Application Note

*在边缘人工智能应用中使用 **MSPM0G5187***

Sal Ye

摘要

本应用手册介绍了在 MSPM0G5187 微控制器上实现边缘人工智能 (AI) 应用的方法。该微控制器集成了 TinyEngine™ 神经处理单元 (NPU)，这是一种专为资源受限嵌入式系统中高效神经网络推理而设计的专用硬件加速器。本文档首先概述 MSPM0G5187 硬件架构和 NPU 的关键特征，随后全面介绍 TI 的边缘 AI 软件生态系统，包括 CCSTUDIO™ Edge AI Studio、Tiny ML Tensorlab 和神经网络编译器 (NNC)。接着，通过两个代表性应用示例展示 MSPM0G5187 的边缘 AI 功能：基于 LeNet-5 CNN 模型的手写数字识别设计，以及基于时间序列 CNN 模型的波形分类器设计。文中提供了基于 NPU 与基于 CPU 推理实现的量化性能基准对比，充分展示了 NPU 硬件加速在推理延迟和能效方面的显著优势。这些结果为在嵌入式平台上进行边缘 AI 设计的工程师提供了实用指导和设计参考。

内容

1 简介....3
2 带 TinyEngine™ NPU 的 MSPM0G5187....4
3 边缘 AI 工具链....5
3.1 TI Edge AI Studio....5
3.2 TI Tiny ML Tensorlab....7
3.3 TI 神经网络编译器....7
4 通过 TI Edge AI Studio 进行快速原型设计....9
4.1 工程设置....9
4.2 数据采集....11
4.3 数据导入....13
4.4 数据注释....16
4.5 模型训练....17
4.6 性能报告....20
4.7 模型导出....23
5 通过 TI Tiny ML Tensorlab 进行高级开发....25
5.1 环境设置....25
5.2 数据采集....26
5.3 模型架构迭代重构....28
5.4 模型导出....31
5.5 量化感知训练....32
5.6 模型编译....33
6 边缘 AI 应用：数字识别....35
6.1 LeNet-5 变体 CNN 模型....35
6.2 NPU/CPU 性能比较....36
7 边缘 AI 应用：波形分类器....37
7.1 特征提取....38
7.2 时序分类模型....40
7.3 模型内存注意事项....41
7.4 NPU/CPU 性能比较....42
8 总结....43
9 参考资料....43
10 修订历史记录....43

English Document: SDAA429
Copyright © 2026 Texas Instruments Incorporated

商标

TinyEngine™, CCSTUDIO™, C7™, Code Composer Studio™, EnergyTrace™, and E2E™ are trademarks of Texas Instruments.
Arm® and Cortex® are registered trademarks of Arm Limited.
PyTorch® is a registered trademark of Linux Foundation.
Linux® is a registered trademark of Linus Torvalds.
Windows® is a registered trademark of Microsoft Corporation.
所有商标均为其各自所有者的财产。

Copyright © 2026 Texas Instruments Incorporated

1 简介

边缘人工智能 (AI) 涉及在本地、数据源附近或数据源处执行推理模型。与云端 AI 相比，边缘 AI 具有显著优势，包括：通过降低延迟提升响应速度；通过减少带宽需求和降低云依赖提高运营效率；通过网络中断期间仍能持续运行增强系统可靠性；以及通过最小化数据传输和强化隐私保护来加强数据安全性。

MSPM0G5187 搭配 TinyEngine NPU，已助力将 AI 功能直接集成到消费电子、智能家居器件、工业及汽车系统等多个领域的终端器件中。专用的硬件 NPU 并行运行，使得 MCU 的推理延迟比无硬件加速的同类 MCU 最高可降低 90 倍，每次推理的能耗最高可降低 120 倍。

然而，边缘 AI 不仅关乎硬件，它需要一个完整的生态系统，才能真正加速用户的开发与部署。从通用分类、回归到预测任务，TI 全面的软件生态系统使工程师能够以极低的复杂度快速进行 AI 解决方案的原型设计、优化和部署。

- **TI Edge AI Studio**：提供直观的图形化环境，支持模型开发、训练、编译和部署
- **TI Tiny ML Tensorlab**：通过命令行工具提供覆盖模型训练、量化、编译和部署的端到端工作流
- **TI Neural Network Compiler**：通过简单配置即可无缝部署至 NPU 或 CPU

MSPM0 SDK 进一步加速开发，提供丰富的即用型边缘 AI 示例项目，使开发者能够快速将参考代码部署到 MCU 上，导入预训练模型进行再训练，或将用户自定义开发的 AI 模型集成到应用中。

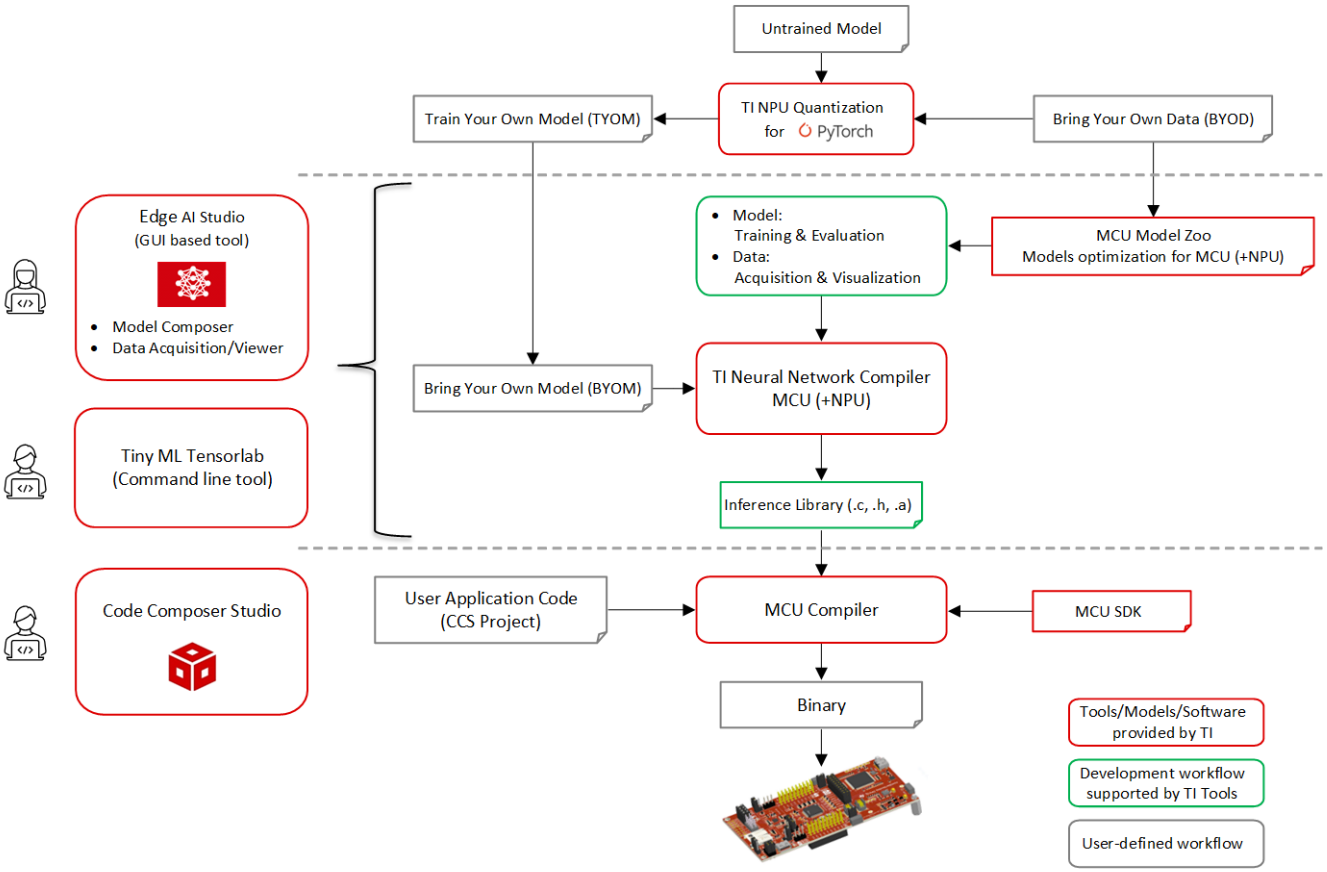


图 1-1. 在 MSPM0G5187 上部署边缘 AI 模型

Copyright © 2026 Texas Instruments Incorporated

2 带 TinyEngine™ NPU 的 MSPM0G5187

MSPM0G5187 微控制器 (MCU) 属于 MSP 高度集成的超低功耗 32 位 MCU 系列，该 MCU 系列基于增强型 Arm® Cortex®-M0+ 32 位内核平台，工作频率最高可达 80MHz。这些 MCU 为需要采用小型封装或高引脚数封装（高达 64 个引脚）的最高 128KB 闪存存储器的应用同时提供了成本优化和设计灵活性。这些器件配备 USB2.0-FS 接口、数字音频接口和 TinyEngine NPU，可在整个工作温度范围内提供出色的低功耗性能。

硬件 NPU 是一款面向深度卷积神经网络 (CNN) 的高度优化内核，支持使用预先训练的模型进行机器学习推理。这与片上 CPU 结合使用，可为 CNN 推理提供更高的性能和更低的功耗。NPU 具备每秒 640 至 2560 百万次操作 (MOPS) 的性能，相比软件实现的延迟低 90 倍，且在待机模式下功耗低于 2 µA。

图 2-1 展示了芯片内模块的顶层视图。

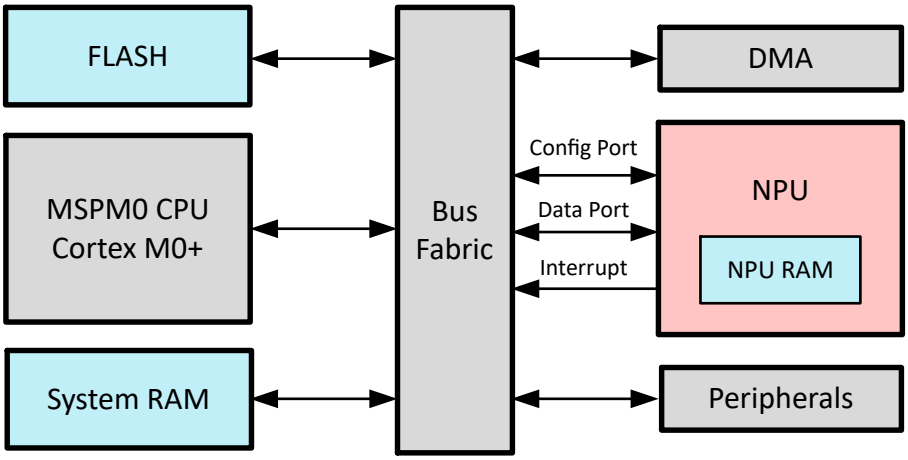


图 2-1. MSP CPU/NPU 系统结构

MSPM0 与 NPU 系统共享片上存储器，包括非易失性闪存存储器和易失性 RAM 存储器。系统 RAM 支持 CPU 或 NPU 进行读写操作。此外，NPU 还拥有用于 CNN 推理的专用 RAM。边缘 AI 模型必须适配可用闪存存储器的容量范围，且峰值 RAM 需求必须在可用系统 RAM 之内。

