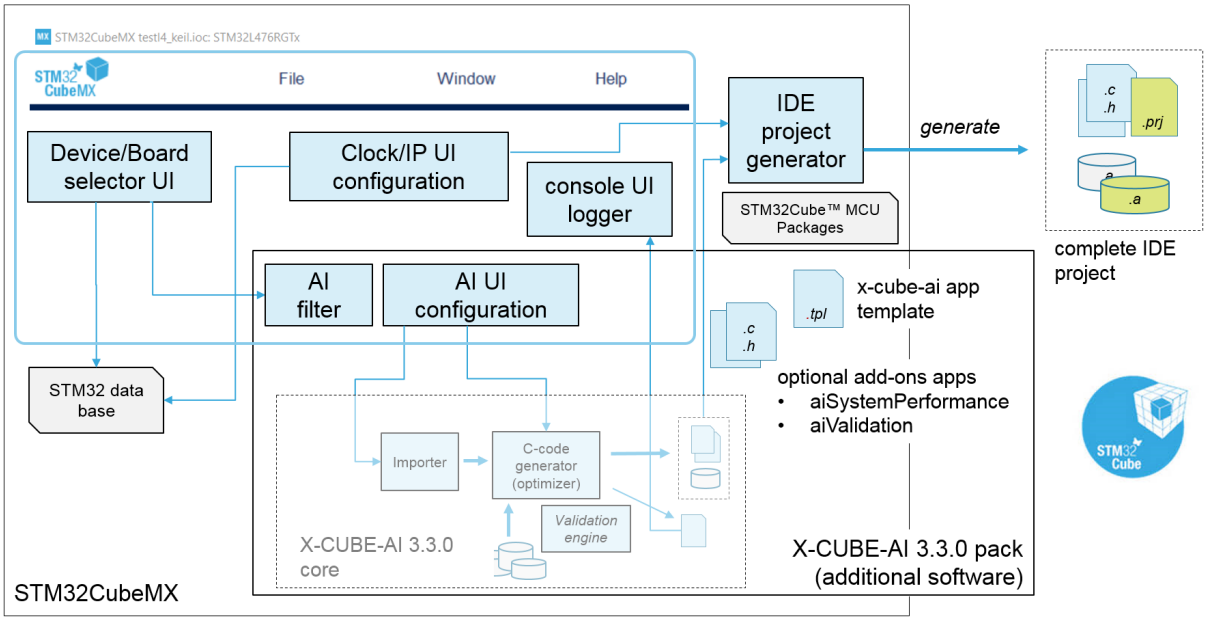


- 仅支持简单的张量输入和简单的张量输出
  - 4 维形状：批、高度、宽度、通道（参考第 8.1 节 输入和输出 x-D 张量布局）
  - 仅浮点类型
- 仅支持 32 位浮点生成的 C 模型
  - 生成过程块（参考第 6.1 节 图形流和存储器布局优化器）
  - 通过结构保证预训练模型的精度

1.4 STM32CubeMX 扩展

STM32CubeMX 是用于 STM32 微处理器的软件配置工具。单击一下，它即可利用图形向导（如引脚分配冲突处理器、时钟树设置助手等）为 STM32 创建完整的 IDE 项目，包括生成用于设备和平台设置（引脚、时钟树、外设和中间件）的 C 初始化代码。

图 3. **STM32CubeMX 中的 X-CUBE-AI 内核**



从用户的角度来看，集成 X-CUBE-AI 扩展包可以视为添加 IP 或中间件 SW 组件。基于 X-CUBE-AI 内核，提供了以下主要功能：

- 利用可选特定 AI 过滤器移除内存不足的设备，扩展 MCU 过滤器选择器。如启用，将直接滤除没有 Arm® Cortex®-M4 或-M7 内核的 STM32 设备。
- 提供完整的 AI UI 配置向导，可上传多个 DL 模型。包括在桌面 PC 和目标上对生成的 C 代码进行的验证过程。
- 扩展 IDE 项目生成器，辅助生成优化 STM32 NN 库并将其集成到选定 STM32 Arm® Cortex®-M 内核与 IDE。
- 可选插件式应用程序可生成完整的即用型 AI 测试应用程序项目，包括生成的 NN 库。用户必须已将其导入到首选 IDE 内，才能生成固件映像并进行刷写。最终用户无需附加代码或修改。

1.5 缩写、缩略语和定义

表 1 详细列出了本文档中所用的特定缩略词和缩写词。

表 1. **本文中所用术语的定义**

AI	人工智能人工智能，有时也称机器智能。通常，AI 是对从人类角度认为可以“智能”地执行任务的机器的广义概念。它代表数字设备能够执行智能相关的任务。
DL	深度学习 （也称为深层结构学习或层级学习）。 DL 模型略微受到生物神经系统中信息处理和通信模式的启发。
ML	机器学习是人工智能（AI）的一种应用，使系统能够自动从经验中学习和改进，无需明确编程。
MACC	乘积累加运算复杂度是从处理的角度表示 DL 模型复杂度的统一体。
PINNR	平台独立神经网络表示是前端（X-CUBE-AI 内核导入器）生成的一个文件，包含所上传的 DL 模型的普遍且可移植的内部表示，供后续阶段（优化器和 C 代码生成器）使用。

1.6 先决条件

必须安装以下软件包（参考第 2 节 安装 X-CUBE-AI）：

- STM32CubeMX 5.0.1 或更高版本
- 附加软件包 - STM32CubeMX AI 3.3.0 包

必须安装用于 STM32 的一个以下工具链或 IDE：

- TrueSTUDIO® 适用于 STM32 v9.0.1 或更高版本的 TrueSTUDIO®（atollic.com/truestudio）
- IAR Embedded Workbench™ IDE - ARM v8.x or v7.x（www.iar.com/iar-embedded-workbench）
- µVision® V5.25.2.0 - Keil® MDK-ARM 专业版（www.keil.com）
- System Workbench for STM32（SW4STM32）
- GNU Arm Embedded Toolchain（developer.arm.com/open-source/gnu-toolchain/gnu-rm）

可以使用 STM32CubeMX 支持的所有操作系统：

- Windows® 10 和 Windows® 7
- Ubuntu® 18.4 和 Ubuntu® 16.4（或衍生版本）
- macOS®（x64）（已在 OS X® El Capitan 和 Sierra 上测试）

提示

Ubuntu®是 Canonical Ltd.的注册商标

macOS®和 OS X®是苹果公司在美国及其他国家注册的商标。

1.7 授权

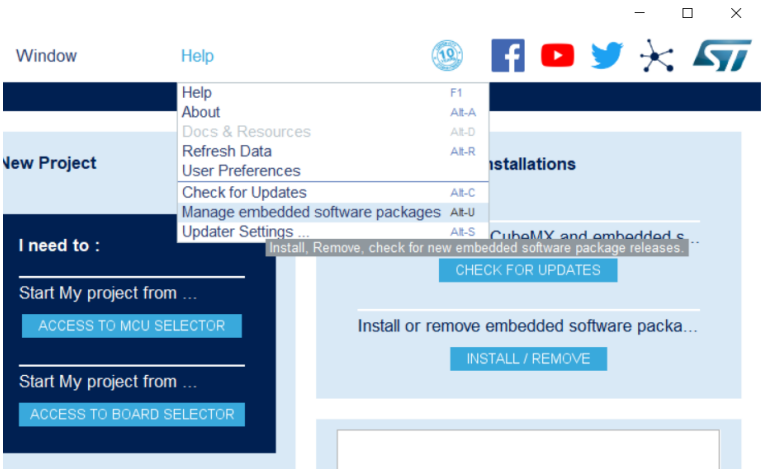
X-CUBE-AI 根据 *Mix Ultimate Liberty+OSS+3rd-party V1* 软件许可协议（SLA0048）交付。

2 安装 X-CUBE-AI

下载、安装并启动 STM32CubeMX（5.0.1 或更高版本）后，几步即可安装 X-CUBE-AI 软件包。

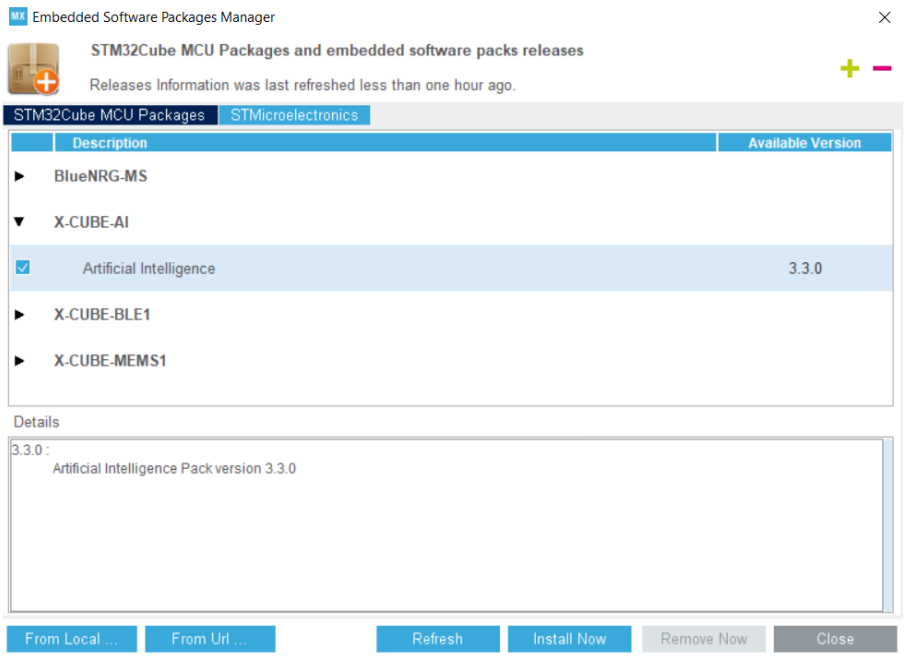
1. 从菜单中选择 [帮助] > [管理嵌入式软件包] 或者直接单击 [安装/移除] 按钮。

图 4. 管理 STM32CubeMX 中的嵌入式软件包



2. 从 [嵌入式软件包管理器] 窗口，点击 [刷新] 按钮，获得扩展包的更新列表。进入 [STMicroelectronics] 选项卡，找到 X-CUBE-AI。

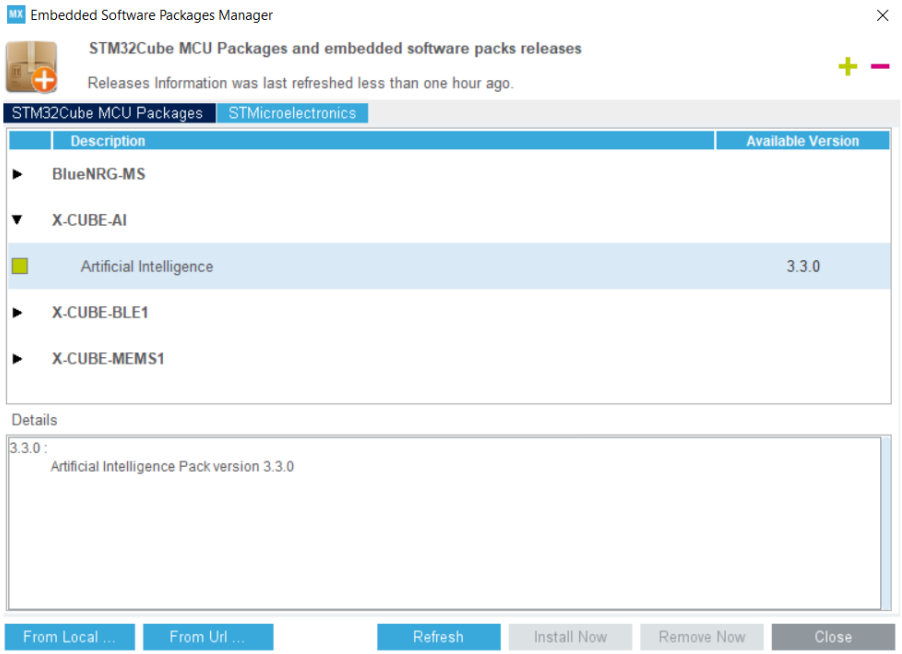
图 5. 在 STM32CubeMX 中安装 X-CUBE-AI



如果已安装 X-CUBE-AI，最好将其移除，然后再重新安装。

3. 选中相应的复选框，然后点击 [立即安装] 按钮进行安装。安装完成后，相应框变为绿色并且可以点击 [关闭] 按钮。

图 6. **X-CUBE-AI in STM32CubeMX**

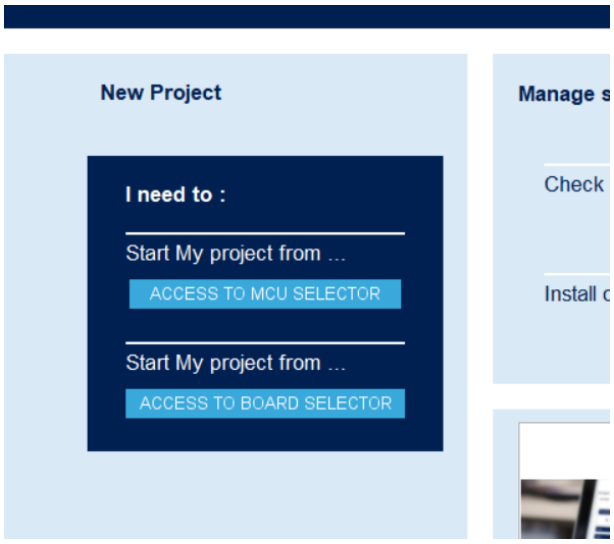


3 启动一个新的 STM32 AI 项目

3.1 MCU 和开发板选择器

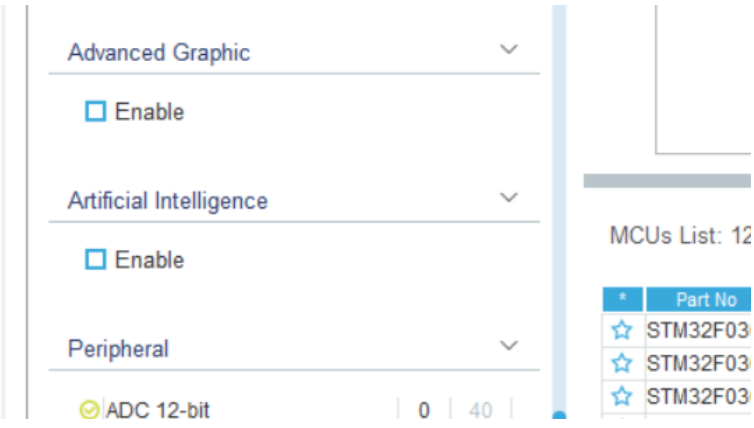
启动 STM32CubeMX 应用程序后，单击[访问 MCU 选择器]或[访问开发板选择器]按钮。或者，选择[文件] > [新建项目...]或者 CTRL-N 快捷键。

图 7. 创建一个新项目



此时，可使用典型的 STM32CubeMX 流程选择一个特定 MCU 或开发板。利用可选 MCU 过滤器条目可排除没有足够嵌入式存储器（RAM、闪存或两者）来存储优化 STM32 NN 库的 MCU。该特定 AI 过滤器如图 8 所示。

图 8. AI 滤波器



提示 *此功能不适用于开发板选择器，仅可用于一个 NN 模型。*

图 9 显示了在默认选项情况下上传并分析 DL 模型的范例。使用公开发布的预训练 NN 模型（Keras 类型）：在Keras 中使用 CNN 实现人类活动识别。

图 9. 默认选项下的 AI 过滤器

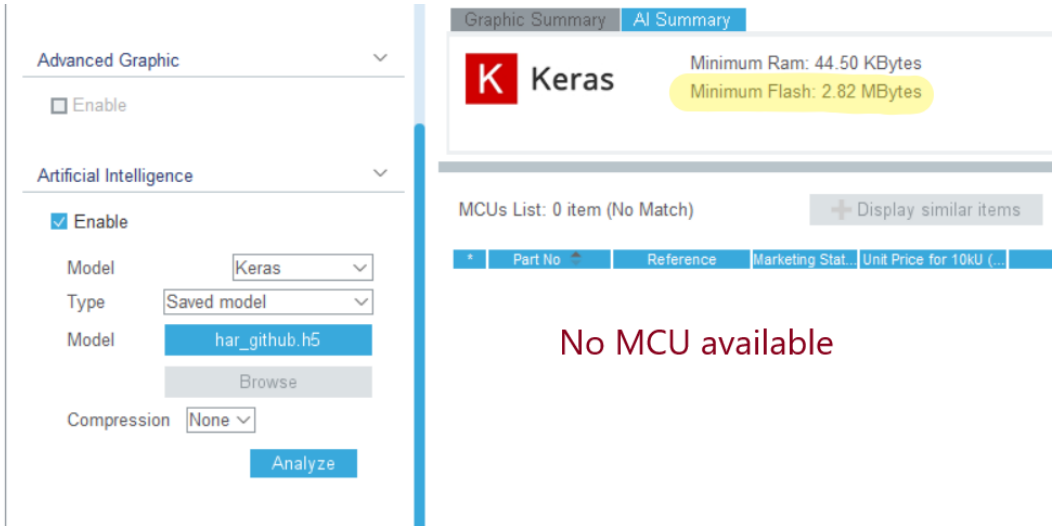
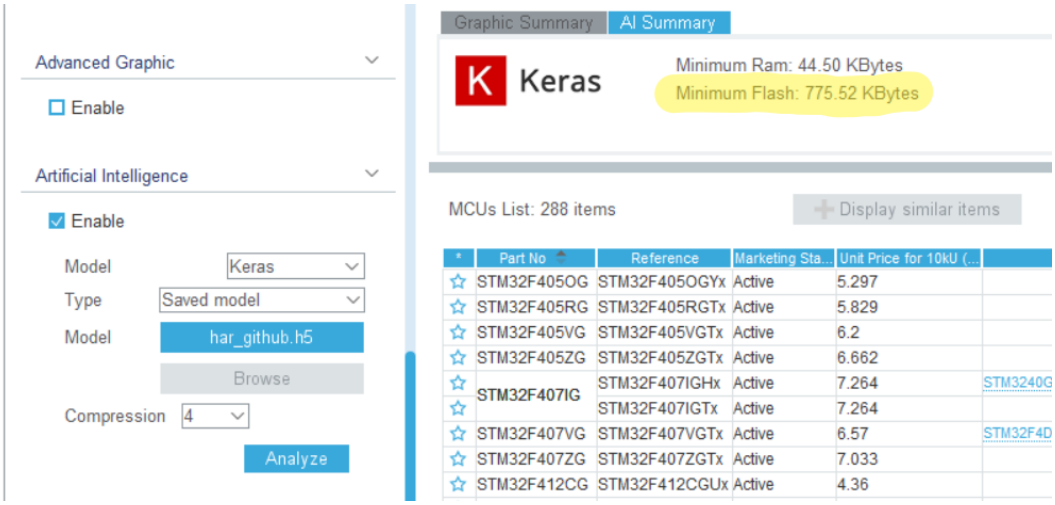


图 10 显示了应用压缩系数 4 的范例。

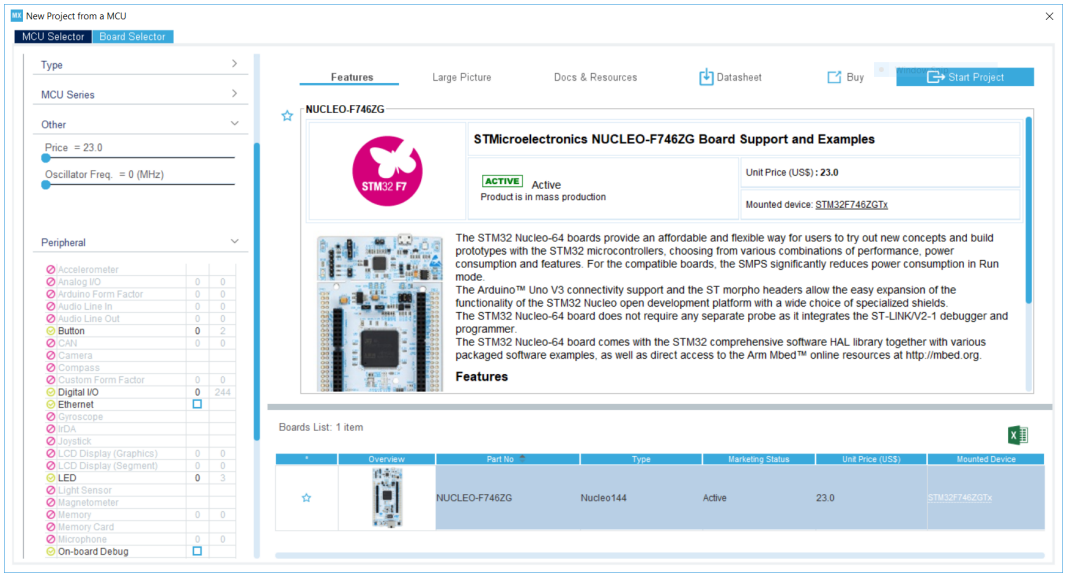
图 10. 压缩系数 x4 的 AI 过滤器



提示 *生成 NN 库期间，还会通过优化器检查存储器大小，如果未根据选定 MCU 遵守最小 RAM 和闪存大小限制，则会通知用户。*

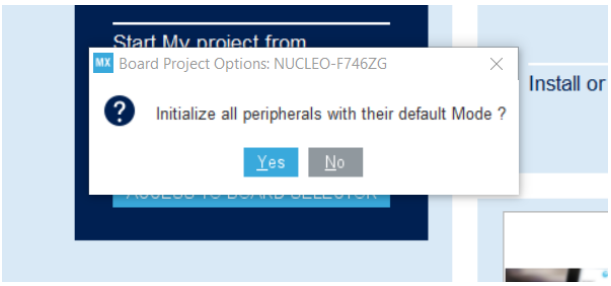
若要继续，选择 NUCLEO-F746ZG 开发套件，如图 11 所示。

图 11. NUCLEO-F746ZG 板选择



单击 [启动项目] 按钮，继续并确认所有外设必须在默认模式下进行初始化。

图 12. 初始化所有外设

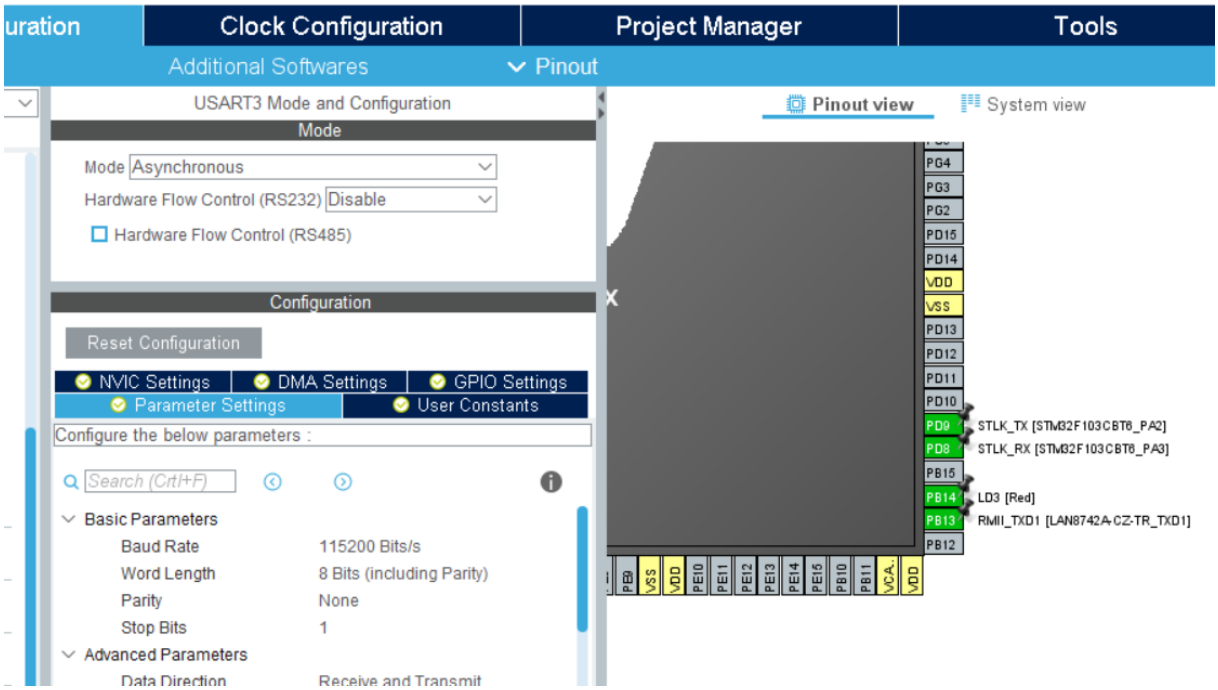


3.2 硬件和软件平台设置

选择一个 MCU 或开发板后，就会显示相关的 STM32 引脚布局。在该窗口中，用户可以通过添加一个或多个附加软件和外设以及配置时钟来设置项目。

如果使用插件式 AI 应用程序（参考第 4.1 节 添加 X-CUBE-AI 组件），预计基于 UART 与主机开发系统连接。对于 STM32 Nucleo-144 开发板，PD9 TX 和 PD8 RX 引脚连接到 ST-LINK IP，以支持虚拟 COM 端口（参考图 13）。

图 13. USART3 配置



对于 NUCLEO-F746ZG，预计还要额外配置时钟和存储器子系统，才能达到高性能配置文件。

3.2.1 增加或设置 CPU 和系统时钟频率

1. 单击 [时钟配置] 选项卡。
   默认情况下，在该选项卡中，系统时钟（SYSCLK, HCLK）为 72 MHz。
2. 在 HCLK（MHz）输入蓝框中输入 216（参考图 15），调用时钟向导来自动配置 PLL IP（和相关的时钟树）。如果显示如图 14 所示的时钟向导弹出框，单击 [确定] 按钮继续。

图 14. 时钟向导弹出框

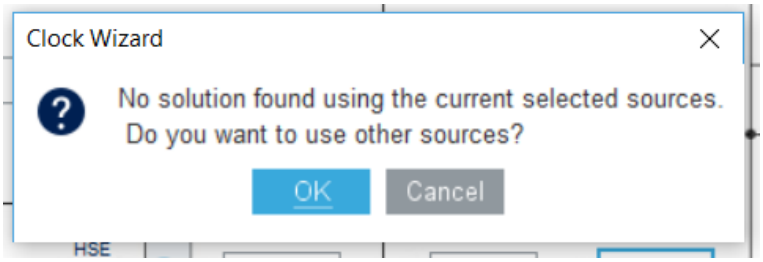
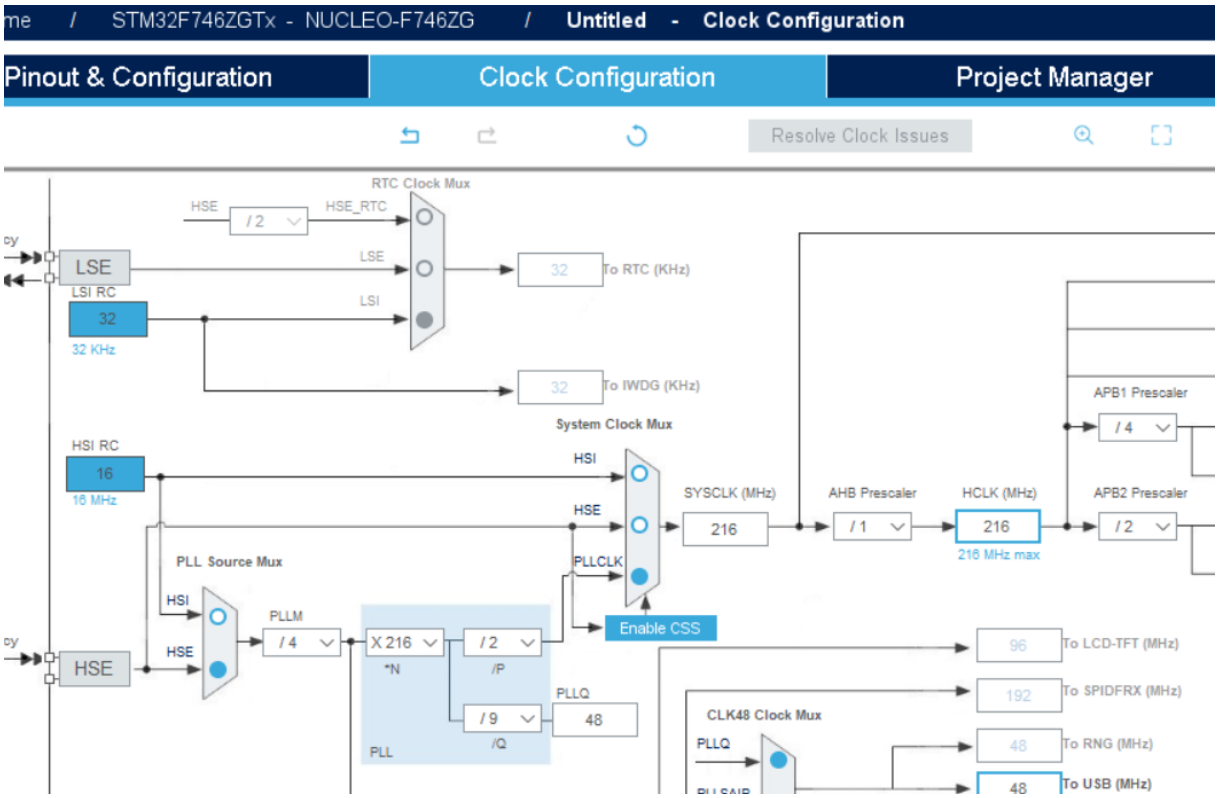


图 15. 系统时钟设置

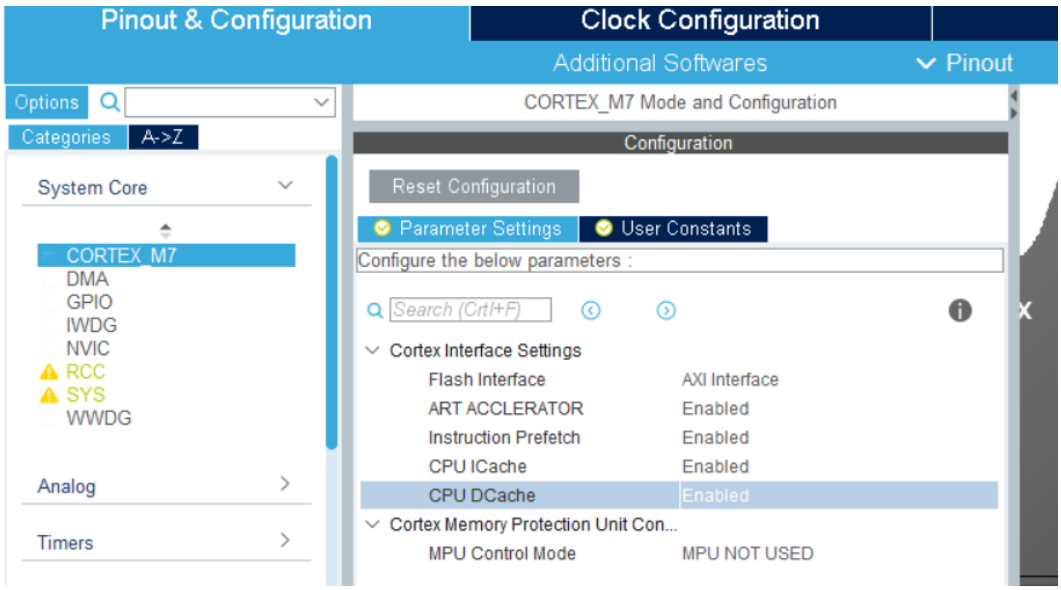


3.2.2 设置 MCU 存储器子系统

- 从[引脚布局与配置]选项卡中（参考图 16），单击[系统内核] > [CORTEX_M7]条目，打开 Cortex®-M7 配置向导。

  必须启用内核指令和数据缓存以及 ART 加速器子系统。

图 16. MCU 存储器子系统（参数设置）



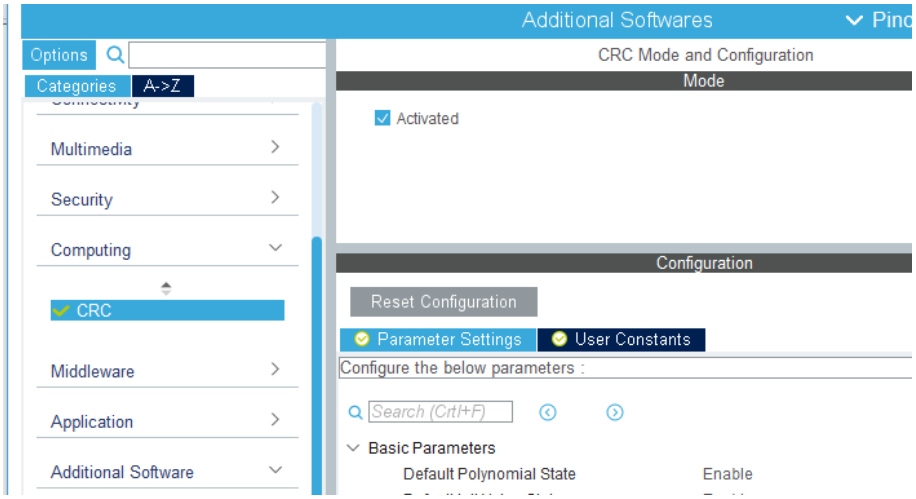
提示 *不必为 MCU 时钟设置最大值。但是必须与最终设计中使用的配置一致。闪存等待状态的设置由 STM32CubeMX 平台代码生成器自动调整。*

3.2.3 CRC

需要 CRC IP 来支持 NN 库运行时保护机制。必须启用。

提示 *当 X-CUBE-AI 组件添加到当前项目时，此 IP 自动启用。*

图 17. 启用 CRC IP



4 X-CUBE-AI 配置向导

4.1 添加 X-CUBE-AI 组件

1. 单击[附加软件]按钮，将 X-CUBE-AI 附加软件添加到项目（参考图 18）。

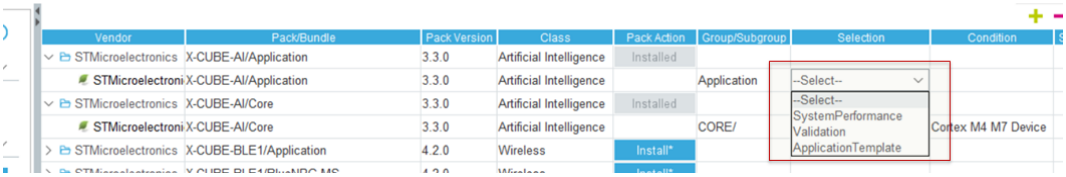
图 18. 附加软件按钮

2. 从[附加软件组件选择]窗口中，必须选中 X-CUBE-AI/Core 选项框（参考图 19）才能上传 NN 模型并生成相关的 STM32 NN 库。这种情况下，该库完全集成为静态库，只需要将基于 AI 的应用程序/中间件实施到所生成的完善 NN API 顶部 [6]。

图 19. 添加 X-CUBE-AI 内核组件

3. 或者，可以从 X-CUBE-AI/应用程序包（参考图 20）中选择一个插件式 X-CUBE-AI 应用程序。
   - 系统性能：独立的 AI 性能测试应用程序
   - 验证：AI 验证测试应用程序
   - 模板应用：AI 应用程序的基本应用模板