English Document: SDAA429
Copyright © 2026 Texas Instruments Incorporated

3 边缘 AI 工具链

Edge AI 的实现包含两个不同的阶段：离线训练工作流程和器件上运行时推理，如图 3-1 所示。

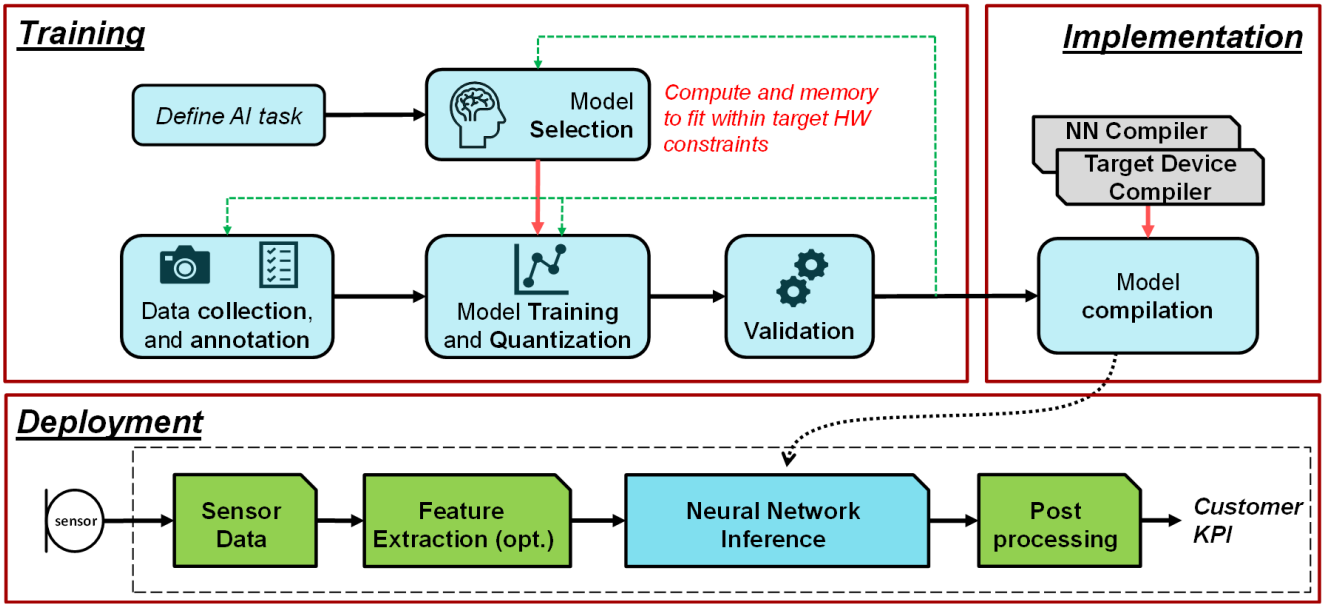


图 3-1. 边缘 AI 工作流程

模型开发工作流程

开发流水线通过四个连续阶段从初始应用定义过渡到最终的硬件编译：

- **任务定义和模型选择：**根据具体的应用要求和操作限制来定义核心 AI 工作负载。开发人员可从 TI Model Zoo 中选择结构优化且高度参数化的参考模型，也可导入专有的定制网络架构。
- **数据集管理：**高保真数据集对于在真实环境中更大限度地提高模型泛化能力至关重要。Edge AI Studio 图形用户界面 (GUI) 为数据采集和注释提供了简化的环境。该平台本身还支持摄取用户自有数据集。
- **训练和量化：**模型在 PyTorch® 框架内进行训练，并通过标准 ONNX 格式导出。为了更大限度地提高硬件效率，该流水线采用 TI 优化的训练后量化 (PTQ) 或量化感知训练 (QAT/QDQ) 方法。
- **编译：**编译后的模型会转换为高度优化的 C 代码或二进制库。该产物可以无缝、零开销地直接集成到下游 Code Composer Studio (CCS) 固件项目中。

器件上运行时数据流

成功部署到物理硬件后，系统会通过确定性流水线执行实时处理和边缘推理：

- **信号采集：**物理传感器（传感器）持续对环境变量（如加速度、电压瞬变或声学波形）进行采样，从而将其转换为数字化传感器数据。
- **特征提取：**根据可用的计算开销，原始数据可以选择性地进行局部特征提取。该阶段可降低数据维度并减轻下游推理引擎的计算负担。
- **神经网络推理：**此内核执行阶段执行高速计算和分类。硬件神经处理单元 (NPU) 极大地加速了网络工作负载，并采用 CPU 处理系统编排和控制流管理来更大限度地减少延迟。
- **后处理：**原始张量输出经过解码、标准化和转换，最终转化为高级可操作指标和客户关键绩效指标 (KPI)。

总之，这种解耦架构使得整个循环（包括物理数据感知、算法推理和确定性决策）都能够在网络边缘完全在本地执行，从而消除了对云的依赖并优化了确定性响应时间。

3.1 TI Edge AI Studio

TI Edge AI Studio 是一套综合性的图形化及命令行工具集，旨在简化开发、训练、编译 AI 模型并将其部署到各种德州仪器 (TI) 处理器、微控制器和雷达传感器上的流程。

为了适应不同的开发环境和安全要求，Edge AI Studio 提供了两种不同版本：

- 云版本：完全基于 Web 的环境，无需本地安装。该版本支持快速原型设计和快速启动，只需极少的配置。

Copyright © 2026 Texas Instruments Incorporated

- 桌面版：在用户计算机上运行的本地安装。该版本针对具有严格数据机密性要求的项目进行了优化，用于验证所有数据集和经过训练的模型是否均保留在内部，且 TI 绝不会访问这些数据。

Edge AI Studio 为以下任务提供了强大的工作流程：

- **数据管理：**无缝数据捕获、注释和可视化。
- **模型训练：**
  - 使用用户提供的数据集（自带数据，BYOD）训练 TI 优化的模型。
  - 自定义训练参数以微调模型性能。
- **模型部署：**
  - 自动编译目标硬件的训练模型。
  - 实时推理预览，用于实时性能验证。

Edge AI Studio 目前不支持某些高级自定义工作流程：

- 训练用户定义的模型（训练自己的模型，TYOM）
- 修改 TI 模型架构
- 自定义特性提取流程或量化流程
- 直接编译预训练或客户提供的模型

备注

对于需要高级模型自定义（例如 TYOM 或架构修改）的工作流程，请参阅节 3.2，其为这些功能提供了扩展支持。

Edge AI Studio 支持多个专用系列中 22 款以上的 TI 微控制器，包括采用 C7™ NPU 或 AI 加速器件的处理器（例如采用 TinyEngine 的微控制器），以及没有专用加速器的支持 AI 器件。有关更多详细信息，请访问 EDGE-AI-STUDIO 产品页面。

Copyright © 2026 Texas Instruments Incorporated

3.2 TI Tiny ML Tensorlab

Tiny ML Tensorlab 是德州仪器 (TI) 的综合 AI 工具链，用于在资源受限的微控制器 (MCU) 上开发、训练、优化和部署机器学习模型，使用户能够：

- 训练用于时间序列分类和图像分类任务的机器学习模型
- 使用量化（2 位、4 位、8 位）优化模型，用于嵌入式部署
- 将模型编译为在 TI MCU 上高效运行，可选 NPU 加速
- 使用 Code Composer Studio (CCS) 部署模型

该工具链提供端到端工作流程，引导用户通过原始数据生成在 TI MCU 上运行的编译模型。工具链处理：

- **数据准备和特征提取：**自动格式化、预处理原始信号并将其转换为高维特征。
- **模型训练：**专为嵌入式资源限制而构建的神经网络训练。
- **模型优化：**实现量化感知训练 (QAT) 或训练后量化 (PTQ)，以支持 2 位、4 位和 8 位权重精度。
- **编译：**通过 TI 神经网络编译器生成高度优化的器件特定代码。
- **性能分析：**在硬件部署之前对模型精度和推理指标进行全面评估。

TI Tiny ML Tensorlab 支持以下机器学习 (ML) 任务类型：

表 3-1. Tensorlab 支持的任务类型

任务类型	说明
时序分类	将时间序列数据分类为离散类别（例如，故障检测、活动识别）
时间序列回归	根据时间序列输入预测连续值（例如，扭矩估算）
时间序列预测	根据历史模式预测未来数值（例如，温度预测）
异常检测	使用基于自编码器的模型识别异常模式（例如，设备监控）
图像分类	将图像分类到不同的类别（例如，视觉检测、数字识别）

Tensorlab 支持多个专业系列中的多种 TI 微控制器，包括 **C2000 DSP**、**MSPM0**、**MSPM33**、**AM13x/AM26x** 和 **Connectivity**。对于延迟关键型和低功耗应用，部分器件（例如 F28P55、AM13E2 和 MSPM0G5187）采用 TI 的集成神经处理单元 (NPU)，用于硬件加速推理。

备注

集成新硬件后，支持的器件列表可能会发生变化。有关支持的器件型号的最新完整列表，请参阅 Tiny ML Tensorlab 用户指南：支持的器件。

其他资源

- Tiny ML Tensorlab GitHub 存储库
- Tiny ML Tensorlab 用户指南

3.3 TI 神经网络编译器

TI 神经网络编译器 (NNC) 是 TI 为 MCU 器件开发的嵌入式 AI 编译工具链，基于 TVM 编译框架构建。TI NNC 通过简单的编译配置，可灵活地将 AI 模型部署到硬件 NPU 加速器或主机 CPU 上，无需对应用程序代码或模型端进行任何修改。推理功能始终通过统一接口调用，使开发者无需深入了解底层硬件架构，即可将 AI 推理集成到嵌入式应用中。

表 3-2. NPU/CPU 部署配置比较

目标部署	配置符号	初始化需求	调用推理
主机 CPU	TVMGEN_DEFAULT_TI_NPU_SOFT	是的，在推理之前初始化 NPU	tvmgen_default_run()
硬件 NPU	TVMGEN_DEFAULT_TI_NPU	否	

要为 MSPM0 系列编译模型，请配置您的环境并执行如下所示的 TVMC 编译命令。

English Document: SDAA429
Copyright © 2026 Texas Instruments Incorporated

Linux® / macOS

```
export ARM_CGT_PATH=/path/to/ARM-CGT
export PATH=$ARM_CGT_PATH/bin:$PATH
export TIARMCLANG_OPTIONS_M0="-Os -mcpu=cortex-m0plus -march=thumbv6m -mtune=cortex-m0plus -mthumb
-mfloat-abi=soft -I. -Wno-return-type"
```

```
tvmc compile --target="c, ti-npu type=soft" --target-c-mcpu=cortex-m0plus ./model.onnx -o
artifacts_m0/mod.a --cross-compiler="tiarmclang" --cross-compiler-options="$TIARMCLANG_OPTIONS_M0"
```

Windows® Git BASH

```
export ARM_CGT_PATH="\path\to\ARM-CGT"    # e.g. "C:\\ti\\ti-cgt-armllvm_4.0.3.LTS"
export PATH=$ARM_CGT_PATH\\bin:$PATH
export TIARMCLANG_OPTIONS_M0="-Os -mcpu=cortex-m0plus -march=thumbv6m -mtune=cortex-m0plus -mthumb
-mfloat-abi=soft -I. -Wno-return-type"
```

```
tvmc compile --target="c, ti-npu type=soft" --target-c-mcpu=cortex-m0plus ./model.onnx -o
artifacts_m0/mod.a --cross-compiler="tiarmclang" --cross-compiler-options="$TIARMCLANG_OPTIONS_M0"
```