图 20. 插件式 X-CUBE-AI 应用程序



4. 单击[确定]，完成选择

4.2 启用 X-CUBE-AI 组件

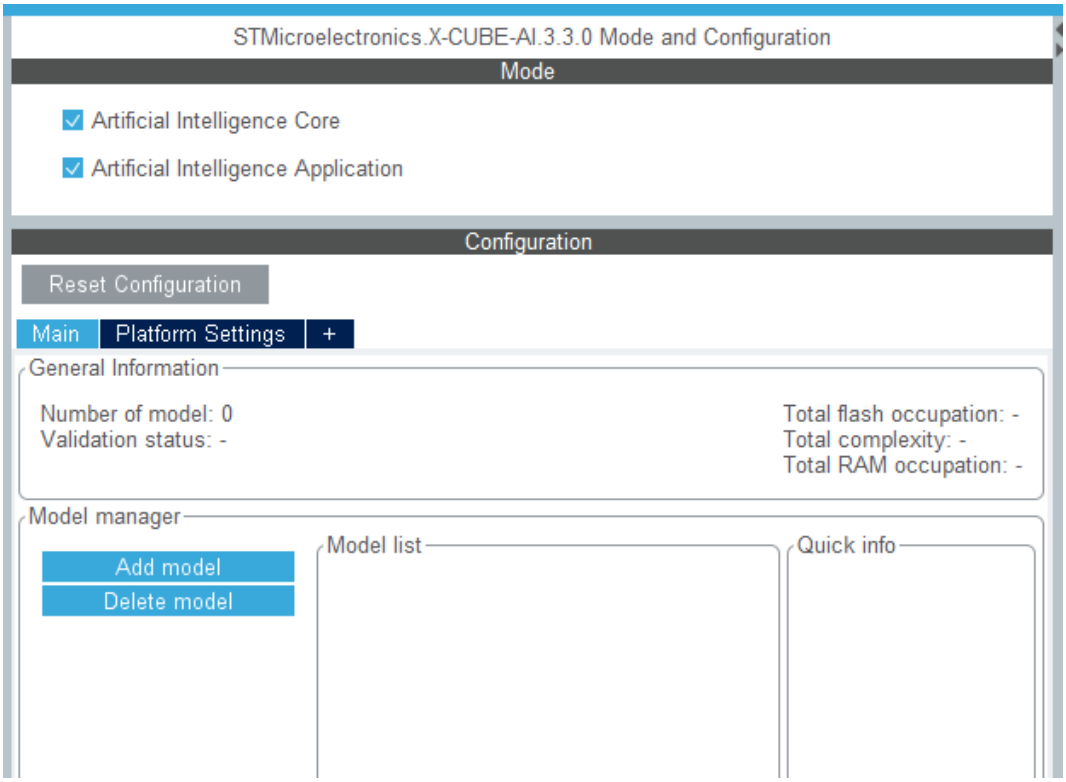
要启用和配置 X-CUBE-AI 组件，需要执行以下附加步骤：

1. 从[引脚布局与配置]选项卡中，单击[附加软件]选择器，找到附加软件。单击[STMicroelectronics X-CUBE-AI 3.3.0]打开初始 AI 配置窗口。

2. 选中 [人工智能内核]，启用 X-CUBE-AI 内核组件。还必须选中 [人工智能应用]，才能添加插件式 AI 应用程序。

   这里选择的 AI 应用程序与前一步中启用的应用程序一致（参考图 20）。

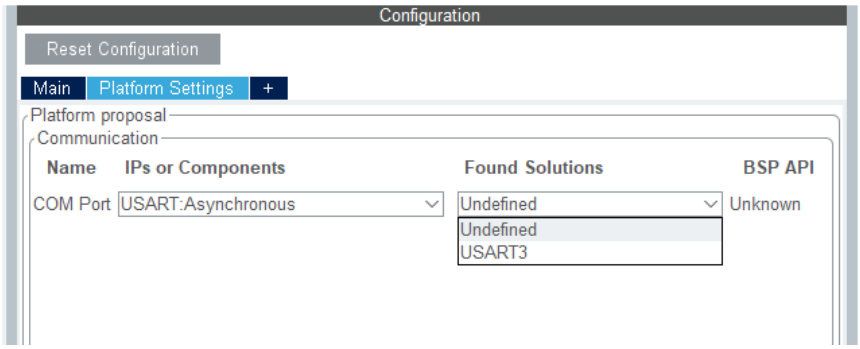
图 21. 主要 X-CUBE-AI 配置面板



– [主要] 选项卡提供概览以及添加或移除网络（分别使用 [添加模型] 和 [删除模型] 按钮）的入口点。也可以直接使用 [+] 添加网络。

– [平台设置] 选项卡指示用于报告信息（AI 系统性能应用程序）或与主机通信（AI 验证应用程序）的 UART IP 句柄。

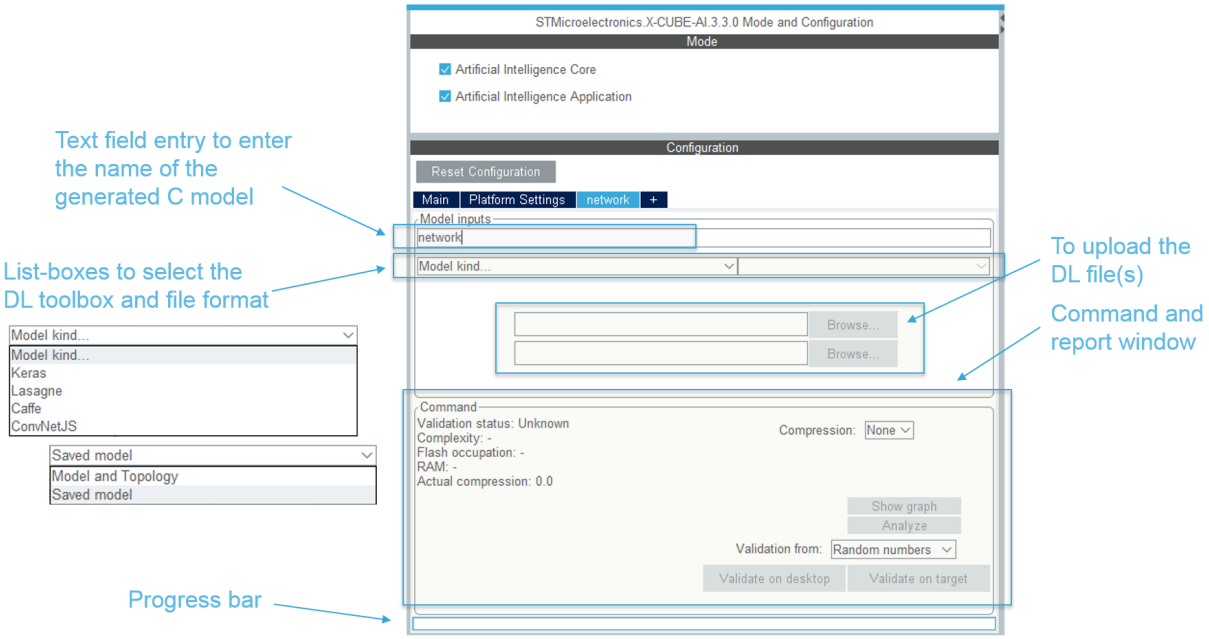
图 22. X-CUBE-AI 平台设置面板



4.3 上传预训练的 DL 模型文件

从 [主要] 选项卡中，单击 [添加模型] 或直接单击 [+]，打开新的 <模型名称> 专用配置向导。或者，如果之前已经通过 MCU 过滤器提供了该模型，则直接单击 [网络] 选项卡，打开 NN *配置*窗格。

图 23. NN 配置向导



1. 用于定义网络 C 模型名称的文本框（最多 32 个字符）。该字符串直接用于生成嵌入式客户端推理 API 的名称（参考[6]）。如果预计只有一个网络，可保留默认网络字符串名称。
2. 指定用于导出 DL 模型文件和相关文件格式的 DL 工具箱的列表框（有关详细信息，请参考第 12 节 支持的深度学习工具箱和层深度学习）。
   – 单击 [浏览..] 按钮，上传主机文件系统中的 DL 文件。对于该实际操作选项卡，上传公共 Keras HAR模型文件（保存的模型格式）。
3. 单击 [分析..] 按钮，触发维度信息报告网络的预分析（系统集成角度）。注意，压缩系数设置小于 4，否则会弹出警告消息，如图 24 所示。如果弹出 [无效网络] 消息框，选择 [窗口] > [输出]，查看日志控制台中的更多详细信息（参考第 13 节 错误处理）。更新最小 RAM、闪存占用情况和原始 DL 模型复杂度（参考第 4.4 节 ）。

图 24. RAM/闪存不足消息框

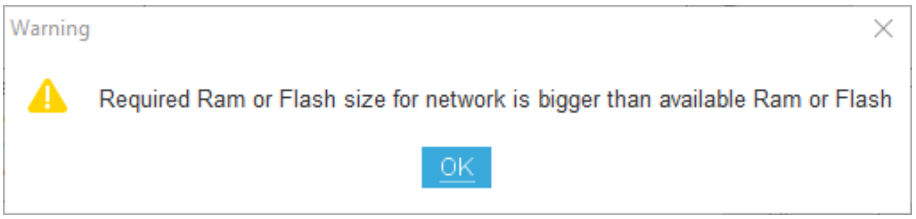
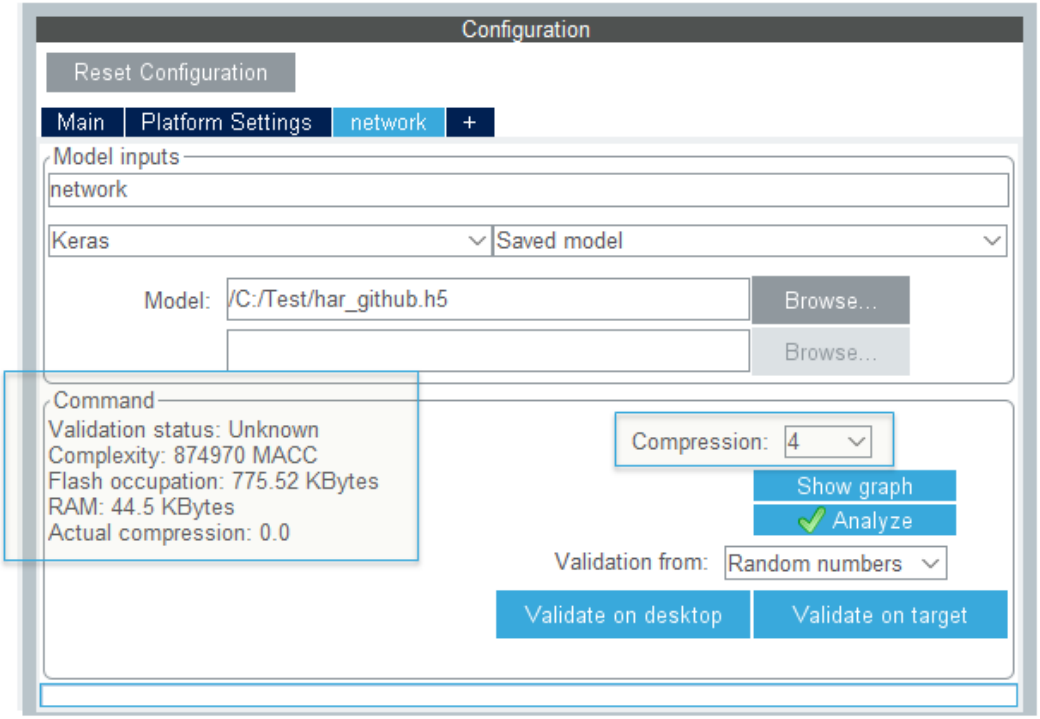


图 25. 上传并分析 DL 模型



提示 *附加调试/日志信息可在文件“C:\Users\<username>\.stm32cubemx\STM32CubeMX.log ” or “$HOME/.stm32cubemx/STM32CubeMX.log”中找到。*

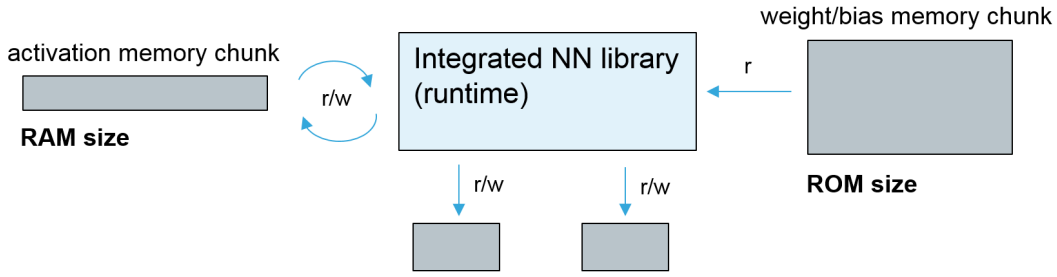
4.4 维度信息报告

当处理 DL 模型时，将报告表 2 和图 26 中展示的维度系统信息。

表 2. 系统信息报告

报告的信息	说明
RAM	指示用于存储中间推理计算值（.data 或.bss 部分）的预期 RW 存储块的大小（字节）。
ROM/Flash	如有要求，指示用于存储压缩后权重/偏差参数（.rodata 部分）的生成 RD 存储块的大小。
复杂度	指示在乘法累加运算（MACC）中导入的 DL 模型的功能复杂度。它还包括激活功能的近似值（采用相同单位表达）。

图 26. 集成 C 模型（运行时视图）



提示 *AI 总结中列出的最小 RAM 和闪存大小要求未考虑用户应用程序的存储器限制（包括存储输入和输出张量的 RAM）。这里只考虑了 DL 模型权重/偏差和激活存储器要求。 NN 暂且还没有考虑内核功能以及专业模型代码，包括最小栈/堆大小。*

4.4.1 CPU 周期/MACC

NN C 库的报告复杂度与实际性能之间没有理论关系（CPU 周期/MACC）。由于目标环境的多变性（包括 Arm®工具链、MCU 和底层子系统存储器设置、NN 拓扑结构和各层以及应用的优化），很难离线估计准确的 CPU 周期/MACC，由于不同的 STM32 系统设置。但是，可以直接使用以下粗略估计数据（用于 32 位浮点 C 模型）：

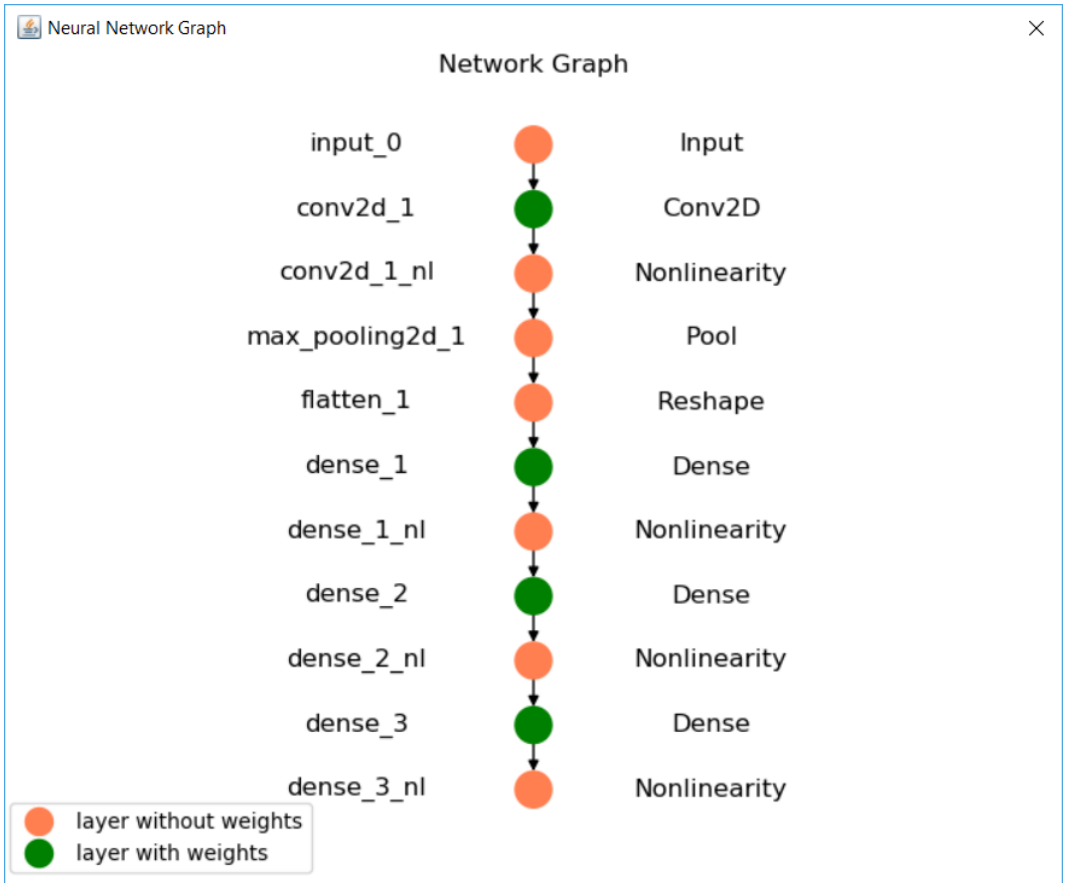
- STM32 Arm® Cortex®-M4：~9 个周期/MACC
- STM32 Arm® Cortex®-M7：~6 个周期/MACC