Windows PowerShell

```
$env:ARM_CGT_PATH="\path\to\ARM-CGT"      # e.g. "C:\\ti\\ti-cgt-
armllvm_4.0.3.LTS"$env:PATH=$env:ARM_CGT_PATH + "\bin;" + $env:PATH $env:TIARMCLANG_OPTIONS_M0="-Os
-mcpu=cortex-m0plus -march=thumbv6m -mtune=cortex-m0plus -mthumb -mfloat-abi=soft -I. -Wno-return-
type"
```

```
tvmc compile --target="c, ti-npu type=soft" --target-c-mcpu=cortex-m0plus .\model.onnx -o
artifacts_m0/mod.a --cross-compiler="tiarmclang" --cross-compiler-
options="$env:TIARMCLANG_OPTIONS_M0"
```

> 如果用户不需要二进制输出（例如，仅生成 C 代码时），则不需要 ARM Clang 编译器。可以忽略命令中的交叉编译器参数。

如需更多信息，请参阅适用于 MCU 的 TI 神经网络编译器用户指南。

English Document: SDAA429
Copyright © 2026 Texas Instruments Incorporated

4 通过 TI Edge AI Studio 进行快速原型设计

Edge AI Studio 旨在加速原型设计，并支持快速验证 TI 优化的参考模型，从而显著降低开发开销。

本节全面概述了开发环境。表 4-1 总结了此工作流程所需的工具，包括其各自的版本和访问链接。下面提供了详细的分步设置说明。

表 4-1. 开发工具要求

工具	版本	功能	访问链接
Edge AI Studio	1.7.0	用于 ML 模型开发的 GUI 工具	EDGE-AI-STUDIO-MCU
MSPM0 SDK	2.11	用于 TI MSPM0 MCU 的软件开发套件	MSPM0-SDK
ARM CLANG 编译器	5.1.1	用于 TI MSPM0 MCU 的器件编译器	ARM-CGT-CLANG
Code Composer Studio IDE	21.0	用于 MSPM0 MCU 的集成开发环境	CCStudio™ IDE 工具

备注

本文档中介绍的过程基于 Edge AI Studio 桌面版；但云版本的工作流程与之类似。

4.1 工程设置

要在 Edge AI Studio 中启动新的 AI 项目，用户可以导航至 *Workspace* 菜单。该平台提供两种主要工作流程。标准工作流程包括选择 *"Create New Project"* 选项，如图 4-1 所示。

在此过程中，用户必须通过选择以下四个专用 AI 任务类别之一来定义应用范围：

- **分类：**用于涉及分类标识和模式识别的任务。
- **回归：**用于预测连续数值。
- **异常检测：**用于识别数据集中的异常值或异常模式。
- **预测：**用于根据历史趋势预测未来数据点。

AI 任务 的选择至关重要，因为该平台会动态配置底层训练方法和优化算法，以满足所选任务的特定数学要求。

在 *Sensor Selection* 页面中，为获得理想结果，TI 建议选择 *Generic Sensor*，然后在 *Home* 页面上配置其具体类型。如果应用需要来自多个来源的数据，请在 *Data Format Definition* 页面上选择 *Multi-sensor* 选项。

配置项目后，重命名该项目以完成设置，并开始边缘 AI 开发工作流程。

English Document: SDAA429
Copyright © 2026 Texas Instruments Incorporated

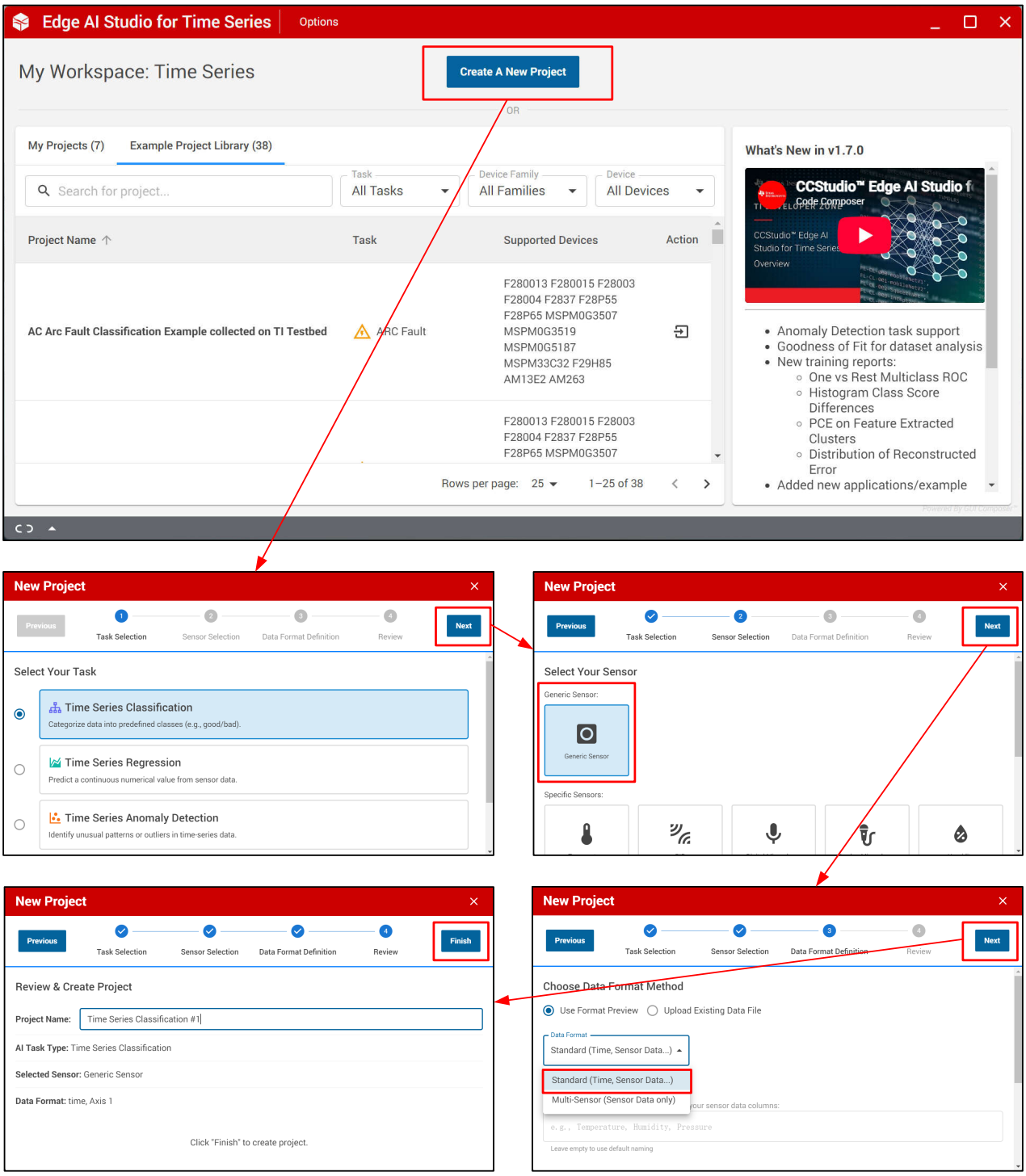


图 4-1. 使用 Edge AI Studio 创建新项目

或者，用户也可以利用 TI 预配置的示例项目库来加快开发周期。目前，该平台支持 38 个专业项目，这些项目按目标应用、特定数据集或不同的模型架构进行分类。

这些示例可作为各种工业和消费领域应用场景的基础参考。例如，在*电弧故障检测*领域中，用户可以从多种实现路径中进行选择：

- **TI 测试数据实现：**利用专有 TI 数据集实现优化的性能。

English Document: SDAA429
Copyright © 2026 Texas Instruments Incorporated

- **公共数据集实现：**利用标准化社区数据集进行通用基准测试。
- **异常检测实现：**利用专门的*异常检测 AI 任务*来识别异常电气特征。

要利用现有示例，请使用 *Search* 功能通过关键字查找特定项目。找到后，选择 *Import* 按钮可将该项目直接添加到活动的 *Workspace* 中，以立即进行开发和测试。

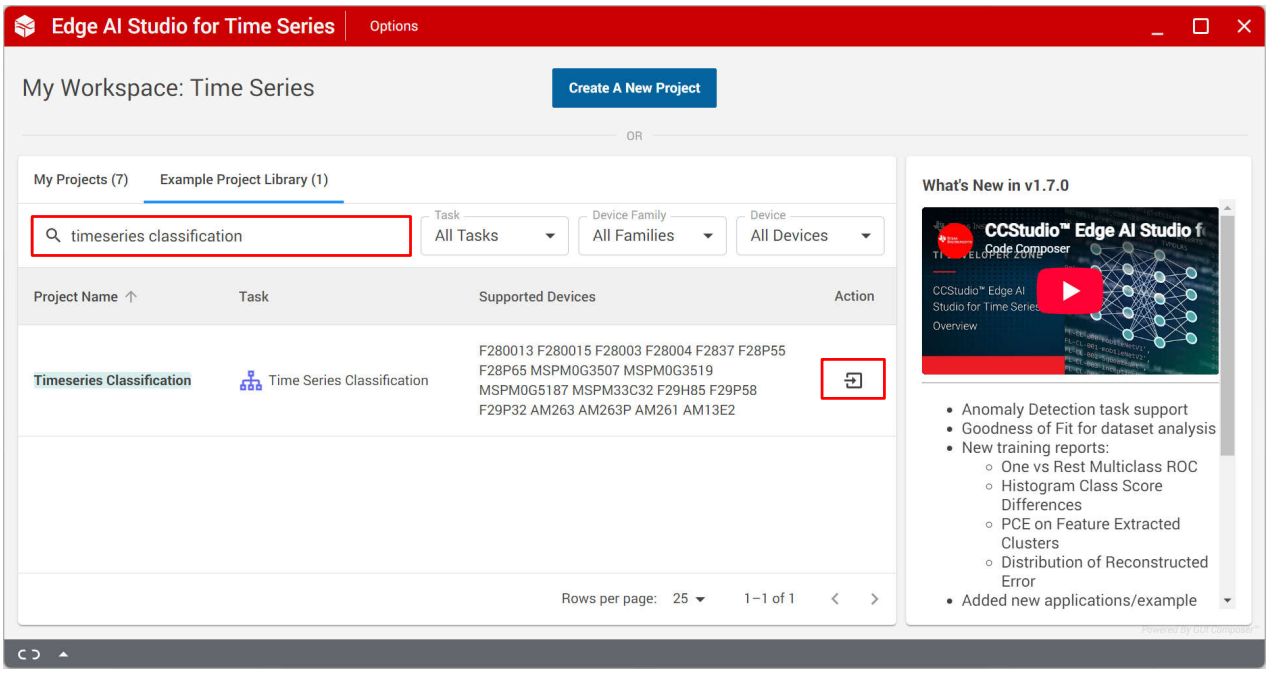


图 4-2. 使用 Edge AI Studio 导入参考项目

4.2 数据采集

本节介绍使用 TI 微控制器 (MCU) 采集实时传感器数据以及在 Edge AI Studio 中配置这些数据以进行 AI 模型开发的过程。

数据捕获支持两种主要工作流程：

- **实时数据捕获：**此工作流程利用与兼容传感器评估模块（例如 TIDA-010997）连接的 TI MCU（例如 LP-MSPM0G5187）。系统遵循标准化的 GUI 通信协议，在运行时捕获并流式传输实时传感器数据。
- **离线数据上传：**用户可以导入从其他硬件平台获取的预捕获数据集。有关数据导入过程的详细说明，请查阅节 4.3。