插件式*"AI 系统性能"*测试应用程序专门设计用于报告真实的设备性能（有关详细信息，请参考第 9 节 AI 系统性能应用程序）。

4.4.2 所生成的 C 模型的图形表示

单击 [显示图形] 按钮，显示 C 代码生成器考虑的上传 DL 模型的主要结构信息（图 27）。

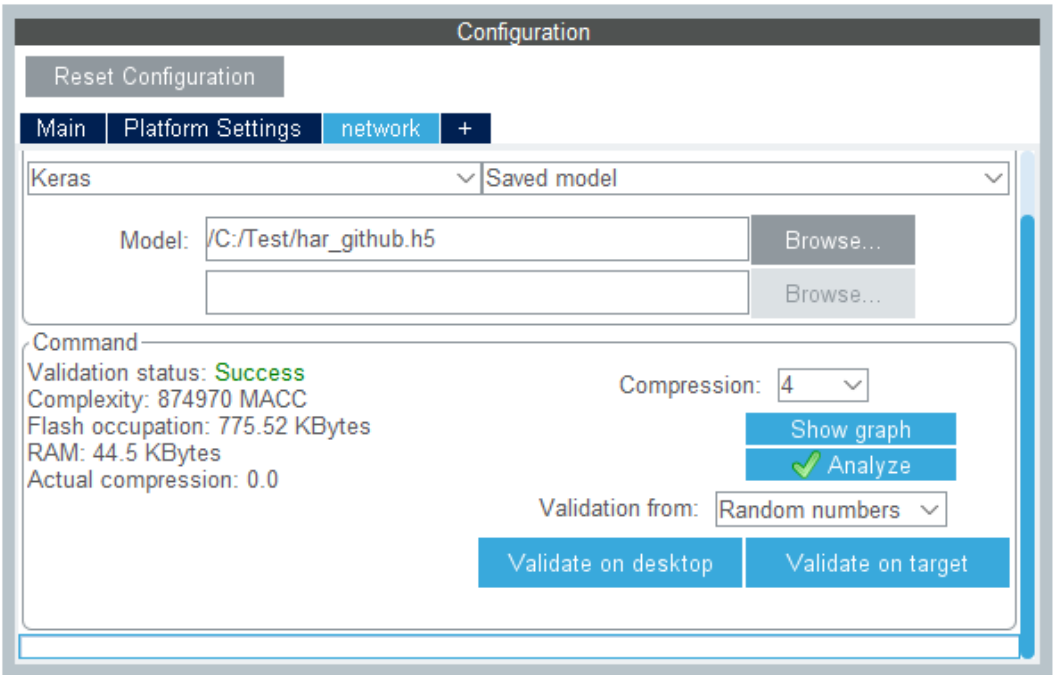
图 27. 所生成的 C 模型的图形



4.5 验证所生成的 C 模型

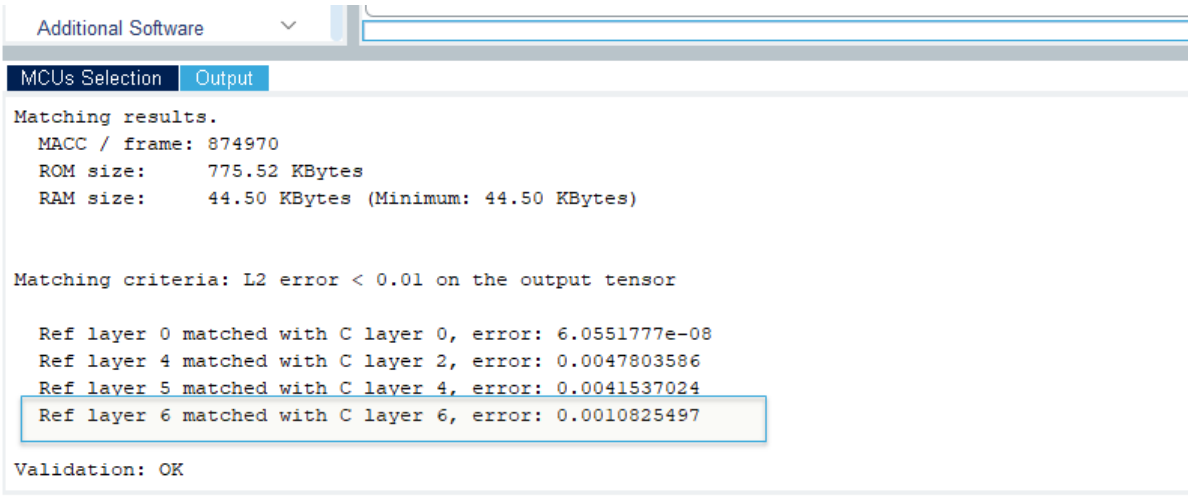
单击 [桌面上验证] 按钮，启动所生成的 C 模型的验证过程。[验证状态] 字段（参考图 28）会根据所报告的 L2 相对误差值（参考第 6.2 节 验证引擎）更新。注意，这一步是可选的，但是最好执行，尤其是要求压缩系数时（参考第 6.1 节 图形流和存储器布局优化器）。

图 28. 验证状态字段



更多详细信息报告在 UI 日志控制台中（[输出]窗口），如图 29 所示。尤其是还报告每个所生成的 C 层的 L2 误差。

图 29. 桌面上验证 - 日志报告



提示 *由于优化过程，报告“Ref layer X matched with C layer Y, error: 0.0...”会提供 DL 模型（X）原始层与优化 C 层（Y）之间的关联。请参考“操作融合”优化（参见第 6.1 节 图形流和存储器布局优化器）。*

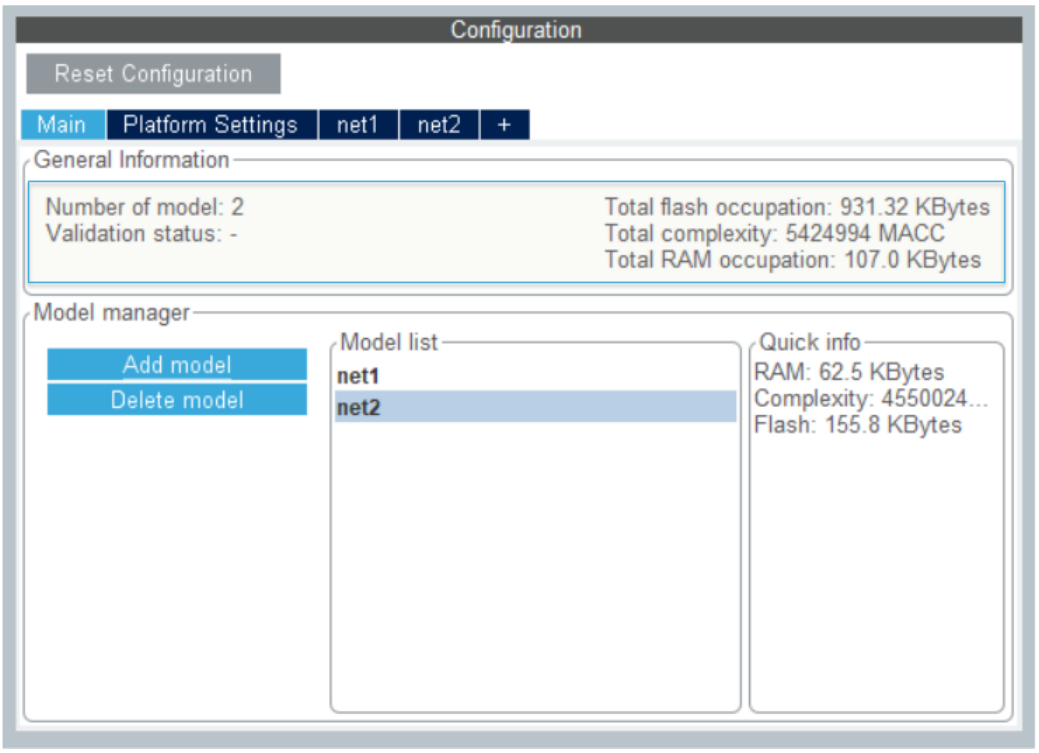
在这步中，上传的 DL 模型已准备好集成到所生成的 IDE 项目。

只有在使用特殊测试应用程序“AI 验证”刷写目标设备之后才能使用 [目标上验证] 选项。“”必须在前一步中（第 4.1 节 添加 X-CUBE-AI 组件中的第 3 步）选择此选项。报告信息和使用情况在第 10 节 AI 验证应用程序中有充分说明。

4.6 添加新 DL 模型

可导入多个 DL 模型。总数不受向导限制，但是，初始限制大多数与选定 STM32 MCU 设备中的可用 RAM 和闪存大小有关。单击 [+] 按钮，导入新 DL 模型并采用与之前相同的步骤。主要视图会总结总 RAM 和闪存占用情况。

图 30. 包含多个网络的主要视图



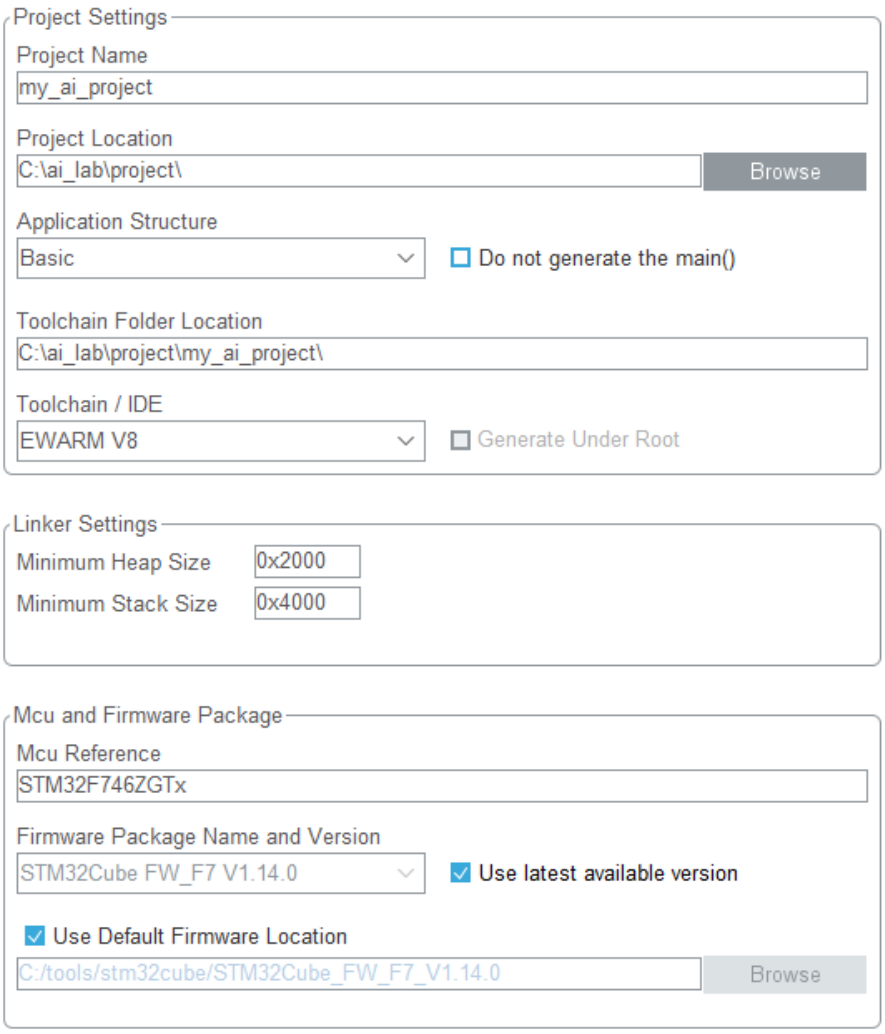
5 生成、构建和烧录

5.1 生成 IDE 项目

以下步骤按顺序显示了在不添加任何特定 AI 扩展名情况下生成 IDE 项目的典型 STM32CubeMX 过程：

1. 单击 [项目管理器] 视图
2. 设置项目位置和名称
3. 选择一个工具链和 IDE（如 EWARM for IAR™、TrueSTUDIO® for Atollic IDE 或其他）
4. 更新最小堆栈大小，在第一时间最大限度减少可能的溢出（预计*"AI 验证"*测试应用程序的堆最小 2 KB）

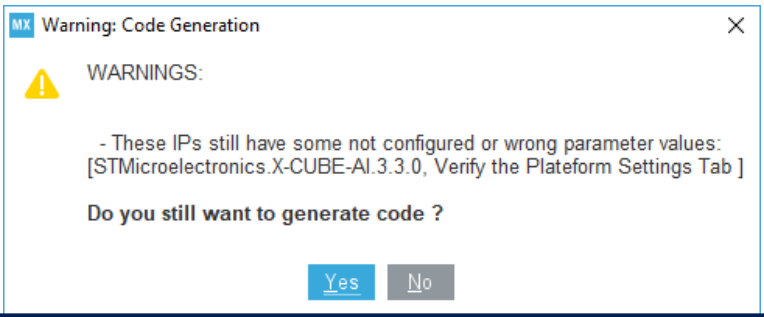
图 31. IDE 代码生成器的项目设置视图



5. 单击 [生成代码] 按钮，生成与当前项目配置（包括 IDE 项目文件）对应的代码

   生成 IDE 项目期间，如果用户已选择并启用插件式 AI 应用程序，但是忘记设置平台相关依赖（如 UART 句柄），则会弹出图 32 所示的消息框。有关详细信息，请参见第 4.1 节 添加 X-CUBE-AI 组件。

图 32. AI IP 配置不完整



5.2 构建和烧录

当使用 STM32CubeMX 工具成功生成 IDE 项目时，利用标准构建过程构建并烧录 STM32 板开发套件或客户板：

1. 启动 IDE 应用程序并打开所生成的项目文件
2. 构建并烧录固件映像。如果已经选择 AI 测试应用程序，预计不用修改或更新代码。否则必须添加基于用户 AI 的应用程序代码才能使用所生成的推理 C API。

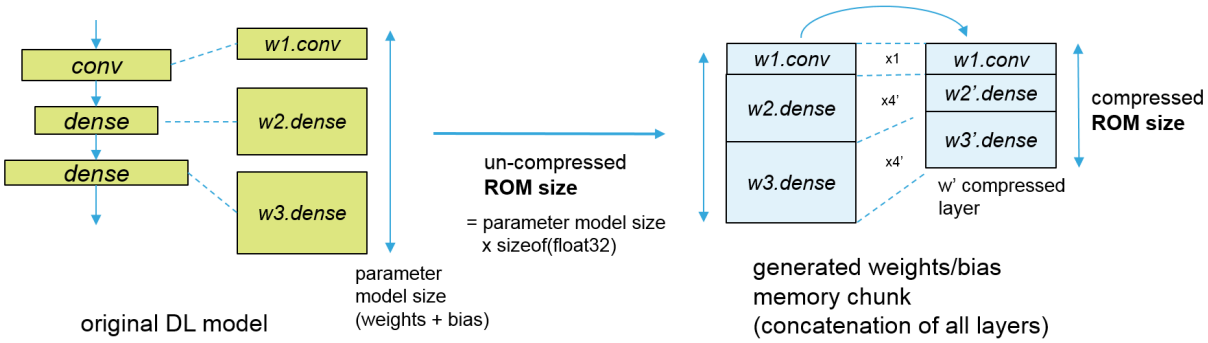
6 X-CUBE-AI 内部构件

6.1 图形流和存储器布局优化器

C 代码生成优化引擎力图根据推理计算时间（还考虑功耗）优化存储器使用（RAM & ROM）。该引擎基于无数据集的方法，意味着不需要经过训练的有效或测试数据集来应用压缩和优化算法（预计没有重新训练/重新定义权重/偏差阶段来维持初始模型的精度）。

- 权重/偏差压缩（目标系数：无、x4、x8）
  - 仅适用于稠密（或全连接）层类型
  - 采用基于权重共享的算法（K 均值聚类）
  - 如果选择“*无*”，则保证初始 DL 模型精度。残差（~10^{-08}）与使用 32 位 C 浮点时的原生模型 64 位浮点数大小有关。但是，对于大型网络，10^{-06} 更为常见。

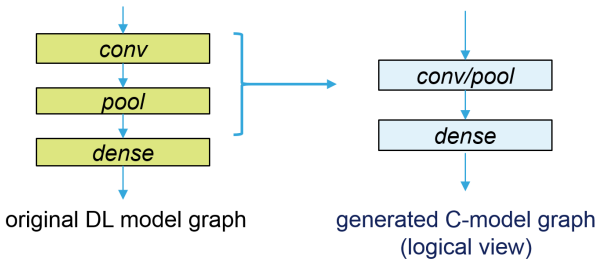
图 33. 权重/偏差压缩



这种方法的优势是压缩过程快，但是最终结果并不是无损的，全局精度会受到影响。所生成的 C 模型的“*验证*”过程是评估所产生的错误的缓解措施（参考第 6.2 节 ）。

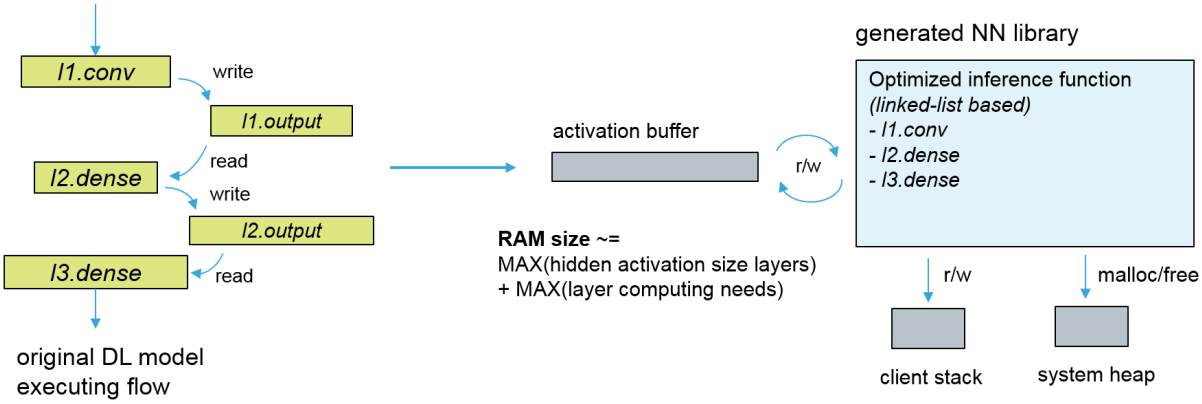
- 操作融合
  - 合并两层，从而优化数据放置和相关计算内核。转换或优化过程中会删除一些层（如“*丢包*”、“*重塑*”），而有些层（如非线性层以及卷积层之后的池化层）会融合到前一层中。效果是转换后的网络通常比原始网络的层数少。

图 34. 操作融合



- 最佳激活/工作存储器：定义一个 R/W 块来存储临时隐藏的层值（激活操作器的输出）。它可视为推理功能使用的暂存缓冲区。激活存储器在不同层之间重复使用。因此，激活缓冲区的大小由两个连续层的最大存储要求决定。

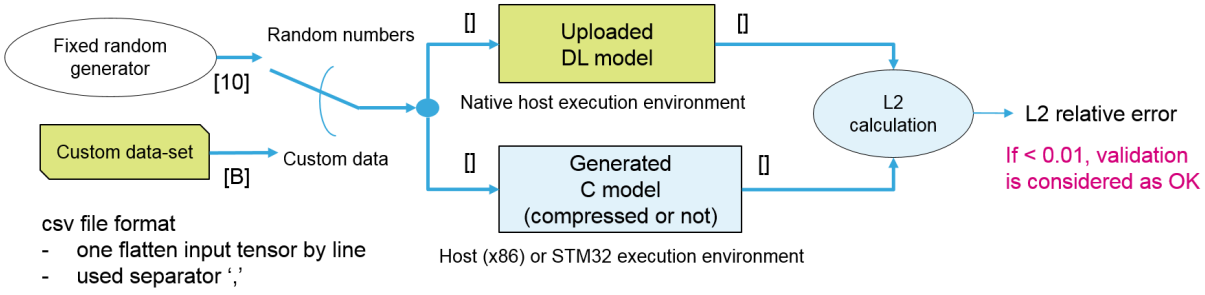
图 35. 最佳激活/工作缓冲区



6.2 验证引擎

提供一种简单快捷的验证机制，用于从数据角度对比所生成的 C 模型与上传的 DL 模型之间的精度（参考图 36）。为两个模型提供相同的输入张量（固定的随机输入或自定义数据集；参考第 14.2 节 定制数据集文件格式？）。然后计算所有推理的 L2 相对误差。仅使用输出层的 L2 相对误差指示所生成的 C 模型是否有效（有效性阈值设置在 0.01 以下）。X-CUBE-AI 扩展包为所有支持的 DL 框架提供一个推理 DL 执行引擎。注意，即使工具报告验证失败，仍可考虑优化所生成的 C 模型。性能可能与原始 Python™模型不一致，但是仍可使用 C 模型。需要利用自定义数据集和 NN 输出跟踪进行进一步检查。