4.2.1 硬件设置

为了提供无缝数据流式传输并防止数据损坏，必须按照以下过程配置目标硬件和软件环境。

1. **导入项目：**打开 **Code Composer Studio (CCS)** 并从以下 SDK 目录导入项目。

   ```
   ...\ti\mspm0_sdk_<SDK_Version>\examples\nortos\LP_MSPM0G5187\edgeAI\tida_010997_data_capture
   ```

2. **构建和刷写：**编译项目并将生成的二进制文件刷写到 LP-MSPM0G5187 目标板。

备注

软件中的串行通信参数必须与固件配置匹配，以防止数据溢出或丢包。

在建立连接之前，请验证固件和主机应用中的波特率设置：

1. **固件验证 (SysConfig)：**在 CCS 项目中打开 *SysConfig* 工具，并验证 UART 配置模块中已编程的波特率
2. **Edge AI Studio 设置：**

English Document: SDAA429
Copyright © 2026 Texas Instruments Incorporated

a. 导航至 *Options* 选项卡
b. 选择 *Select Port* 以识别分配给 MCU 的有效 COM 端口
c. 设置 *Baud Rate*，以匹配 *SysConfig* 工具中已验证的值

参数同步后，请按照以下步骤启动数据采集：

1. 导航至 *Capture* 选项卡
2. 选择 *Connect To Device*
3. 通过验证左下方状态栏中的 *"Hardware connected"* 状态来确认连接

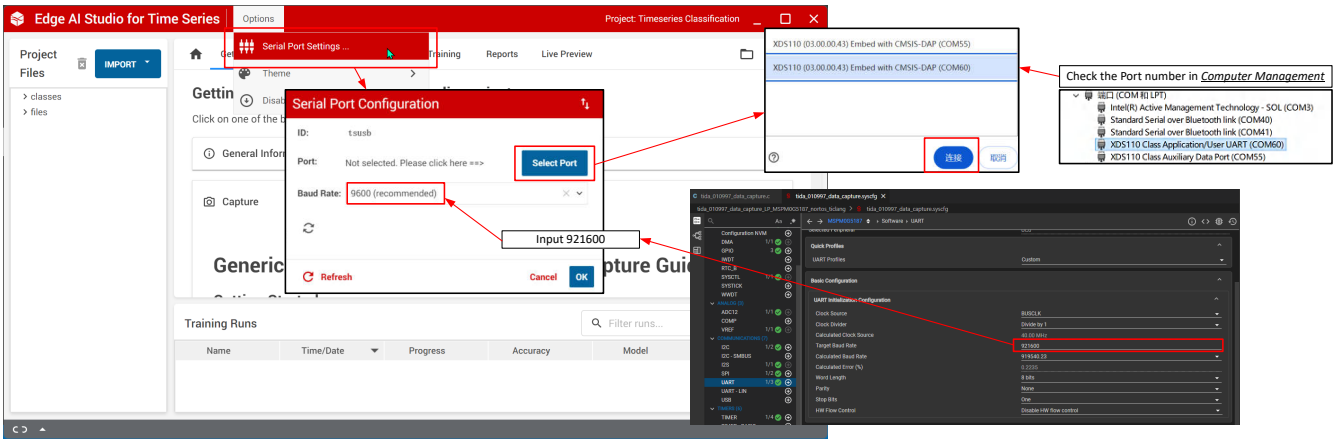


图 4-3. 将硬件连接到 Edge AI Studio

4.2.2 特征概述

Edge AI Studio 中的 **Data Capture** 界面提供了一整套工具，用于配置传感器硬件和管理数据采集参数。该平台支持多种传感器模块。例如，TIDA-010997 评估模块集成了多个高性能传感器，包括：

- 运动
  - 带菲涅耳透镜的双元件 PIR 传感器 (IRA-S210ST01 + IML-0685)
  - 低功耗 6 轴加速计和陀螺仪 (BMI270)
  - 高精度线性 3D 霍尔效应传感器 (TMAG5170)
- Environment：
  - 数字湿度和温度传感器 (HDC3020)
  - 高速、高精度环境光传感器 (OPT4001)
  - 数字压力传感器 (BMI384)
- 音频：
  - 数字 I2S 输出底部端口麦克风 (ICS-43434)
  - 超低噪声、差分模拟麦克风 (ICS-40740)

在数据捕获过程中，用户可以配置以下参数，以提供有序且可重复的数据收集：

- **数据标签：**为数据集分配自定义标识符，该标识符用于自动文件命名约定，以实现高效的数据管理
- **样本计数：**定义单个捕获会话中要收集的样本数
- **捕获间隔：**配置连续捕获周期的数量和每个周期之间的延时时间，从而实现精确的硬件同步和测试

采集完成后，用户可以通过导航至以下位置 来查看捕获的时间序列数据：*Project Files → files*。

Copyright © 2026 Texas Instruments Incorporated

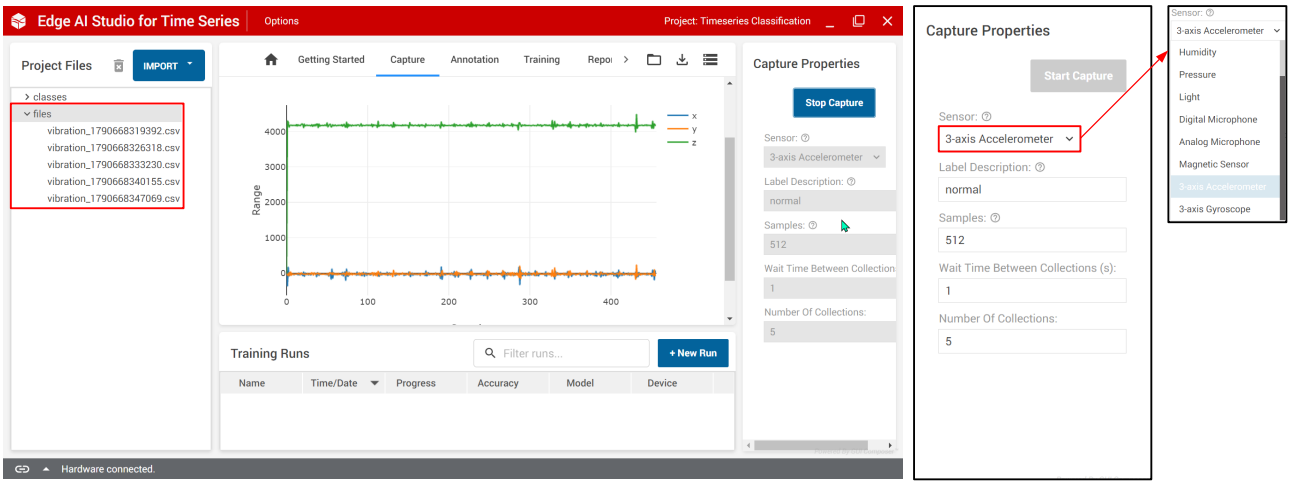


图 4-4. 捕获页面概览

4.3 数据导入

为便于进行直接模型训练，用户可以从本地 PC 导入预先捕获的数据集，如图 4-5 中所示。

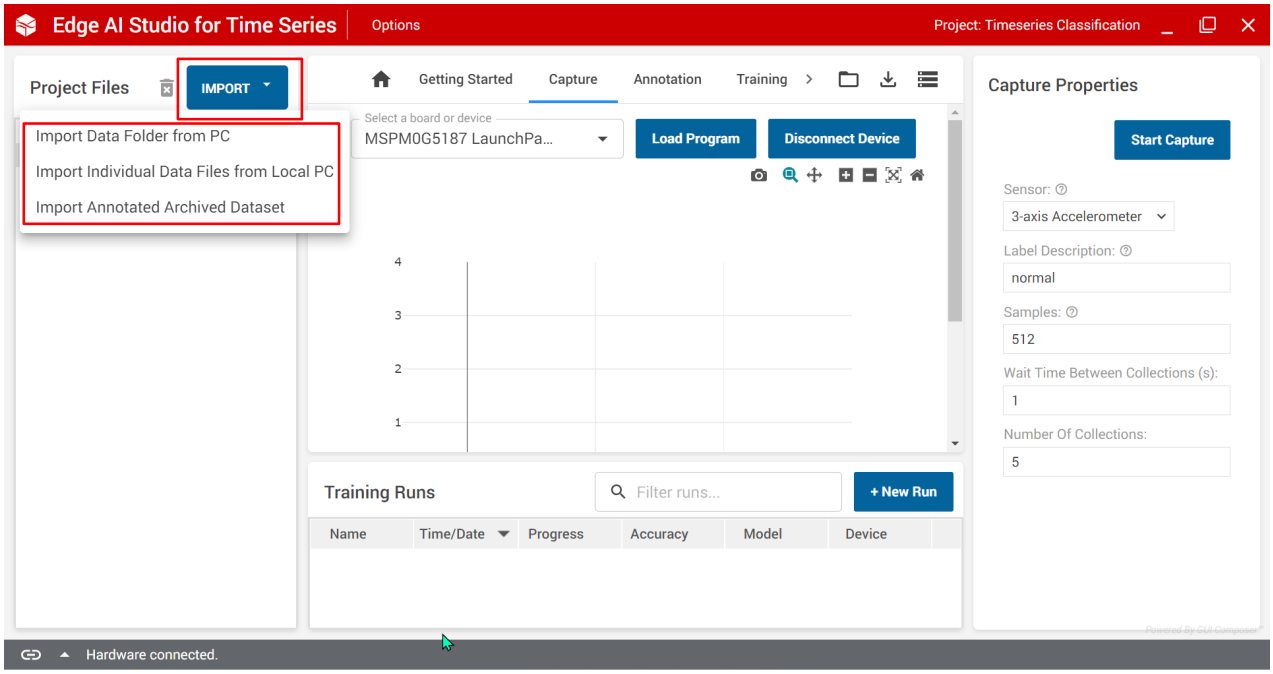


图 4-5. 从本地 PC 导入数据

要成功获取数据，上传的文件必须符合每个任务复杂度所需的特定格式。如需数据准备的全面指导，请参阅 TI Tiny ML Tensorlab 用户指南：自带数据。

以下各节详细介绍了各种 AI 任务支持的数据格式。

4.3.1 分类任务

数据格式

A. 无表头格式（简单）

English Document: SDAA429
Copyright © 2026 Texas Instruments Incorporated

此格式适用于单变量数据集。无表头格式包含原始数值，每行一个测量值。

示例：

```
0.523
0.612
0.498
0.701
...
```

B. 有表头格式（建议用于多变量）

建议对多通道传感器数据采用此格式。第一行必须包含列名。

示例：

```
channel_x,channel_y,channel_z
0.523,0.112,-0.234
0.612,0.098,-0.198
...
```

C. 时间列处理

Edge AI Studio 会自动检测并排除时间序列时间戳，防止其被视为特征变量。在导入过程中，任何名称包含字符串 **"time"**（不区分大小写）的列都会自动删除。

示例：

```
Time,value1,value2
0.001,0.523,0.112
0.002,0.612,0.098
...
```

4.3.2 回归任务

数据格式

回归任务涉及根据多个输入特征预测连续数值。对于回归任务，*目标值（标签）必须位于数据集的最后一列*。该文件必须采用带表头的格式，其中第一行定义特征名称和目标变量。格式结构如下所示：

```
feature1,feature2,feature3,...,target
0.5,18.2,45.5,...,0.187
0.6,18.5,45.6,...,0.245 ...
```

示例：电机扭矩预测

- 第 1-6 列：输入特征
- 第 7 列（最后一列）：要预测的目标值

```
current_d,current_q,voltage_d,voltage_q,motor_speed,pm_temp,target_torque
-0.450,0.032,18.805,1.499,0.002,24.55,0.187 -0.325,0.045,18.818,1.542,0.003,24.54,0.245
-0.440,0.028,18.876,1.456,0.002,24.54,0.176
```

为防止时间戳被错误地处理为输入特征，Edge AI Studio 会自动识别和删除任何包含时间的列。

标记逻辑

备注

回归任务需要*简单窗口*变换。

Edge AI Studio 对目标值应用窗口均值处理。该处理可稳定时间序列建模的目标标签：

- **窗口提取：**从原始目标列中提取大小为 N（由 *frame_size* 参数定义）的窗口
- **均值计算：**对该窗口内的目标值求平均

English Document: SDAA429
Copyright © 2026 Texas Instruments Incorporated

- **标签分配：**此计算的均值将成为整个窗口的单一目标标签

示例计算 (frame_size = 4)：

```
Window 1: rows 0-3, targets [0.18, 0.24, 0.17, 0.19] → avg = 0.195
Window 2: rows 2-5, targets [0.17, 0.19, 0.22, 0.20] → avg = 0.195
```

4.3.3 预测任务

预测任务包括根据历史观察预测一个或多个指定变量的未来值。

在预测中，所有变量均按列进行组织。与回归（目标为专用列）不同，预测利用数据集中的现有变量。用户必须明确指定哪些列用作输入特征以及哪些列用作要通过配置参数进行预测的变量。

示例：温度预测

```
ambient,coolant,current,pm_temp
19.85,18.81,2.28,22.93
19.85,18.79,2.28,22.94
19.85,18.79,2.28,22.94
19.85,18.77,2.28,22.94
...
```

在此示例中，数据集的结构使前三列作为输入特征，而最后一列表示要预测的目标变量。列映射定义如下：

- 第 0 列 (**ambient**)：输入特征
- 第 1 列 (**coolant**)：输入特征
- 第 2 列 (**current**)：输入特征
- 第 3 列 (**pm_temp**)：要预测的目标

虽然回归和预测任务都使用连续的数值目标，但它们在时间逻辑和目标定义方面有着根本的不同，如表 4-2 所示。

表 4-2. 回归和预测任务的比较

特征	回归任务	预测任务
主要目标	根据当前或最近的窗口特征预测连续值。	通过识别历史时间模式和趋势来预测未来值。
目标定义	窗口平均值：目标定义为当前观察窗口中的平均值。	未来值：目标是预定义未来时间点的特定值。
数据要求	需要标准窗口转换。	需要显式配置预测范围。

4.3.4 注释列表

为便于进行精确的模型训练和评估，Edge AI Studio 支持通过专用注释文件进行显式数据集分区。用户可以定义用于训练、验证和测试的特定子集，以提供强大的模型性能。

> 对于较小的数据集或快速原型设计，用户可以直接在 Edge AI Studio 图形用户界面 (GUI) 中手动分配标签和分区。

项目目录结构

所需的目录组织取决于 AI 任务类型（分类与回归/预测）。请参考以下结构：

对于分类任务：必须将数据集整理到代表每个目标类的子目录中。

```
my_dataset/
├── classes/
│   ├── class_name_A/
│   │   └── sample_01.csv
│   └── class_name_B/
│       └── sample_01.csv
└── annotations/
    ├── file_list.txt              # All files (auto-generated if missing)
    ├── instances_train_list.txt   # Training subset
```


English Document: SDAA429
Copyright © 2026 Texas Instruments Incorporated

```
    ├── instances_val_list.txt      # Validation subset
    └── instances_test_list.txt     # Test subset (optional)
```

对于回归和预测任务：数据作为单个文件存储在平面目录结构中。

```
my_dataset/
├── files/
│   ├── datafile_01.csv
│   ├── datafile_02.csv
│   └── datafile_N.csv
└── annotations/
    ├── instances_train_list.txt
    ├── instances_val_list.txt
    └── instances_test_list.txt
```

注释文件格式

每个注释文件 (.txt) 都必须包含一个相对路径列表，其中每条路径独占一行。

示例：

```
# instances_train_list.txt
class_A/sample_01.csv
class_A/sample_02.csv
class_B/sample_01.csv
```

4.4 数据注释

成功采集或导入数据后，工作流程将继续进行数据注释和数据集分区。这些过程通过 Edge AI Studio GUI 执行，以准备用于模型训练和评估的原始数据。

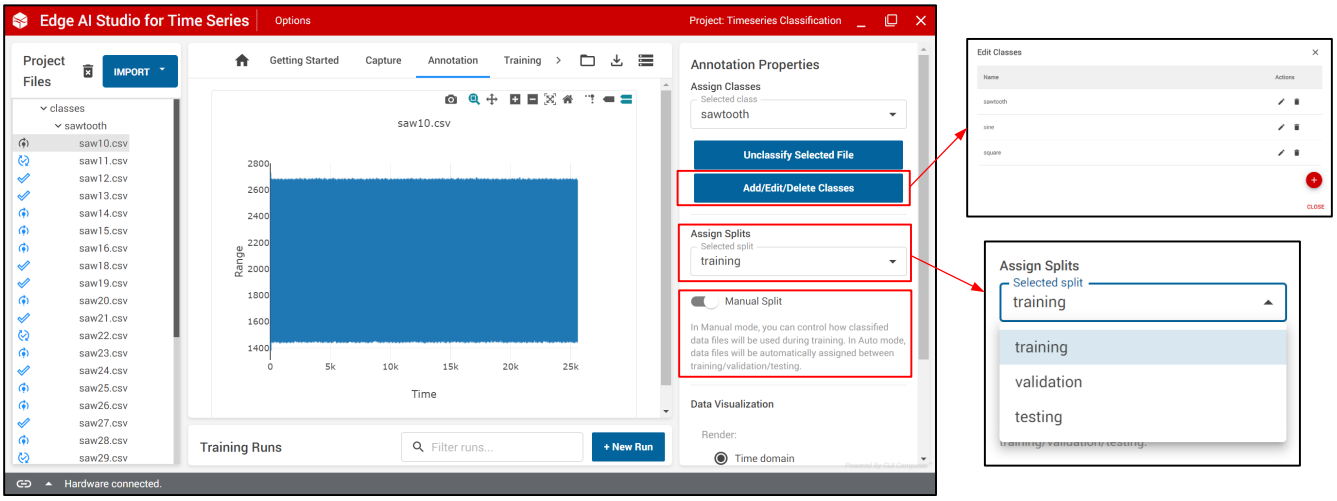


图 4-6. 注释页面概览

注释工作流程

在**数据注释**界面中，用户可执行以下操作将原始数据组织成有意义的类：

- **类管理：**用户可以创建新类的类别或重命名现有类的类别，以符合目标应用要求。
- **数据分配：**可以选择单个文件并将其分配给特定类，也可以在现有类别之间移动以优化数据集。

数据集分区

分配类后，必须将数据集划分为三个不同的子集：*训练*、*验证* 和 *测试*。表 4-3 中总结了每个子集的要求和用途。

表 4-3. 数据集要求

子集	要求	用途
训练数据集	必需	模型用来学习特征和优化权重。

English Document: SDAA429
Copyright © 2026 Texas Instruments Incorporated

表 4-3. 数据集要求（续）

子集	要求	用途
验证数据集	必需	在训练过程中用于调优超参数和监测是否过度拟合。
测试数据集	可选	用于以未见过的数据评估最终模型性能。

Edge AI Studio 提供了**手动拆分**模式，允许用户对样本分布进行完全精细的控制。对于必须严格保留特定边缘情况用于测试的专用应用，建议使用此模式。

必须仔细考虑测试数据集的分布，以确保模型评估具有统计学意义。

备注

为获得更可靠的性能指标，测试集必须占总数据集的相当大一部分。对于大多数应用，建议测试集占总数据的 50% 以上。随着数据集整体规模的增加，测试集的比例也必须相应提高，以确保评估结果在统计上具有稳健性并代表实际工作条件。