图 36. 验证流程概览



使用图 37 中所示的公式计算 L2 相对误差，其中：

- F_j：C 代码层输出 j 的扁平化数组
- f_i：相应原始层输出 i 的扁平化数组

图 37. L2 计算

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

提供两种执行模式：

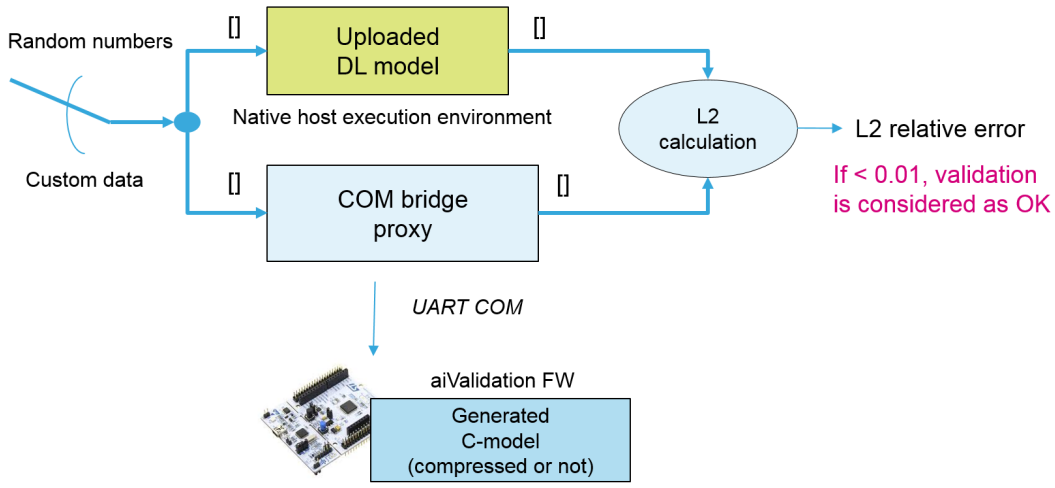
- [桌面上验证]：使用这种模式可以对比 DL 模型与其生成的 X86 C 模型。这种模式在主机上运行。相关输出如第 4.5 节 验证所生成的 C 模型中所示。

- [目标上验证]：这种模式对比 DL 模型与目标设备上运行的 C 模型。这种模式需要一个特殊的 AI 测试应用程序嵌入所生成的 NN 库中，还需要 COM 代理来与主机系统通信。输出和使用情况如第 10 节 AI 验证应用程序中所示。

[目标上验证]特征：

- 自动检测所连接的 STM32 板
- 使用经过验证的 DL 模型检查所生成的嵌入式 C 模型的签名
- 仅在最后一个输出层计算 L2 误差（由于上传数据的 COM 速度）。
- 通过其他层报告附加信息，如推理执行时间（参考第 10 节 AI 验证应用程序）

图 38. 目标上验证



提示　　与目标交换的所有消息（包括数据）存储在“ C:\Users\<username>\.stm32cubemx\ai_stm32_msg.log”专门的日志文件中。

7 生成的 STM32 NN 库

仅为每个导入的 DL 模型生成专门的（取决于 DL 模型）C 文件。这些文件的名称前缀是用户提供的网络名称（参考第 4.3 节 上传预训练的 DL 模型文件）。它们基于 *network_runtime.a* 库实现的内部私有 API：

- *<名称>.c* 和 *<名称>.h* 文件用于拓扑
- *<名称>_data.c* 和 *<名称>_data.h* 文件用于权重/偏差

提示 *所有工具链和 STM32 MCU 系列共用所生成的专用数据和网络文件。*

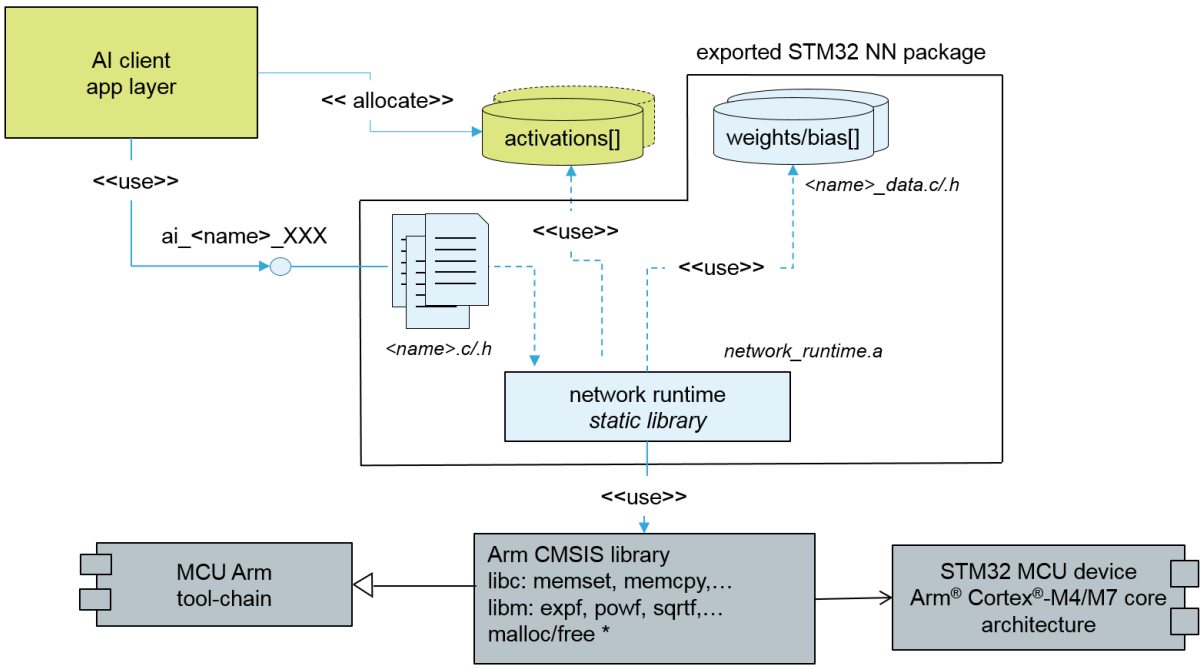
Network_runtime.a 库或 NN 计算内核作为静态库：

- 在链接阶段删除了所有未使用的符号和方法
- 由于不适合存储器资源有限的设备，因此未基于网络图形方法（如 ARM-NN、TensorFlow™精简版部署环境）。

7.1 固件集成

图 39 图 39 列出了所生成的 STM32 NN 包的 MCU 集成模型和视图（包括运行时相关性）。

图 39. MCU 集成模型和视图



对于应用层，导出的 NN 库被视为“黑盒”或自给自足的对象。仅提供专用文件、网络拓扑（*<名称>.c* 和 *<名称>.h* 文件）和权重/偏差参数（*<名称>_data.c* 和 *<名称>_data.h* 文件）作为源文件。这些文件基于公共网络运行时间库（*network_runtime.a*）。与系统运行时间的相关性最小：

- 标准 *libc* 存储器操作功能（`memcpy`、`memset`），通常由 MCU 工具链提供。
- 支持 Cortex®-M 优化操作（FPU 和 DSP 指令）的 CMSIS 库，它是 STM32 HAL 包的一部分。
- `malloc`/`free` 目前预计支持*“循环类型”*层（*“GRU”*和*“LSTM”* 层）。在未来版本中，计划使用静态缓冲区分配方法进行替换。通过众多循环 cell 单元和相关处理时间减轻性能影响。
- 为了支持 `expf`、`powf`、`tanhf` 和 `sqrtf` 函数，还需要一个数学库（支持 DSP/FPU）。
- 需要的栈最小（可使用*“AI 系统性能”*应用程序测量实际值；参考第 9 节 AI 系统性能应用程序）。

提示 *链接最终用户阶段（固件映像生成）必须解决所有外部相关性。*

激活存储器缓冲区可动态分配到堆或者作为全局数组（.bss 和 .data 部分）。请参考 `ai_<name>_init()` 函数，了解如何将权重/偏差缓冲区和激活存储器缓冲区传递到 NN 内核库。

7.2 库源代码树视图

当创建 IDE 项目时，所生成的 NN 对象会导出到子文件夹*<project_name>/Middlewares/ST/AI/*。
还会添加 CMSIS-DSP 库中的 *arm_dot_prod_f32.c* 文件。

```
<项目名称>
    |- Drivers\CMSIS\DSP_Lib\Source\BasicMathFunctions
    |                                      \_ arm_dot_prod_f32.c
    |- Inc
    |    |- bsp_ai.h                     /* STMCubeMX 生成的文件 */
    |    |- constants_ai.h               /* 用于 AI 应用示例 */
    | ...
    |--Middlewares
        \-- ST/AI
            | /*=====================================================================*/
            |-- AI          /* X-CUBE-AI 内核生成的 NN 库 */
            |     |-- data
            |     |     |-- <name_1>_data.c      /* 专用 NN 权重/偏差 */
            |     |     |-- <name_1>_data.h
            |     |     |-- <name_2>_data.c
            |     |     \-- <name_2>_data.h
            |     |-- include
            |     |     |-- <name_1>.h           /* 专用公共/客户端 API */
            |     |     |-- <name_2>.h
            |     |     \- *.h                   /* 通用和私人标头文件 */
            |     |-- lib
            |     |     \-- network_runtime.a    /* 通用运行时间库 */
            |     \-- src
            |         |-- <name_1>.c             /* 专业的 NN 实现*/
            |         \-- <name_2>.c
            | /*=====================================================================*/
            |
            \-- Application
                    |-- SystemPerformance       /* 通用应用示例 */
                        |-- Inc
                        | \-- aiSystemPerformance.h
                        \-- Src
                            \-- aiSystemPerformance.c
```

提示 *除了“<名称>.h”文件，“include”文件夹中的所有标头文件都是私人头文件（“network_ runtime.a” 库）。*

提示 *只有添加并启用插件式 AI 应用程序时，才会生成或复制“bsp_ai.h”, “constants_ai.h”和“SystemPerformance.*”文件。*

IDE 项目文件中的 STM32CubeMX 生成器自动添加以下特定系统构建选项：

```
# 构建 AI 相关文件的子生成文件

# AI 子文件夹位置
NN_LIB ?= ./Middlewares/ST/AI
STM32_HAL_DIR ?= ./Drivers

# 专用 AI 源
SRC += $(NN_LIB)/AI/data/*_data.c
SRC += $(NN_LIB)/AI/src/*.c

# C/C++编译器选项
CPPFLAGS += -I$(NN_LIB)/AI/Include
CPPFLAGS += -I$(NN_LIB)/AI/data

# 编译“arm_dot_prod_f32.c”文件
CPPFLAGS += -DARM_MATH_CM7 -D_FPU_PRESENT=1U
CPPFLAGS += -D$(STM32_HAL_DIR)/CMSIS/Include
SRC += $(STM32_HAL_DIR)/CMSIS/DSP_Lib/Source/BasicMathFunctions/arm_dot_prod_f32.c

# 通用 NN 运行时间库的链接器选项
LDFLAGS += -L$(NN_LIB)/AI/lib -l:network_runtime.a

# 用于 AI 系统性能应用
SRC += $(NN_LIB)/Application/SystemPerformance/Src/aiSystemPerformance.c
CPPFLAGS += -I$(NN_LIB)/Application/SystemPerformance/Inc
```

7.3 特定 AI 平台文件

对于 AI 应用程序，STM32CubeMX 代码生成器会自动生成特定文件。由于 STM32 HAL 层以及相关的明确宏定义，利用这些文件可通过不同的 STM32 硬件平台和设置轻松移植应用程序代码（参考 *aiSystemPerformance.c* 和 *aiSystemPerformance.h* 文件实例）。

```
/* @file - bsp_ai.h - STM32Cube MX 生成的代码 */

...
#include "stm32f7xx.h"
#include "stm32f7xx_hal.h"
#include "app_x-cube-ai.h"
#include "constants_ai.h"
#define UartHandle huart3
#define MX_UARTx_Init MX_USART3_UART_Init
...


/* @file - app_x-cube-ai.c - STM32Cube MX 生成的代码 */

...
#include "bsp_ai.h"
#include "aiXXX.h"
...
void MX_X_CUBE_AI_Init(void)
{
    MX_UARTx_Init();
    aiXXXInit(); /* 初始化应用程序的入口点 */
    /* 用户代码开始 0 */
    /* 用户代码结束 0 */
}

/*
* AI 后台任务
*/
void MX_X_CUBE_AI_Process(void)
{
    aiXXXProcess(); /* 执行应用程序内核的入口点 */
    HAL_Delay(1000); /* 延迟 1s */
    /* 用户代码开始 1 */
    /* 用户代码结束 1 */
}
...
...
```

7.4 多网络推理 API

app_x-cube-ai.c and *app_x-cube-ai.h* 文件还提供 AI 客户端应用程序可以使用的通用多网络推理 API。它与原生嵌入式推理客户端 API 非常相近（参考第 8 节 嵌入式推理客户端 API）；只是 `create()` 函数不同。必须传递网络的 C 名称字符串才能创建底层网络实例。此接口主要供插件式 AI 测试应用程序使用，作为寻址不同嵌入式网络的通用方法。

```
/* @file - app_x-cube-ai.h/.c - GSTM32Cube MX 生成的代码 */
 ...
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* output);
```

7.5 重入和线程安全注意事项

为了保护入口点避免出现并发访问，未实施内部同步机制。如果在多线程环境中使用 API，必须通过应用层本身保证对实例化 NN 的保护。

为了最大限度减少 RAM 的使用，可使用相同的激活存储块（`SizeSHARED`）来支持多个网络。这种情况下，用户必须保证正在进行的推理执行不能被另一网络的执行抢占。

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

提示 *如果预计会由于实时限制或延迟原因而发生抢占，每个网络实例必须具有自己私有的激活缓冲区。*

7.6 代码和数据放置注意事项

对于当前的 STM32 存储器架构（基于 L4/F4/F3 和基于 F7/H7），由于性能因素，没有特定的数据和代码放置需求。Flash ART IP 以及 Arm®内核子系统缓存（基于 Cortex®- M7 的架构）会有效限制存储器延迟的副作用。 NN 代码（*.text* 部分）和 RO 数据（*.rodata* 部分）可放置在内部闪存区。RW 数据（.data.*data* 和 *.bss* 部分）必须放置在嵌入式 SRAM 中。使用客户端栈；它必须放置在零等待状态存储器中。

提示 *激活缓冲区中没有数据保留要求。该缓冲区在实际中可以视为暂存或工作缓冲区。在两次推理之间，可以重复使用缓冲区进行预处理，或者可以在系统进入深度睡眠时关闭相关的存储器设备。*

7.7 调试注意事项

库必须视为二进制格式的优化型黑盒（不提供源文件）。不支持运行时内部数据或状态自省。NN 的映射和端口通过 X-CUBE-AI 生成器保证。一些集成问题可以通过 `ai_<name>_get_error()` 函数高亮显示。

8 嵌入式推理客户端 API

嵌入式推理客户端 API 是*<project_name>/Middlewares/ST/AI/src/<name>.h* 文件的一部分。所有函数和宏均根据所提供的 C 网络名称生成。

8.1 输入和输出 x-D 张量布局

输入和输出缓冲区被定义为具有最多 3 个维度的张量（HWC 布局格式，代表高度、宽度和通道）。为了处理一组张量，可添加批维度。缓冲区完全通过 `struct ai_buffer` C 结构定义来定义。

```
/* @file: ai_platform.h */

typedef struct ai_buffer_ {
    ai_buffer_format    format;     /*!< 缓冲区格式 */
    ai_u16              n_batches;  /*!< 缓冲区中的批数 */
    ai_u16              height;     /*!< 缓冲区高度维度 */
    ai_u16              width;      /*!< 缓冲区宽度维度 */
    ai_u32              channels;   /*!< 缓冲区通道编号 */
    ai_handle           data;       /*!< 指向缓冲区数据的指针 */
} ai_buffer;
```

提示 *目前，输入和输出张量仅支持 `AI_BUFFER_FORMAT_FLOAT` 格式（32 位浮点）。*
这种 C 结构还可用于处理不透明和特定数据缓冲区。

8.1.1 一维张量

对于一维张量，预计采用标准 C 数组类型处理输入和输出张量。

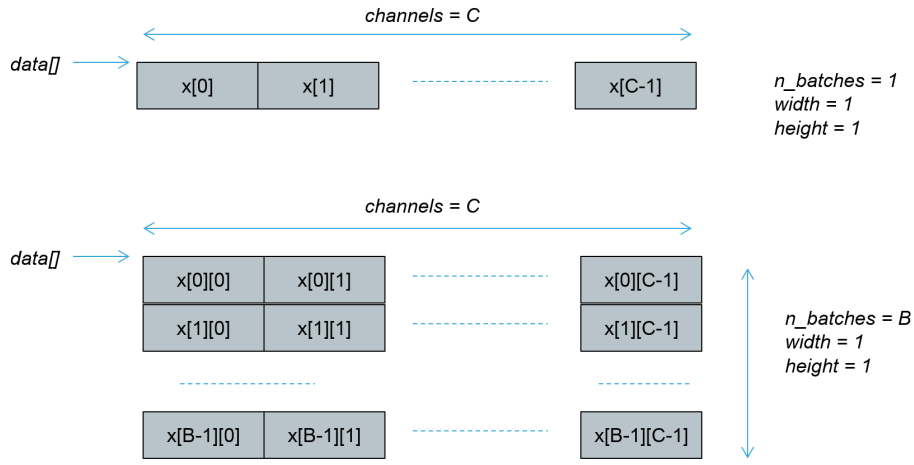
```
#include "network.h"

#define xx_SIZE VAL                 /* = H * W * C = C */

ai_float xx_data[xx_SIZE];          /* n_批次 = 1，高度 = 1，宽度 = 1，通道 = C */

ai_float xx_data[B * xx_SIZE];      /* n_批次 = B，高度 = 1，宽度 = 1，通道 = C */
ai_float xx_data[B][xx_SIZE];
```