4.5 模型训练

Edge AI Studio 模型训练界面提供了配置模型架构和训练超参数的简化工作流程。本节概述配置序列和底层编译过程。

Copyright © 2026 Texas Instruments Incorporated

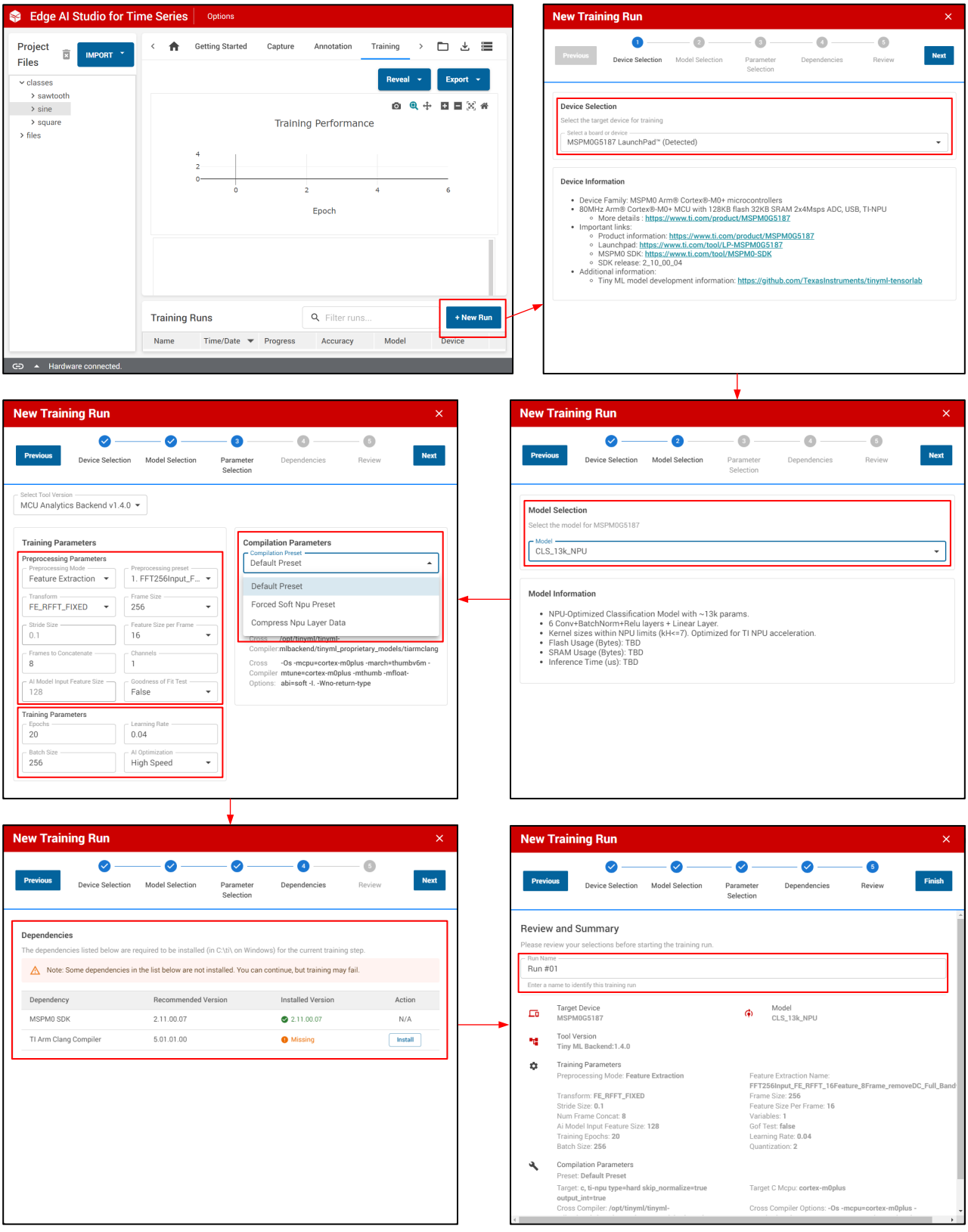


图 4-7. 训练页面概览

English Document: SDAA429
Copyright © 2026 Texas Instruments Incorporated

型号和硬件配置

要启动训练过程，用户必须定义目标硬件和模型架构。

1. **目标器件选择：**用户必须从提供的下拉菜单中选择目标微控制器 (MCU)。Edge AI Studio 支持包含 20 多个 MCU 系列的多样化生态系统。这种跨平台功能允许根据特定的功率和性能要求在不同的硬件平台之间移植和部署单个 AI 模型架构。
2. **AI 模型选择：**对于分类任务，Edge AI Studio 提供了 24 种以上的 TI 优化模型架构。选择模型时，用户必须在计算复杂性和预测准确性之间进行权衡：
   a. **模型复杂性：**具有更多层和参数的较大模型通常可以实现更高的准确性。
   b. **资源限制：**增加的参数数量与更高的存储器（SRAM/存储器）消耗和更长的推理延迟直接相关。

选择模型并定义训练参数（有关详细参数说明，请参阅节 4.5.1）后，用户必须指定执行目标：

- **默认预设 (NPU)：**通过神经处理单元 (NPU) 优化模型以实现硬件加速。
- **强制软 NPU 预设 (CPU)：**将推理工作负载路由到 CPU，通常用于调试，或用于没有专用 NPU 硬件的器件。

训练和编译流水线

从经过训练的模型到目标就绪二进制文件的转换遵循严格的多级流水线。此过程在训练阶段完成后开始，并生成模型产物。

1. **训练阶段：**系统根据提供的数据集优化模型参数。成功完成后，Edge AI Studio 会生成 ONNX 格式的高级模型文件。此 ONNX 文件用作后续转换阶段的标准化中间表示
2. **ONNX 到源代码转换：**生成的 ONNX 模型随后由 **TI NNC** 处理。NNC 执行图形优化和运算符映射，以将高级 ONNX 表示转换为优化的源代码
3. **源代码到二进制转换：**在最后一个阶段，优化的源代码被传递到**器件编译器**（例如 ARM CLANG）。编译器会执行低级机器代码生成，以生成为特定目标 MCU 架构量身定制的最终二进制库（.a 文件）

备注

如果 PC 系统上未安装器件编译器，则编译流水线将终止，并出现致命错误。由于训练独立于编译工具链，因此上述模型训练阶段正常完成。

4.5.1 训练参数

训练参数配置分为两个主要阶段：预处理参数和训练超参数。

预处理参数

Edge AI Studio 支持两个预处理工作流程，以准备进行模型训练的数据：

- **简单窗口：**该方法通过滑动窗口机制使用数据集中的原始时间序列数据。用户可以在 0.01 至 1.0 范围内配置窗口大小
- **特征提取：**此工作流程执行高级数据转换。在特征提取模块中，用户可以在两种处理模式之间进行选择：
  - **FE_RFFT_FIXED：**使用 FFT 来提取频域信息。支持的 FFT 大小必须为 2 的幂，输出大小定义为“帧大小/2”。FFT 输出随后会聚合到由“每帧特征大小”参数定义的特定数量区间中
  - **原始：**利用原始的时间序列数据而不进行频率变换

以下参数用于配置特征提取工作流程。这些设置适用于 **FFT** 和**原始**处理模式：

表 4-4. 特征提取参数

参数	说明
帧大小	指定每个处理帧的输入样本数。
每帧特性大小	定义每帧生成的区间数（或特征数）。
要连接的帧	指定要从缓冲区中检索的前一个帧的区间数。这种机制使得模型能够捕获整个频谱上的时间变化。
通道数	定义了要同时处理的传感器通道数量。

English Document: SDAA429
Copyright © 2026 Texas Instruments Incorporated

有关全面的实现示例，请参阅节 7.1（波形分类器应用）。

训练超级参数

训练超参数控制优化过程和最终模型特征：

- **轮次数量：**定义完整遍历训练数据集的总次数。通常，增加轮次的数量可使模型参数收敛到更接近理想设计，但过多轮次可能会导致过度拟合。
- **学习率：**确定在每个轮次梯度下降后参数更新时的步长。较高的学习率可以加快接近全局理想状态，但可能会导致振荡或不稳定。相反，较低的学习率提供更稳定的收敛，但需要更多的轮次，并且可能会陷入局部理想的情况。
- **批次大小：**指定在单次迭代中处理的训练样本数。请注意，较大的批次大小会增加主机 PC 上的计算负载和存储器压力。
- **优化（精密模式）：**定义经过训练的模型的目标精度：
  - **高精度：**训练全精度浮点模型。
  - **高速：**利用量化感知训练 (QAT) 生成针对高速推理优化的量化定点模型。

4.5.2 工具链依赖项

即使主机系统上已安装所需的器件编译器 (ARM Clang) 和 MSPM0 SDK，Edge AI Studio 中的模型编译过程仍可能会失败。此问题主要在早期版本的 Edge AI Studio 中出现。

在某些版本中，Edge AI Studio 可能无法自动检测所需工具的安装路径，导致后端在编译阶段无法正确调用编译器。

如果由于工具检测问题导致编译失败，您可以通过以下步骤手动指定工具安装路径：

1. 导航至以下目录（将 [version] 替换为具体安装版本）：

```
...\ti\eaistudio[version]-timeseries\model-composer-extensions\tinyml-backend-[version]
\_internal\tinyml_modelmaker\ai_modules\timeseries
```

2. 打开文件：

```
constants.py
```

3. 找到以下代码块并修改 TOOLS_PATH 变量：

```
TOOLS_PATH = os.path.abspath(os.getenv('TOOLS_PATH', os.path.join(f'{HOME_DIR}', 'bin')))
```

4. 将其改为直接指向您的 TI 安装目录，通常如下所示：

```
TOOLS_PATH = 'C:\\ti'
```

5. 验证编译器版本并确保 MSPM0_CGT_VERSION 与安装的版本匹配。例如：

```
MSPM0_CGT_VERSION = 'ti-cgt-armllvm_5.1.1.LTS'
```

4.6 性能报告

完成训练过程后，用户可以访问 *Reports* 界面来评估模型的预测性能。*Reports* 界面提供了三个不同的分析维度：混淆矩阵、多类 ROC 曲线和成对类预测差异，从而能够对准确性、稳健性和决策裕度进行全面评估，如图 4-8 所示。

English Document: SDAA429
Copyright © 2026 Texas Instruments Incorporated

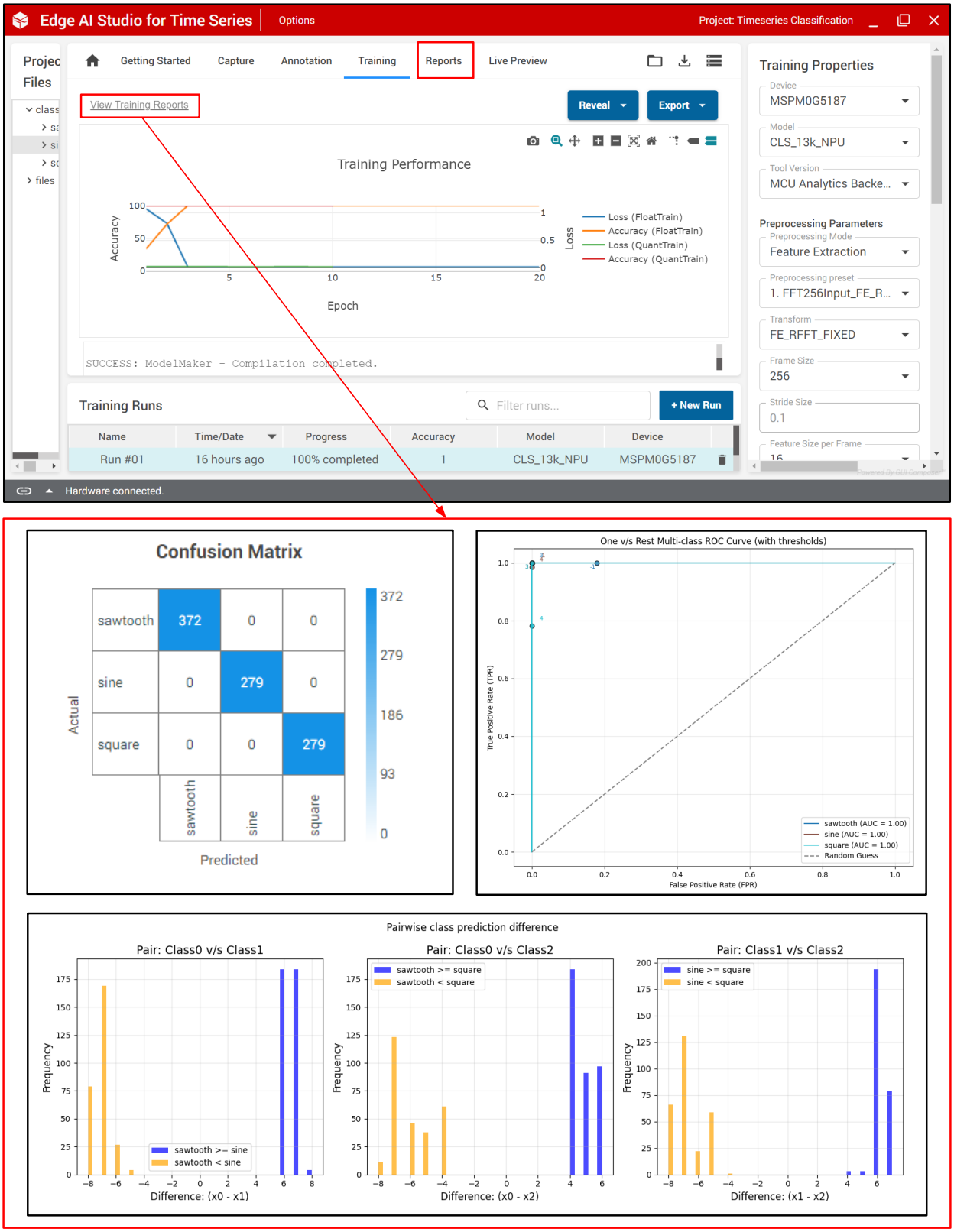


图 4-8. 报告页面概览

English Document: SDAA429
Copyright © 2026 Texas Instruments Incorporated

混淆矩阵

混淆矩阵是在特定决策阈值下可视化受监督学习算法性能的基本工具。混淆矩阵提供了模型如何在实际数据集之间分发预测的精细视图。

该矩阵以网格形式组织，其中：

- **行（Y 轴）：**表示测试数据集中的**实际/真实类**标签。
- **列（X 轴）：**表示模型生成的**预测类**标签。
- **对角线元素：**表示**正确的预测**（预测标签与实际标签匹配）。
- **非对角线元素：**表示**错误分类**（错误）。

混淆矩阵用作评估模型可靠性的多层面诊断工具。首先，沿主对角线的值集中程度直接反映了模型的整体准确率，以及各个类的精确率和召回率；对角线值密度较高，表明模型成功捕获了每个类的底层模式。

除了整体准确率之外，该矩阵还可以通过非对角线元素分析实现复杂的错误模式诊断。通过观察错误分类聚集的位置，工程师可以识别特征混淆的特定实例 — 例如，如果类 A 经常被误认为是类 B，则矩阵表明当前特征集缺乏区分这两个特定类别所需的判别分辨率。此类见解为优化上游特征工程或数据预处理提供了有针对性的方向。

此外，该矩阵有助于对数据集完整性进行关键验证。通过计算各行或各列中元素的总和，用户可以验证测试集中的样本分布。此步骤至关重要，可确保性能指标不因分类不平衡而偏斜，从而保证所报告的准确率能够客观反映模型的真实泛化能力。

多类 ROC 曲线（一对多）

接收器工作特征 (ROC) 曲线通过绘制真阳性率 (TPR) 与假阳性率 (FPR) 的关系图来评估模型区分不同类的能力。对于多类任务，Edge AI Studio 采用一对多 (OvR) 策略。在这种方法中，每个单独的类被视为“正”类，而所有其他类被聚合到一个“负”类。这样便可为模型中的每个类生成不同的 ROC 曲线。

多类 ROC 曲线全面展现模型在不同运行条件下的判别能力。此评估的主要指标是曲线下面积 (AUC)；对于给定的类，AUC 值接近 1.0 表示其相应的曲线被紧密推到左上角，表示目标类与数据集的部分之间近乎完美分离。相反，AUC 接近 0.5 表示该模型缺乏预测效用，其表现与随机猜测无异。

除了单个标量值外，ROC 曲线也是评估阈值稳健性的重要工具。由于曲线跟踪不同分类置信度阈值范围内的性能，因此曲线的稳定性至关重要，特别是其在广泛的点范围内保持高真阳性率 (TPR) 同时尽可能降低假阳性率 (FPR) 的能力。将显式阈值注释直接集成到曲线上，使工程师能够直观地了解特定分数如何映射到真阳性率和假阳性率。

对于在实时嵌入式系统中部署模型的工程师而言，稳定的 ROC 曲线表明，即使动态调整分类阈值以满足特定应用要求（例如灵敏度或误报容差），该模型仍然可靠。

成对类预测差异

成对类预测差异直方图分析决策裕度 — 即模型区分任意两个特定类的置信度。该直方图绘制了两个类之间原始决策分数（或预测概率）差异的分布。直方图捕获模型输出分布的统计行为，而不仅仅是最终的硬标签。

传统指标侧重于二元正确性，而成对类预测差异直方图则能更深入地分析模型的决策裕度和概率确定性。通过检查任何两个类之间的分数差异分布，工程师可以表征定义模型分类边界的统计“差距”。

高性能模型的特征是具有明显的双峰分布，其中两个峰值相距甚远。这种结构表明模型具有宽决策裕度，从而对其输出结果具有较高的置信度。更重要的是，零点附近（即**“模糊区”**）的分布密度是衡量模型抗噪能力的关键指标。

零值周围密度极小或“真空”的直方图表明，该模型对轻微信号扰动具有高度韧性。在实际部署中，这意味着模型具有出色的抗干扰能力，因为从统计角度来看，其由于瞬态噪声或环境波动而触发错误的状态转换的可能性较低。

English Document: SDAA429
Copyright © 2026 Texas Instruments Incorporated

4.7 模型导出

训练阶段完成后，可以导出经过训练的模型，以便集成到自定义开发工作流程中。对于边缘端部署，Edge AI Studio 还支持导出编译的模型。请注意，从版本 1.7.1 开始，支持的导出格式仅限于针对边缘器件进行优化的二进制文件。

Edge AI Studio 中的模型导出工作流程如图 4-9 所示。

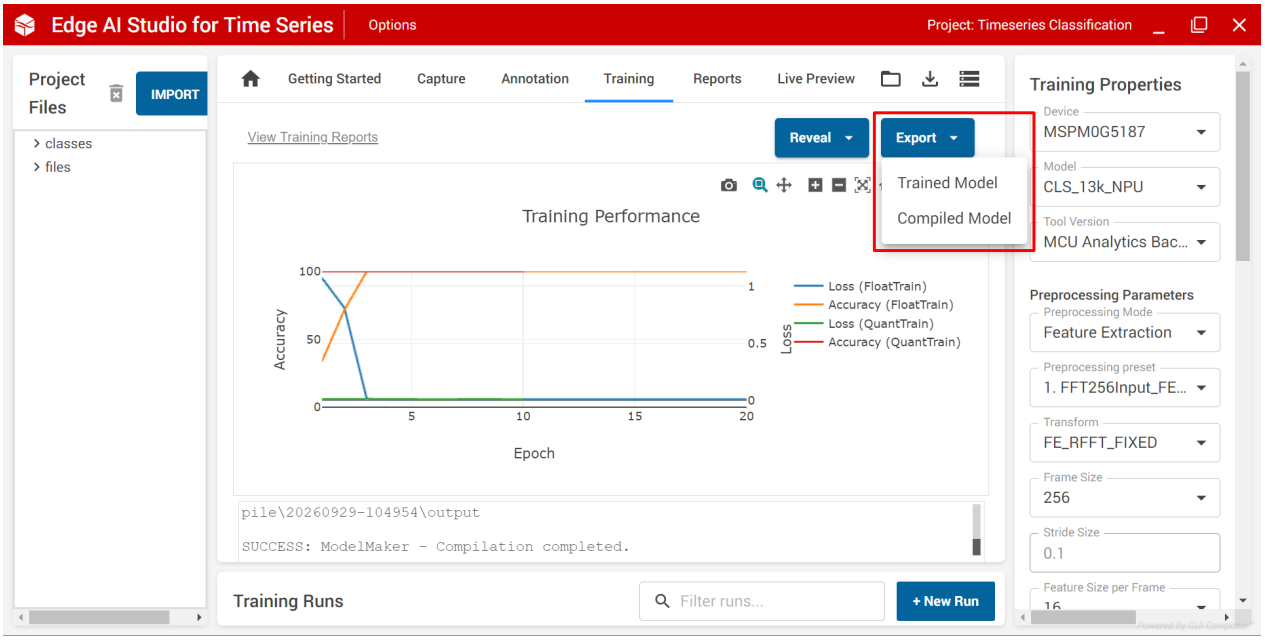


图 4-9. 从 Edge AI Studio 导出模型

成功导出后，系统会生成两种不同类型的产物：**训练后的** NN 模型文件和**编译后的** NN 模型文件。图 4-10 和图 4-11 中分别提供了这些导出文件的详细目录结构和具体内容。

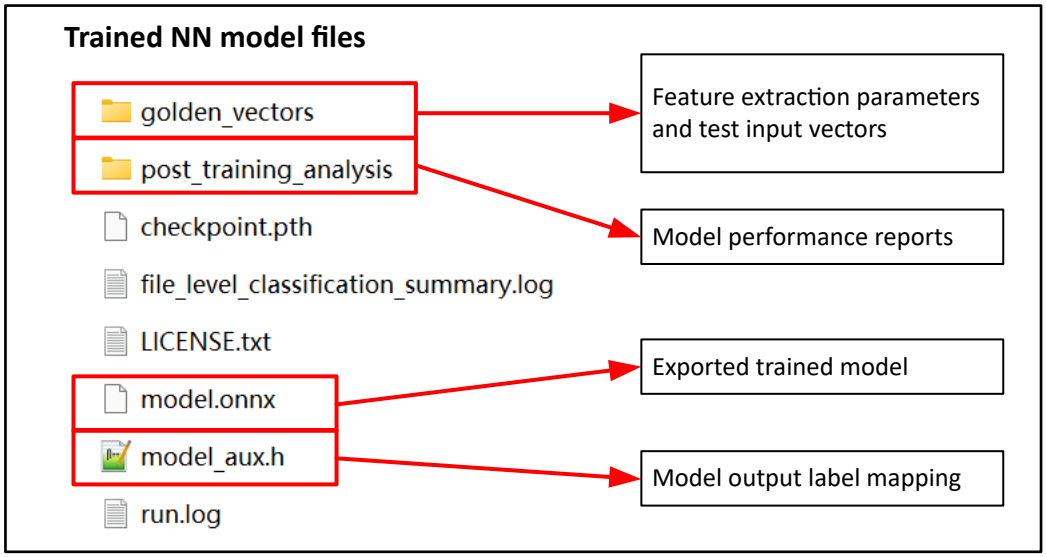


图 4-10. 训练后的 NN 模型文件

Copyright © 2026 Texas Instruments Incorporated

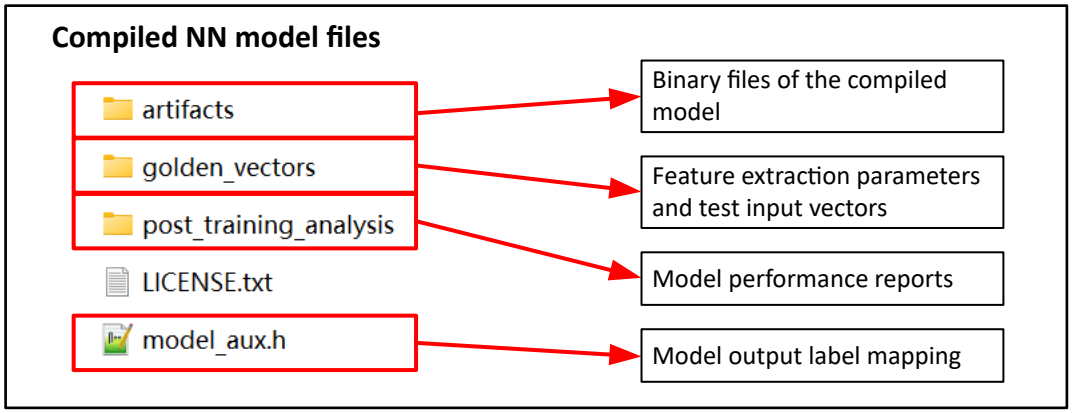


图 4-11. 编译后的 NN 模型文件

English Document: SDAA429
Copyright © 2026 Texas Instruments Incorporated

5 通过 TI Tiny ML Tensorlab 进行高级开发

本节介绍 Tensorlab，一个面向工作流程的工具集，其对基于 PyTorch 的模型训练进行优化。通过利用核心 PyTorch 模块和类，Tensorlab 有助于神经网络的端到端设计和训练。

与高级抽象工具不同，Tensorlab 专为需要对机器学习管道进行精细端到端控制的专业开发人员而设计。Tensorlab 为高度定制的开发提供了必要的灵活性，能够精确操作数据采集、模型架构、训练超参数和编译工作流程。