图 40. 一维张量数据布局



8.1.2 二维张量

对于二维张量，采用标准二维 C 数组存储器排列来处理输入和输出张量。两个维度映射到原始工具箱表达中的张量的前两个维度：Keras / TensorFlow™中的 H 和 C、Lasagne 中的 H 和 W。

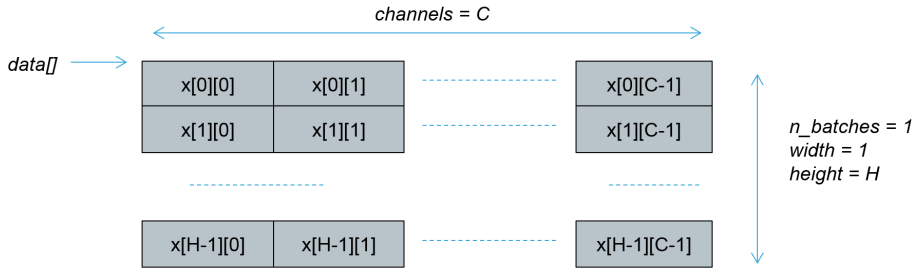
```
#include "network.h"

#define xx_SIZE VAL              /* = H * W * C = H * C */

ai_float xx_data[xx_SIZE];       /* n_批次 = 1，高度 = H，宽度 = 1，通道 = C */
ai_float xx_data[H][C];

ai_float xx_data[B * xx_SIZE];   /* n_批次 = B，高度 = H，宽度 = 1，通道 = C */
ai_float xx_data[B][H][C];
```

图 41. 二维张量数据布局



提示 *如果原始工具箱中的维序与 HWC 不同（如 Lasagne：CHW），用户应负责重新正确排列各元素。*

8.1.3 三维张量

对于三维张量，采用标准三维 C 数组存储器排列来处理输入和输出张量。

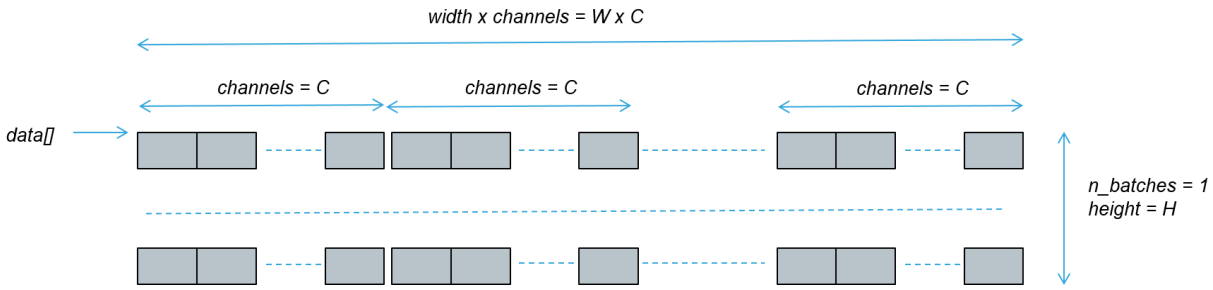
```
#include "network.h"

#define xx_SIZE VAL              /* = H * W * C */

ai_float xx_data[xx_SIZE];       /* n_批次 = 1，高度 = H，宽度 = W，通道 = C */
ai_float xx_data[H][W][C];

ai_float xx_data[B * xx_SIZE];   /* n_批次 = B，高度 = H，宽度 = W，通道 = C */
ai_float xx_data[B][H][W][C];
```

图 42. 三维张量数据布局



8.2 create() / destroy()

```
ai_error ai_<name>_create(ai_handle* network, const ai_buffer* network_config);
ai_handle ai_<name>_destroy(ai_handle network);
```

这份强制函数是 AI 客户端实例化和初始化 NN 实例必须调用的早期函数。 ai_handle 引用必须传递到其他函数的不透明上下文。

- <name>_config 参数是编码为 ai_buffer 的特定网络配置缓冲区，由（AI_<NAME>_DATA_CONFIG C 定义）AI 代码生成器（参见 *<名称>.h* 文件）生成

- 当应用程序不再使用该实例，必须调用 ai_<name>_destroy() 函数，释放所分配的资源

典型使用

```
#include <stdio.h>
#include "network.h"
...
/* 引用实例化 NN 的全局句柄 */
static ai_handle network = AI_HANDLE_NULL;
...
int aiInit(void) {
    ai_error err;
    ...
    err = ai_network_create(&network, AI_NETWORK_DATA_CONFIG);
    if (err.type != AI_ERROR_NONE) {
        /* 管理错误 */
        printf("E: AI error - type=%d code=%d\r\n", err.type, err.code);
        ...
    }
    ...
}
```

8.3 get_error()

```
ai_error ai_<name>_get_error(ai_handle network);
```

该函数可供客户端用于在执行 ai_<name>_xxx() 函数过程中检索报告的第一个错误。请参考 *ai_platform.h* 文件，获得返回的错误类型（ai_error_type）和相关代码（ai_error_code）列表。

典型 AI 错误函数处理程序（用于调试和日志）

```
#include "network.h"
...
void aiLogErr(const ai_error err, const char *fct)
{
    if (fct)
        printf("E: AI error (%s) - type=%d code=%d\r\n", fct, err.type, err.code);
    else
        printf("E: AI error - type=%d code=%d\r\n", err.type, err.code);
}
```

8.4 get_info()

```
ai_bool ai_<name>_get_info(ai_handle network, ai_network_report* report);
```

该可选函数可供客户端用于检索实例化 NN 的信息（用于调试和日志）。

network 句柄必须是有效句柄。请参考 ai_<name>_create() 函数。

典型使用

```
#include "network.h"
...
/* 引用实例化 NN 的全局句柄 */
static ai_handle network = AI_HANDLE_NULL;
...
int aiInit(void) {
    ai_network_report report;
    ai_bool res;
...
    res = ai_network_get_info(network, &report)
    if (res) {
        /* 显示/使用报告数据 */
        ...
    }
...
}
```

8.5 init()

```
ai_bool ai_<name>_init(ai_handle network, const ai_network_params* params);
```

该强制函数必须由客户端用于初始化实例化 NN 的内部运行时结构：

- params 参数是一个结构（ai_network_params 类型），用于传递所生成权重（params 属性）的引用以及激活/崩溃存储器缓冲区（activations 属性）。
- network 句柄必须是有效句柄。请参考 ai_<name>_create() 函数。

```
/* @file: ai_platform.h */
typedef struct ai_network_params_ {
  ai_buffer params;           /*! 参数缓冲区信息（必需！） */
  ai_buffer activations;      /*! 激活缓冲区信息（必需！） */
} ai_network_params;
```

必须使用多个 C 宏助手和特定 AI_<NAME>_XX C 定义初始化此参数：

- params 属性处理权重/偏差存储器缓冲区
- activations 属性处理转发过程使用的激活/崩溃存储器缓冲区， 参考 ai_<name>_run() 函数）
- 相关存储块的大小分别由以下 C 定义（参考文件<*名称*>*_data.h*）来定义。这些缓冲区的存储布局取决于实现的神经网络。
  - AI_<NAME>_DATA_WEIGHTS_SIZE
  - AI_<NAME>_DATA_ACTIVATIONS_SIZE

典型使用

```
#include <stdio.h>
#include "network.h"
#include "network_data.h"
...
/* 引用实例化 NN 的全局句柄 */
static ai_handle network = AI_HANDLE_NULL;

/* 处理激活数据缓冲区的全局缓冲区 - R/W 数据 */
AI_ALIGNED(4)
static ai_u8 activations[AI_NETWORK_DATA_ACTIVATIONS_SIZE];
...

int aiInit(void) {
    ai_error err;
...
    /* 初始化网络 */
    const ai_network_params params = {
        AI_NETWORK_DATA_WEIGHTS(ai_network_data_weights_get()),
        AI_NETWORK_DATA_ACTIVATIONS(activations) };
    if (!ai_network_init(network, &params)) {
        err = ai_network_get_error(network);
        /* 管理错误 */
        ...
        }
...
}
```

8.6 run()

```
ai_i32 ai_<name>_run(ai_handle network, const ai_buffer* input, ai_buffer* output);
```

该函数是馈送 NN 的主函数。输入和输出缓冲区参数（ai_buffer 类型）提供输入张量并存储预测的输出张量（参考第 8.1 节 输入和输出 x-D 张量布局）：

- 返回值是处理的输入张量数（n_batches > 1）。如果返回值是负数或为空，则使用 ai_<name>_get_error() 函数获取错误。
- 必须使用 AI_<NAME>_IN_1（和 AI_<NAME>_OUT_1）初始化输入（和输出）缓冲区句柄
- AI_<NAME>_IN_1_SIZE（和 AI_<NAME>_OUT_1_SIZE）用于初始化输入数据（和输出数据）缓冲区

提示 *可以传递两个单独的输入和输出 ai_buffer 列表。这样即可在未来支持具有多个输入、输出或两者的神经网络。AI_<NAME>_IN_NUM 和 AI_<NAME>_OUT_NUM 分别用于在编译时获取输入和输出的数量。这些数值也由 ai_network_report 结构返回（参考 ai_<name>_get_info() 函数）。*

典型使用

```
#include <stdio.h>
#include "network.h"
...
/* 引用实例化 NN 的全局句柄 */
static ai_handle network = AI_HANDLE_NULL;
...
static ai_buffer ai_input[AI_NETWORK_IN_NUM] = { AI_NETWORK_IN_1 };
static ai_buffer ai_output[AI_NETWORK_OUT_NUM] = { AI_NETWORK_OUT_1 };
...
int aiRun(const ai_float *in_data, ai_float *out_data,
          const ai_u16 batch_size)
{
    ai_i32 nbatch;
    ...
    /* 初始化输入/输出缓冲区处理程序 */
    ai_input[0].n_batches = batch_size;
    ai_input[0].data = AI_HANDLE_PTR(in_data);
    ai_output[0].n_batches = batch_size;
    ai_output[0].data = AI_HANDLE_PTR(out_data);

    nbatch = ai_network_run(network, &ai_input[0], &ai_output[0]);
    if (nbatch != batch_size) {
        err = ai_network_get_error(network);
        /* 管理错误 */
        ...
        }
    ...
}
```

9 AI 系统性能应用程序

AI 系统性能应用程序是一个自助型裸机应用程序，可直接测量生成的 NN 的关键系统集成方面。这里没有并且也无法考察精度性能方面。报告的测量项目包括：

- 推理所用的 CPU 周期（时间：ms，CPU 周期、CPU 工作负载）
- 使用的堆栈（字节）

依次执行以下一系列步骤，运行此应用程序：

1. 打开并配置通过 COM 端口连接的主机串行终端控制台（通常由通过 USB 连接的虚拟 COM 端口支持，如 ST-LINK/V2 功能）。
2. 设定 COM 设置。该设置必须与 STM32 UART 设置一致（参考第 3.2 节 硬件和软件平台设置）：
   - 115200 波特率
   - 8 位
   - 1 个停止位
   - 无奇偶校验
3. 重置开发板，启动应用程序

当应用程序运行时，在控制台中输入 [p/P] 会暂停主循环。该应用程序嵌入一个最小的交互控制台，支持以下命令：

```
交互控制台的可能按键：
    [q,Q]       退出应用程序
    [r,R]       重新启动（NN 取消初始化并重新初始化）
    [p,P]       暂停
    [h,H,?]     此信息
     xx         立即继续
```

9.1 系统运行时信息

图 43 图 43 和图 44 显示了日志的第一部分，包含 STM32 运行时的有用信息或者 Keil®和 Atollic IDE 的执行环境：设备 ID、系统时钟值、使用的工具链及其他。

图 43. 系统运行时信息 - Keil® IDE

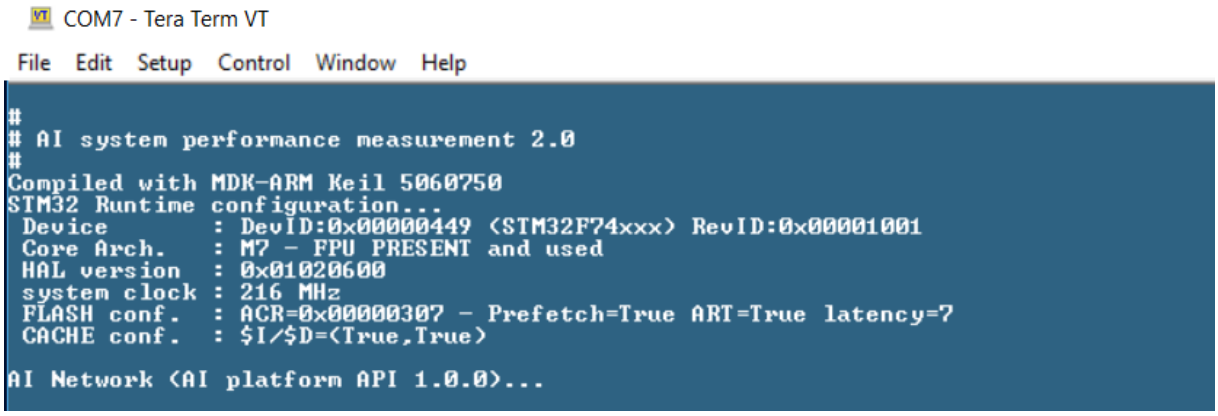
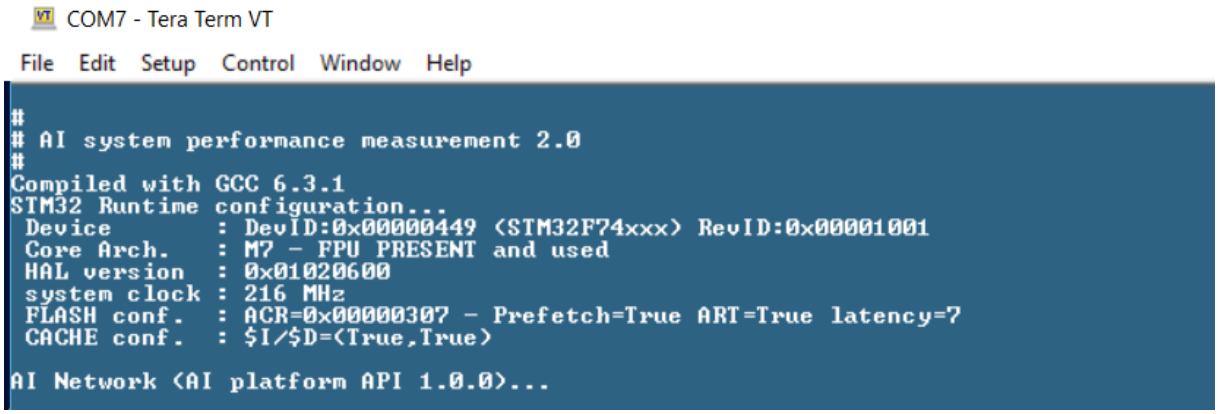


图 44. 系统运行时信息 - Atollic IDE

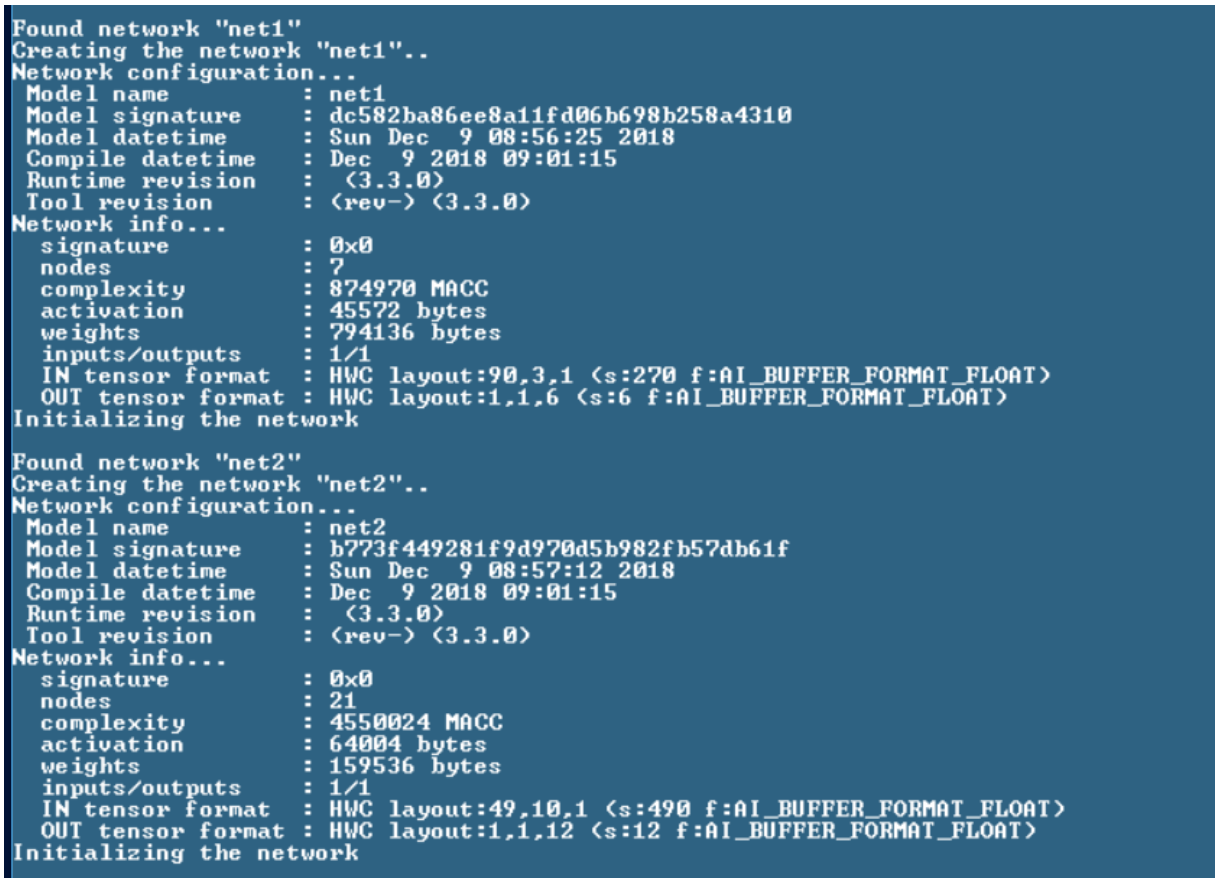


提示 *若要在日志中检索这些信息，可在执行主循环期间在控制台中输入* `r/R`。

9.2 嵌入式 C 模型网络信息

图 45 中所示的这第二部分指示生成的 NN 的主要静态特性。尤其是提供了 RAM/闪存大小（字节，分别在激活/权重字段）和逻辑复杂度（MACC，复杂度字段）。此外还报告了输入和输出张量的形状定义。这些信息还可通过 `ai_<name>_get_info()` 客户端 API 函数利用客户端应用程序代码获得。

图 45. C 模型网络信息



提示 *若要在日志中检索这些信息，可在执行主循环期间在控制台中输入* `r/R`。

9.3 嵌入式 C 模型运行时性能

如图 46 和图 47 所示，日志的最后一部分（主循环）报告测得的直接可用的系统性能。随机注入到网络中，从而通过推理（CPU cycles）测量 CPU 周期数。CPU workload 和 cycles/MACC 会从此数值中减掉。测量期间，IRQ 会被屏蔽。

- duration 是指一个推理所需的时间（ms）。
- CPU cycles 是指一个推理所需的 CPU 周期数。
- CPU workload 对应 1 s 内相关的 CPU 工作负载。
- cycles/MACC 是 MACC 操作所需的 CPU 周期数。

图 46. C 模型运行时性能

```
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
```

图 47. 堆栈检查时的 C 模型运行时性能

```
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
```

提示 *“used heap”是指执行所有推理期间所需的 malloc() 数量和累积分配大小（分别 free()）。为了检测假想的内存泄漏，两次推理或测试迭代之间不会重置计数器。目前情况下，最小堆大小为 29568 / #iter = ~2 KB.*

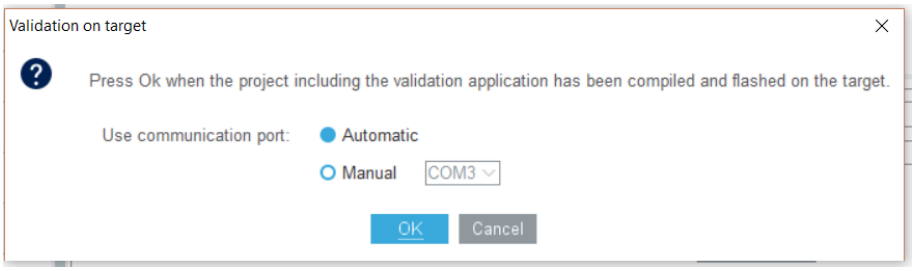
Caution: 目前，在基于 GCC 的环境下仅支持堆监控。

10 AI 验证应用程序

*AI 验证*应用程序是一个自助型裸机应用程序，支持第 6.2 节 验证引擎中所述的设备上验证。它提供一个灵活、完整、基于 UART 的接口，使主机能够导出完整的推理 API。该应用程序必须通过 X-CUBE-AI UI 使用。

1. 当烧录开发板时，重置或重启该应用程序。
2. 重新打开生成固件的 *ioc* 文件。
3. 进入 AI 配置面板并打开必须验证的 [<网络名称>] 选项卡。
4. 单击 [目标上验证] 按钮，开始验证过程。单击 [确定] 按钮前，用户应负责指出所使用的主机 COM 端口，如图 48 所示。否则，会发现所有可用的 COM 端口均检测有效连接的 STM32 开发板（使用发现的第一个开发板）。

图 48. 设备上验证的主机 COM 端口选择器

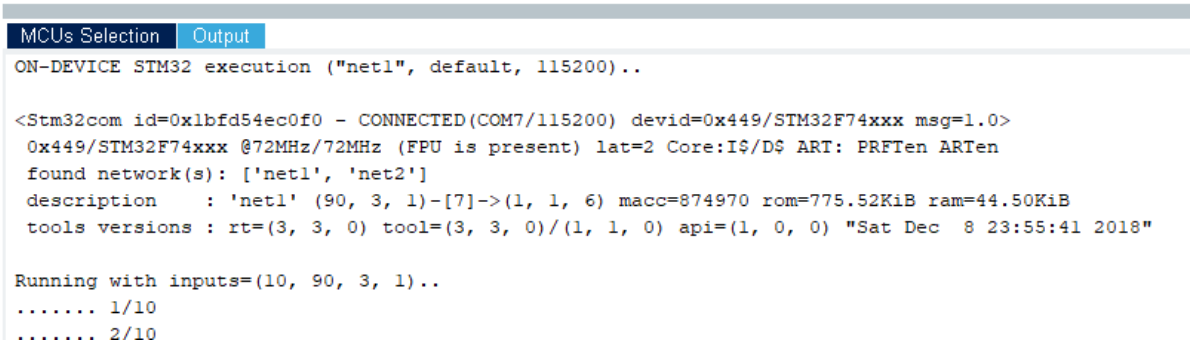


关于 [桌面上验证]，最终结果报告在 [验证状态] 字段中。更多详细信息报告在 UI 日志控制台中（[输出] 窗口）。

10.1 系统运行时信息

图 49 所示报告日志的第一部分显示了主要系统信息：设备 ID、时钟频率、存储器子系统配置、嵌入式网络列表。对于经过验证的网络，还提供形状输入和形状输出张量描述以及使用的 AI 工具版本。

图 49. 系统运行时信息



10.2 嵌入式 C 模型运行时性能

图 50 所示日志的第二部分报告可直接使用的系统性能测量结果（`duration` 推理的平均执行时间）。`cycles/MACC` 会从 `duration` 值中减掉。测量期间，IRQ 会被屏蔽。

- `duration` 是指一个推理所需的时间（ms）。
- `CPU cycles` 是指一个推理所需的 CPU 周期数。
- `cycles/MACC` 是 MACC 操作所需的 CPU 周期数。

图 50. C 模型运行时性能

```
....... 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 逐层运行时性能

图 51 所示日志的下一部分提供了有关生成的 C 模型的附加信息：实现的 C 层的名称和类型（Clayer / id / desc）、输出形态（oshape）以及推理的平均执行时间（ms）。

图 51. 逐层结果 - 目标上验证

```
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 目标上验证的最终结果

图 52 所示日志的最后一部分提供验证过程的最终结果。它与桌面上验证的结果相似，但是仅报告最后一层的 L2 误差（参考第 6.2 节 验证引擎）。

图 52. 目标上验证的最终报告

```
  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 连接期间返回的错误

仅支持以下 UART 设置：

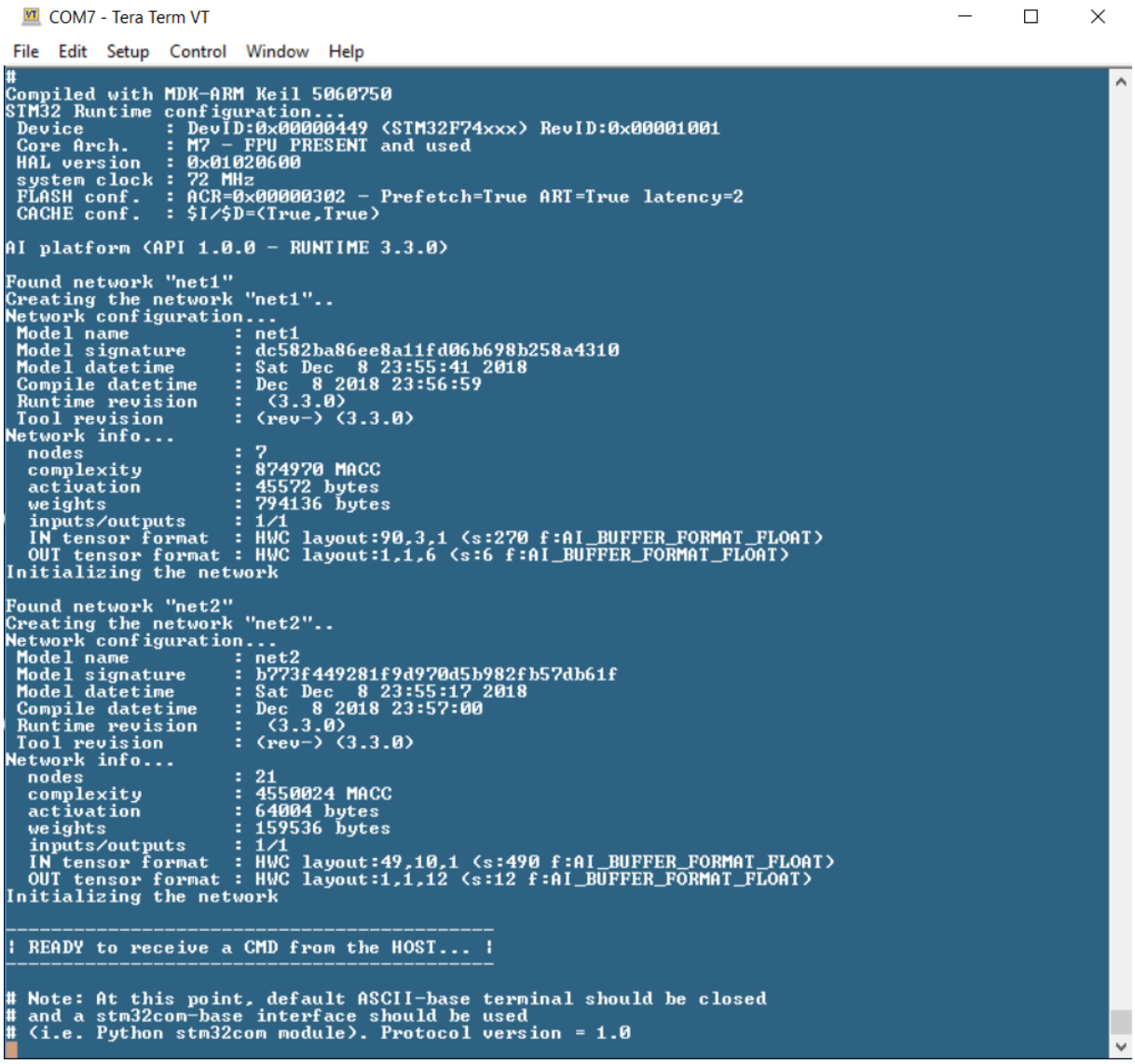
- 8 位
- 1 个停止位
- 无奇偶校验
- 115200 波特率（默认值）。
  如果已经重新定义，则使用新值（参考第 3.2 节 硬件和软件平台设置）。

10.5.1 错误：没有连接开发板、固件无效或者需要重启开发板

表示未连接或者找不到开发板、或者固件不是预期的*"AI 验证"*固件。该错误还表示固件状态不一致，这种情况下，必须重启开发板。

为了检查固件是否正确烧录，在启动时打开主机串行终端控制台，这会生成一个基于 ASCII 的日志。不要忘了在再次启动 [目标上验证] 过程前关闭连接。

图 53. AI 有效 - 初始日志



10.5.2 错误：network_name 不是一个有效网络

表示在连接的开发板中没有通过名称识别的预期 C 模型。有关更多详细信息，请参见 UI 日志控制台（[输出] 窗口）。

10.5.3 错误：嵌入式 STM32 模型与 C 模型不匹配

表示生成的 C 模型的签名与预期模型不一致。用于检查签名的参数如下：

- RAM/ROM 大小
- MACC
- 结点数
- 工具版本

有关更多详细信息，请参见 UI 日志控制台（[输出]窗口）。

11 AI 模板应用程序

选定时，生成的 IDE 项目并不是真正完整的 AI 模板应用程序。*aiSystemPerformance.h* 和 *aiSystemPerformance.c* 文件清楚地说明了此目的。仅生成特定文件（参考第 7.3 节 特定 AI 平台文件），以提供支持多网络 API 并具有通用 UART 句柄的基本框架。这可以用作通过两个简单的入口点（`init` 和 `process` 函数）开发初始裸机应用程序的起点。

12 支持的深度学习工具箱和层 深度学习

X-CUBE-AI 内核目前支持以下 DL 工具箱：

- Keras: //keras.io/
- Lasagne: //lasagne.readthedocs.io/en/latest/
- Caffe: //caffe.berkeleyvision.org/
- ConvNetJs: //cs.stanford.edu/people/karpathy/convnetjs/

对于每个工具箱，根据网络 C API 的表达能力以及特定工具箱的解析器，仅支持所有可能层和层参数的一个子集。以下章节详细介绍每个工具箱的当前版本工具在所支持的各层（使用工具箱命名惯例）以及每层支持的属性（参数）方面支持的配置。同时列出多个层支持的相同功能。

12.1 Keras

在 Keras 中，X-CUBE-AI 利用通道最后维度排序支持 TensorFlow™后端。 Keras 支持 Keras 2.0 至 2.2.4 版，但是 Keras 1.x 中定义的网络未获得正式支持。

可采用两种不同的方式加载模型：

- 通过包含模型和权重（*.h5*/*.hdf5*）的单个文件
- 通过单独文件中的模型配置和权重。这种情况下，从 HDF5 文件（*.h5*、*.hdf5*）加载权重并从文本文件 JSON（*.json*）或 YAML（*.yml*、*.yaml*）加载模型配置。

支持的各层和属性如下：

- Conv2D, Conv1D：卷积层；通过在 x 上添加一行或一列的维度实现一维卷积。支持的属性如下：
  - *padding*：*“SAME”*和*“VALID”*策略
  - *kernel_size*：任意过滤器卷积核大小，只要小于输入大小
  - *stride*：任意步幅，只要小于输入大小
  - *filters*：输出通道数
  - *use_bias*：如果设为*“假”*，则不使用偏差
- DepthwiseConv2D：深度卷积层；支持与二维卷积（*Conv2D*）相同的属性，附加属性如下：
  - *depth_multiplier*：每个特征组的通道数；自动推测输出组和通道的数量。
- SeparableConv2D：联合没有偏差的深度卷积以及 1x1 卷积。支持 *DepthwiseConv2D* 和 *Conv2D* 属性。
- MaxPooling1D、MaxPooling2D：最大池化层；通过在 x 上添加一行或一列的维度支持一维版本。支持的属性如下：
  - *padding*：*“SAME”*和*“VALID”*策略
  - *pool_size*：任意池大小，只要小于输入大小
  - *strides*：任意步幅，只要小于输入大小
- AveragePooling1D、AveragePooling2D：平均池化层；属性与最大池化层相同。
- GlobalMaxPooling1D、GlobalMaxPooling2D、GlobalAveragePooling1D、GlobalAveragePooling2D：全局最大和平均池化层；通过将 *pool_size* 和 *strides* 设置为层的输入大小提供支持。
- Dense：稠密（全连接）层。支持的属性如下：
  - *units*：输出特征数
  - *use_bias*：如果设为*“假”*，则不使用偏差
- Activation：非线性激活层，还在作为 *Conv2D*、*DepthwiseConv2D*、*SeparableConv2D* 或 *Dense* 的一部分时解码。支持的属性如下：
  - *nonlinearity*：非线性激活的类型；支持以下函数：`linear`、`relu`、`relu6`、`softmax`、`tanh`、`sigmoid` 和 `hard_sigmoid`
- BatchNormalization：批归一化层。支持的属性如下：
  - *axis*：执行归一化的输入维度。仅支持最后轴（通道）的归一化。
- Permute：维度转置层。仅支持三维张量。支持的属性如下：
  - *dims*：规定输入维序的目标维序，仅适用于非批维度。
- Flatten：将输入非批输入维度转换为一个向量；映射为特殊*重塑*层。

- Reshape：改变输入张量的形状，但不改变元素数量。支持的属性如下：
  - *shape*：目标形状。还支持整数（-1）和保持相同（0）值。
- ZeroPadding1D、ZeroPadding2D：扩充层，仅支持作为后续 *Conv1D* or *Conv2D* 层的一部分。支持的属性如下：
  - *padding*：扩充值列表，支持对称和非对称扩充
- LSTM：长短期记忆层。支持的属性如下：
  - *units*：cell/隐藏单元数
  - *activation*：隐藏层到输出层激活，支持*激活*值
  - *recurrent_activation*：隐藏层到隐藏层激活；支持以下函数：`linear`、`relu`、`tanh`、`sigmoid` 和 `hard_sigmoid`
  - *return_sequences*：如果设为“*真*”，则返回所有隐藏状态，而不是仅返回最后一个
  - *use_bias*：如果设为“*假*”，则不使用偏差
- GRU：门控循环单元层；支持所有 *LSTM* 属性并额外支持：
  - *reset_after*：仅 *GRU*；使用略微不同的公式来符合 *CuDNN* 实现。仅适用于 Keras 2.1.0 和更高版本。
- Input：用于网络输入的可选占位符，转换时丢弃
- Dropout：纯训练层，转换时忽略。

12.2 Lasagne

Lasagne 基于 Theano 计算库。它是用于深度学习的首批用户友好型工具箱之一。Theano 提供使用周期结束支持，但是由于历史原因，支持 Lasagne，因为几个项目都使用它。仅支持 GitHub 的最新版本（*0.2-dev*），不支持来自 *pip* 的稳定 *0.1* 版。

12.2.1 模型格式

Lasagne 没有用于网络结构的文件格式，但是可以使用 NUMPY（*.npz*）文件保存和加载权重。对于网络结构，该工具期望一个构建网络的专门 Python™模块。

该模块必须具有以下元素：

- INPUT：声明为 Theano 张量的网络输入。
- INPUT_SHAPE：输入张量的形状；第一个维度视为批大小并被忽略。
- network：构建网络的函数；该单个输入参数必须与以上声明的 *INPUT* 变量一致。

如果未定义元素或者 NUMPY 数组中的权重与模型不一致，则加载会失败。

模块示例如下：

```
INPUT = T.tensor4('inputs')
INPUT_SHAPE = (None, 1, 60, 63)

def network(input_var):
    net = InputLayer(input_var=input_var, ..)
    net = Conv2DLayer(net, ..)

    return net
```

12.2.2 层支持

支持的各层和属性如下：

- Conv2DLayer：卷积层，仅支持二维卷积。支持的属性如下：
  - *pad*：仅支持明确（整数列表）扩充
  - *filter_size*：任意过滤器卷积核大小，只要小于输入大小
  - *stride*：任意步幅，只要小于输入大小
  - *num_filters*：输出通道数
  - *use_bias*：如果设为“*假*”，则不使用偏差

- MaxPool2DLayer：最大池化层。仅支持二维版。支持的属性如下：
  - *pad*：仅支持明确（整数列表）扩充
  - *filter_size*：任意过滤器卷积核大小，只要小于输入大小
  - *stride*：任意步幅，只要小于输入大小
  - *ignore_border*：如果设为“*真*”，则在计算输出时忽略部分池化区域
- AvgPool2DLayer：平均池化层，属性与最大池化层相同。
- GlobalPoolLayer：通过设置 *pool_size* 和 *strides* 输入层大小支持的全局最大和平均池化层。
- Dense：稠密（全连接）层。支持的属性如下：
  - *num_units*：输出功能数
  - *use_bias*：如果设为“*假*”，则不使用偏差
- NonlinearityLayer：非线性激活层，还在作为 Conv2D 和 Dense 的一部分时解码。支持的属性如下：
  - *nonlinearity*：非线性激活类型；支持以下函数：*identity*、*linear*、*rectify*、*softmax*、*tanh* 和 *sigmoid*.
- BatchNormLayer：批归一化层。支持的属性如下：
  - *axes*：执行归一化的输入维度。仅支持通道轴归一化。
- LocalResponseNormalization2DLayer：通道内的本地响应归一化层。支持的属性如下：
  - *alpha*, *k*, *beta*, *n*：归一化等式的参数
- ReshapeLayer：改变输入张量的形状，但不改变元素数量。支持的属性如下：
  - *Shape*：目标形状。还支持整数（-1）值。
- DimshuffleLayer：维度转置层；支持的属性如下：
  - *pattern*：规定输入维序的目标维序，仅适用于非批处理维度。
- InputLayer：用于网络输入的可选占位符，转换时丢弃
- DropoutLayer：纯训练层，转换时忽略
-

12.3 Caffe

X-CUBE-AI 仅支持具有以下层的 1.0 官方版 Caffe：

- Convolution：卷积层。支持的属性如下：
  - *pad*：任意扩充值，只要小于过滤器卷积核大小
  - *kernel_size*：任意过滤器卷积核大小，只要小于输入大小
  - *stride*：任意步幅，只要小于输入大小
  - *num_output*：输出通道数
  - *group*：输出中的功能组数（如 AlexNet 架构）
  - *bias_term*：如果设为“*假*”，切勿使用偏差
- Pooling：池化层。支持的属性如下：
  - *pool_size*：任意池大小，只要小于输入大小
  - *stride*：任意步幅，只要小于输入大小
  - *pad*：任意扩充值，只要小于池化卷积核大小
  - *pool_function*：用于池化的函数。支持 MAX（最大池化）和 AVE（平均池化）。
  - *Global_pooling*：如果设为“*真*”，则使用全局池化；通过将 *kernel_size* 和 *stride* 设置为层的输入大小实现。
- InnerProduct：稠密（内积）层。支持的属性如下：
  - *num_units*：输出功能数
  - *use_bias*：如果设为“*假*”，则不使用偏差
- ReLU：ReLU 非线性激活层。支持的属性如下：
  - *negative_slope*：负斜率 *x* 如果用作泄漏 ReLU
- Softmax：softmax 非线性激活层。支持的属性如下：
  - *axis*：计算 softmax 的输入维度。仅支持通道轴归一化。

- Scale：维度级扩展层，可用作纯推理批归一化层。支持的属性如下：
  - *axes*：要扩展的维度集
  - *use_bias*：如果设为“*真*”，还使用偏差
- LRN：本地响应归一化层，仅支持通道内的归一化。支持的属性如下：
  - *alpha*, *k*, *beta*, *local_size*：归一化等式的参数
  - *region*：归一化区域的类型。仅支持“*跨通道*”。
- Flatten：将输入维度转换为一个向量；映射为特殊*重塑*层。支持的属性如下：
  - *axis*, *end_axis*：待转换维度的范围
- Reshape：改变输入张量的形状，但不改变元素数量。支持的属性如下：
  - *Shape*：目标形状。还支持保持相同（无）值。
- Input：用于网络输入的可选占位符，转换时丢弃
- Dropout：纯训练层，转换时忽略

12.4 ConvNetJs

ConvNetJs 是写入到 JavaScript 中的 DL 工具箱。以下层仅支持 *npm* 0.3 版：

- conv：二维卷积层。支持的属性如下：
  - *sx*, *sy*：过滤器大小；支持任意值，只要小于输入大小。
  - *stride*：任意步幅，只要小于输入大小
  - *pad*：任意扩充值，只要小于过滤器大小
- pool：最大池化层。支持的属性如下：
  - *sx*, *sy*：池化大小；支持任意值，只要小于输入大小。
  - *stride*：任意步幅，只要小于输入大小
  - *pad*：任意扩充值，只要小于过滤器大小
- fc：稠密（全连接）层；支持可选偏差。还支持以下属性：
  - *out_depth*：输出特征数
- relu、softmax、sigmoid、tanh：非线性激活层；*softmax* 仅适用于通道维度。
- input：用于网络输入的可选占位符，转换时丢弃

13 错误处理

X-CUBE-AI 内核处理各种不同的错误并将其报告给用户：

- 加载错误：传递至工具的文件缺失或不可读
- 无效模型：网络模型文件损坏或者无法解析
- 无效选项：传递至命令行接口的选项无效
- 未实现：缺少未来版本中可能引入的功能
- 工具错误：工具环境配置错误产生的错误
- 内部错误：意外错误，可能是漏洞

13.1 加载错误

如果包含网络模型描述的文件不存在或者无效（例如：目录代替文件），就会产生此错误。此外，还处理以下情况：

- *“Unable to build the network library.”*：由于编译错误导致验证失败；所以，编译的 C 库不可用。

13.2 无效模型

如果不能正确加载 NN 模型文件，就会产生此错误。根据所选的 DL 工具箱，会报告不同的消息。

13.2.1 Lasagne

- *“Couldn’t load Lasagne model , error: ”*：加载包含模型代码的 Python™模块时出错；消息中包含详细错误信息。
- *“Wrong parameters: , error: ”*：传递的参数与模型不兼容；消息提供了准确原因。

13.2.2 Keras

- *“Model saved with Keras “V1” but <= “V2” is supported”*：使用新版 Keras 保存模型，可能未正确解析；因此停止加载。
- *“Model configuration is not a text file: ”*, *“Unknown format for model configuration file: ”*, *“Model file is not a HDF5 file: ”*：为模型配置传递错误的文件类型、权重或两者。
- *“Couldn’t load Keras model | model configuration | weight , error: ”*：模型文件解析错误；消息提供了更多详细信息，如形状不匹配信息。

13.2.3 Caffe

- *“Not a Caffe model file: ”*：网络结构文件不是一个有效的*.prototxt* 文件。
- *“Invalid file or using the wrong Caffe version: , error: ”*：解析*.prototxt* 文件时出错，因为语法无效或者未识别出属性，例如，由于 Caffe 版本不同或更高。

13.2.4 ConvNetJs

- *“ConvNetJs model is not a valid JSON file , error: ”*：传递的文件未被识别为有效的 JSON 文件并且无法加载。
- *“Invalid ConvNetJs model: ”*：模型没有预期用于 ConvNetJs NN 的结构。

13.3 无效选项

如果将错误或没有意义的选项传递到命令行接口，就会产生此错误。

- *“unrecognized network type: ”*：这种待转换的网络类型不在工具支持的类型范围内。

13.4 未实现

该工具目前仅支持 DL 工具箱中所有功能的一个子集。如果遇到暂不支持的功能，就会返回此错误：

- *“Unsupported layer type: ”*：目前不支持层类型与工具箱的组合，无法解析该层。
- *“Unsupported wrapper: ”*：不支持当前层包装器（元层）。
- *“TimeDistributed non supported on shape ”*：只有包含矢量输出的层才支持 *TimeDistributed* 包装器，否则就会产生此错误。
- *“scale / bias are supported only on the input channel; used”*：扩展和偏差层应用到目前不处理的任意维度。
- *“Unsupported cOde generation for layer ”*：暂不支持生成层的 C 代码。
- *“unsupported nonlinearity: ”*：暂不识别或支持这种非线性类型。
- *“unsupported pool function: ”*：暂不识别或支持这种池化函数类型；目前仅处理 *average* 和 *max*。
- *“unsupported pool padding strategy: ”*：不同的工具箱采用不同的扩充方法进行池化；不支持当前方法。
- *“Unsupported LRN region type: ”*：对于本地响应归一化，仅支持跨通道归一化；空间归一化就会产生此错误。
- *“RNN nonlinearity not supported: ”*：循环神经网络内不支持这种非线性类型。
- *“only 4b or 8b elements supported for compression, found ”*：仅处理 4 位或 8 位元素的压缩；其他元素大小就会产生此错误。
- *“Network type cannot be validated: ”*：此工具不能自动验证这种网络类型。

13.4.1 Keras

- *“Unsupported Keras backend: ”*：目前仅支持 TensorFlow™ 后端；对于使用其他后端保存的模型，就会返回此错误。
- *“Pad type not supported: ”*：Keras 中不支持这种扩充策略类型。
- *“RNN ‘go_backwards | return_state’ argument not supported”*：循环层中仍不支持这两个选项。

13.4.2 Caffe

- *“Running statistics in Caffe BatchNorm are not supported.”*：该层正在使用执行期间在线计算的批统计信息，但是暂不支持。

13.5 工具错误

如果不符合对系统设置或结果的假设，就会产生此错误。例如，如果相关性缺失，或者在转换 NN 期间遇到意外状况，就会发生此错误。如果用户环境配置正确，这种错误可能代表存在漏洞。

13.6 内部错误

当出现意外或异常状况时，就会产生此错误，这种情况很可能是存在漏洞的结果。工具仍捕捉错误并将其报告给用户。

14 FAQ

14.1 日志文件用于调试?

当执行“*目标上验证*”过程时，与目标交换的所有消息（包括数据）都会存储在专门的日志文件中：

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

如果验证或生成过程失败，附加调试/日志信息也会存储在文件中：

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

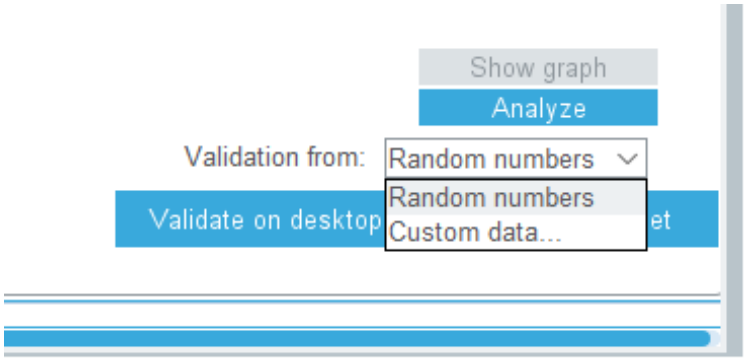
14.2 定制数据集文件格式?

对于验证过程（参考第 6.2 节 验证引擎），可以提供定制数据集代替随机输入数据（两种情况下都会计算 L2 相对误差）。预期文件是一个 *csv* 格式的简单文本文件，每行一个扁平化输入张量。采用逗号作为分隔符。破折号代表注释的开头。跳过空行。

```
# 这是注释行
-1.0760075, 6.2789803, 3.9499009, ... , 9.112013, 0.08172209
-1.920469, 7.8589406, 1.3075534, ... , 14.818938, -1.8387469
1.719910651445388794e-03,1.286244078073650599e-04, ... ,9.708247184753417969e-01
```

提示 *采用 `numpy.genfromtxt()` Python™ 函数上传数据。*

图 54. 定制数据选择器



14.3 多网络限制?

除了 RAM 和闪存可用性外，使用多个网络没有实际限制。每个网络都有自己的 `ai_<name>_XXX()` 推理函数集合（参考第 8 节 嵌入式推理客户端 API）。 section). 如果在多个网络之间共享激活存储块，必须注意。不允许抢占（参考第 7.5 节 重入和线程安全注意事项）。

14.4 无法编译文件 *arm_dot_prod_f32.c*

当重新生成 IDE 项目文件时，*arm_dot_prod_f32.c* 的编译可能会失败：

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

根据目标 STM32 设备，必须在项目设置中重新定义以下 C 定义：

```
ARM_MATH_CM7,__FPU_PRESENT=1
```

14.5 使用的堆或栈：已禁用或暂不支持

该日志指示所使用的工具链中已明确禁用或者尚未实现栈（和堆）监控器。

表 3. 支持堆栈监控

工具链	栈监控器	堆监控器	注释
GCC	支持	支持	SW4STM32
IAR™ 8.X 和 7.x	支持	未实现	-
MDK-ARM	未实现	未实现	-

堆栈监控激活示例：

```
/* @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
...
```

14.6 为什么*“used heap”*始终为零？

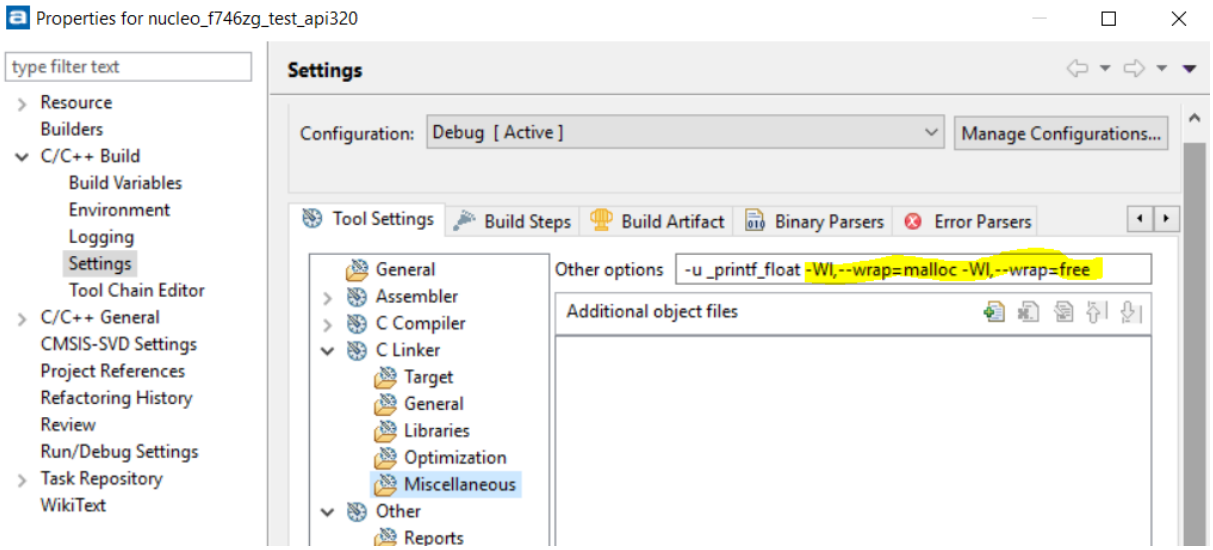
仅适用于基于 GCC 的项目：

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

这种结果不一定代表有问题。大多数 *network_runtime.a* 库都基于预分配的 R/W 缓冲区方案（激活缓冲区）。对于一些特定层（循环层类型），当前实现方案会请求通过 `malloc()` 函数动态分配这些工作缓冲区的一部分。

堆监控器基于工具链特定机制，允许包装系统 `malloc()` 和 `free()` 函数。要启用这种包装，必须在构建系统中设置 `-Wl,--wrap=malloc -Wl,--wrap=free` 其他链接器选项，如图 55 所示。

图 55. 启用堆监控器的链接器选项



14.7 基于 GCC 的项目中的格式化浮点数为空

要输出格式化浮点数，必须添加以下链接选项：

```
-u _printf_float
```

14.8 CPU 周期/MACC

请参见第 4.4.1 节 CPU 周期/MACC 和第 9 节 AI 系统性能应用程序。

14.9 是否有必要启用或配置 TIMER IP？

没有必要。通过推理测量 CPU 周期数的机制利用专门的 Arm® Cortex®-M 调试单元（DWT：数据观察点和跟踪单元），所有支持的 STM32 设备上均配有该单元。它采用由 CPU 时钟（HCLK 系统时钟）提供时钟的自由运行计数器。

14.10 如何在我生成的项目中只更新导出的 NN 库？

如果使用 STM32CubeMX 设计指南（/* USER CODE BEGIN*/, /*... END*/标记）更改导出和生成的文件，则十分简单。可直接重新打开<*项目名称*>.*ioc* 文件，上传新 NN 模型并更新 IDE 项目。

14.11 是否可以导出 NN 库用于非基于 STM32CubeMX 的项目？

答案是双重的：

- 由于当前集成的 STM32 NN 映射工具面向 UI，暂不支持 CLI（参考第 14.12 节 ），所以没有可用的内置库导出机制。
- 由于导出的 NN 库位于明确定义和自带目录的子文件夹中（参考第 7 节 生成的 STM32 NN 库），所以该 AI 子文件夹可完全复制到目标项目的源代码树中：
  1. 为用户的 STM32 MCU 设备创建一个新的虚拟 STM32CubeMX 项目。
  2. 生成用于用户工具链/IDE 的 IDE 项目。这一步要求添加正确的 *network_runtime.a* 库，该库取决于工具链和 Arm®-Cortex®-M 架构。
  3. 将生成的 AI 子文件夹复制到新项目源代码树中。
  4. 将 *network.c* and *network_data.c* f 文件和 *network_runtime.a* 库添加到系统，构建并更新 C/C++编译器和链接器选项，如第 7 节 生成的 STM32 NN 库所述。
  5. 返回到第 1 步，更新并评估修改的 NN 模型。

14.12 命令行接口？

当前版本 X-CUBE-AI 未考虑 CLI。

15 参考

表 4. 参考

ID	说明	链接
[1]	NUCLEO-F746ZG 开发套件	www.st.com/en/product/nucleo-f746zg
[2]	适用于 STM32 v9.0.1 的 Atollic TrueSTUDIO®	atollic.com/truestudio
[3]	IAR Embedded Workbench™ IDE - ARM v8.x 或 v7.x	www.iar.com/iar-embedded-workbench
[4]	µVision® V5.25.2.0 - Keil® MDK-ARM 专业版	www.keil.com
[5]	STM32CubeMX - 初始化代码生成器	www.st.com/en/product/stm32cubemx
[6]	System Workbench for STM32（SW4STM32）	www.st.com/en/product/sw4stm32

版本历史

表 5. 文档版本历史

日期	版本	变更
2019 年 1 月 15 日	1	初始版本。

目录

表一览

图一览