以下实现将 MNIST（数字识别）数据集用作基础基准，以演示如何从第一性原理构建神经网络。技术进步从基线实现开始，并以迭代方式整合 PyTorch 生态系统的高级功能。该工作流程包含整个部署生命周期：通过 ONNX 导出模型、使用 TVM (TI NNC) 进行编译和 PC 端推理。本节最后介绍了量化感知训练 (QAT) 的实现，该训练利用 TI 特定的 NPU 量化技术来优化边缘部署的模型性能。

其他资源

- Edge AI 用户指南 (MSPM0-SDK)
- Tiny ML Tensorlab 用户指南
- PyTorch 网页：What is torch.nn really?
- Tiny ML 工作流程示例 Python 脚本包

5.1 环境设置

安装

要在系统上安装 Tiny ML Tensorlab，请参阅 Tiny ML Tensorlab 用户指南：安装。选择与您的开发环境一致的安装路径，请注意此处所述的工作流程已针对开发人员安装进行了验证。

虚拟环境 (venv) 配置

为提供环境隔离并防止依赖项冲突，所有 Python 脚本都必须在 Tensorlab 虚拟环境中执行。所有操作都必须从根安装目录启动：

```
C:\ti\tinyml-tensorlab
```

根据您的首选终端接口，使用以下命令激活该环境：

A. PowerShell

```
cd C:\ti\tinyml-tensorlab
\venv\Scripts\Activate.ps1
```

B. Git Bash

```
cd C:\ti\tinyml-tensorlab
source ./venv/Scripts/activate
```

VS 代码配置

要将虚拟环境与 Visual Studio Code 集成以进行开发和调试，请执行以下步骤：

1. 按以下组合键打开“命令面板”

   ```
   Ctrl + Shift + P
   ```

2. 搜索并选择：

   ```
   Python: Select Interpreter
   ```

3. 从结果列表中，选择与“venv”目录关联的解释器。

English Document: SDAA429
Copyright © 2026 Texas Instruments Incorporated

验证依赖项

激活环境后，可通过执行以下命令来验证已安装的软件包并确保所有必需的依赖项均存在：

```
pip list
```

5.2 数据采集

为提供模块化且可验证的开发工作流程，数据采集过程分为两个不同的阶段：数据检索和数据验证。这种模块化方法可以对网络连接性、文件系统完整性和 PyTorch 张量兼容性进行独立测试。

5.2.1 数据检索

第一阶段涉及执行脚本以下载 MNIST 数据集并在本地项目结构内组织该数据集。为防止执行过程中出现与路径相关的错误，脚本会根据当前文件位置使用绝对路径解析。这可确保

```
data/
```

目录始终相对于脚本所在目录创建和访问，无论终端的当前工作目录为何。

1. 创建一个以下名称的新文件

   ```
   01_test_download_data.py
   ```

   （位于项目根目录中）。

2. 实现以下脚本

   ```
   from pathlib import Path
   import requests

   # 1. Define the absolute base directory and target path
   BASE_DIR = Path(__file__).resolve().parent
   PATH = BASE_DIR / "data" / "mnist"

   # 2. Ensure the target directory hierarchy exists
   PATH.mkdir(parents=True, exist_ok=True)

   # 3. Define remote source and local filename
   URL = "https://github.com/pytorch/tutorials/raw/main/_static/"
   FILENAME = "mnist.pkl.gz"

   # 4. Execute conditional download
   if not (PATH / FILENAME).exists():
       print("Downloading MNIST dataset. Please wait...")
       content = requests.get(URL + FILENAME).content
       (PATH / FILENAME).open("wb").write(content)
       print("Download complete.")
   else:
       print("Local dataset detected. Skipping download.")
   ```

3. 使用终端运行该脚本。成功完成后，终端会显示以下日志：

   ```
   Downloading MNIST dataset. Please wait...
   Download complete.
   ```

5.2.2 数据验证

第二阶段验证下载文件的完整性，将原始数据可视化，并将 NumPy 数组转换为 PyTorch 张量。

1. 创建一个以下名称的新文件

   ```
   02_test_load_visual_data.py
   ```

2. 实现以下验证脚本

   ```
   import pickle
   import gzip
   from pathlib import Path
   from matplotlib import pyplot
   ```


English Document: SDAA429
Copyright © 2026 Texas Instruments Incorporated

```
import numpy as np
import torch

# 1. Synchronize path resolution with Stage 1
BASE_DIR = Path(__file__).resolve().parent
PATH = BASE_DIR / "data" / "mnist"
FILENAME = "mnist.pkl.gz"

print("--- 1. Loading and Decompressing Dataset ---")
# Validate file existence before proceeding
if not (PATH / FILENAME).exists():
    raise FileNotFoundError(f"Dataset not found at: {PATH / FILENAME}. "
                            "Please ensure Stage 1 was completed successfully.")

with gzip.open((PATH / FILENAME).as_posix(), "rb") as f:
    ((x_train, y_train), (x_valid, y_valid), _) = pickle.load(f, encoding="latin-1")

print(f"Training set shape (NumPy): {x_train.shape}")

print("\n--- 2. Visualizing Sample Data ---")
# Reshape to 28x28 for image rendering
pyplot.imshow(x_train[0].reshape((28, 28)), cmap="gray")
print("Note: Close the graphical window to proceed with tensor conversion.")
pyplot.show()

print("\n--- 3. Converting NumPy Arrays to PyTorch Tensors ---")
x_train, y_train, x_valid, y_valid = map(
    torch.tensor, (x_train, y_train, x_valid, y_valid)
)

print("\n--- 4. Verifying PyTorch Tensor Integrity ---")
print(f"Training Data Sample:\n{x_train}\n{y_train}")
print(f"Converted Tensor Shape: {x_train.shape}")
print(f"Label Range: Min={y_train.min()}, Max={y_train.max()}")
print("Success: Data pipeline and PyTorch integration verified.")
```

3. 运行该脚本。窗口将显示 MNIST 数据集的第一个手写数字，如图 5-1 所示。关闭该图像窗口后，终端会显示转换的张量形状和成功消息。成功执行可确认数据集已正确加载、环境已正确配置，且 PyTorch 依赖项正常工作

English Document: SDAA429
Copyright © 2026 Texas Instruments Incorporated

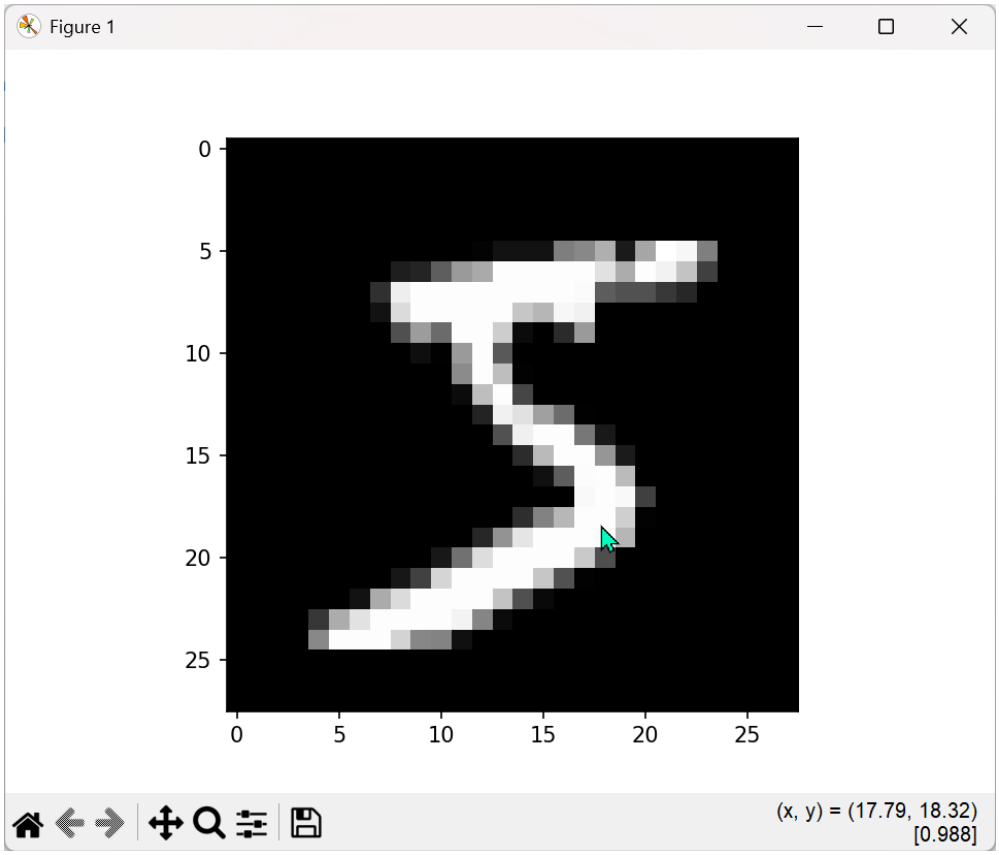


图 5-1. 手写数字图像

备注

MNIST 数据集已缩放至 [0, 1] 的浮点范围。

5.3 模型架构迭代重构

随着开发流水线的成熟，模型架构必须从基本的数学抽象演变为能够捕获复杂数据模式的复杂高性能结构。本节演示了从线性分类器到卷积神经网络 (CNN) 的过渡，重点介绍了 PyTorch 生态系统中模块化的强大功能。

5.3.1 基线实现

基于第一性原理实现一个最简线性分类器，以建立基线。该构造利用了基本的 PyTorch 张量运算，而不是像 *torch.nn* 这样的高级抽象，从而清晰地展示神经网络的数学机制。

该实现利用了两项核心 PyTorch 功能：

1. **张量运算：**手动管理权重和偏置，并将其作为张量使用，同时应用缩放初始化以确保数值稳定性。
2. **Autograd 引擎：**激活

```
requires_grad=True
```

，使 PyTorch 能够跟踪所有张量运算，构建用于反向传播期间自动梯度计算的动态计算图。

通过标准 Python 逻辑定义前向传播、激活函数 (log_softmax) 和损失函数 (NLL)，展示了 PyTorch 向量化后端完全优化自定义编码架构的能力。

English Document: SDAA429
Copyright © 2026 Texas Instruments Incorporated

示例脚本

下载 Tiny ML 工作流程示例 Python 脚本包，并使用其中包含的“03_test_train_from_scratch.py”脚本。使用终端执行脚本后，通过监测日志来验证训练进度：

```
Data loaded successfully. Training samples: 50000, Features per sample: 784
--- Pre-training Model State ---
...
Initial Loss: 2.2634
Initial Accuracy: 0.1406 (Expected near random)
------------------------------

--- Commencing Model Training ---
...
Epoch 10/10 completed.

--- Post-training Model Performance ---
Final Loss: 0.0559 (Target: Significant decrease)
Final Accuracy: 1.0000 (Target: Significant increase)
```

备注

观察到的 100% 准确率是在未使用单独验证集的情况下，仅在训练集上对模型进行评估的结果，表明可能存在**过拟合**。

5.3.2 使用 PyTorch 内置模块进行重构

此会话通过利用 PyTorch 的高级抽象（nn.Module 和 torch.optim）从手动张量操作过渡到标准化工作流程。这种重构将开发模式从低级数学实现转变为面向对象的编程 (OOP) 方法。

nn.Module

为提高模块化程度，模型封装在继承自 nn.Module 的类中。这提供了几个关键优势：

- **参数管理：***nn.Linear* 模块将自动处理参数初始化和配准，而不是手动定义和初始化 *self.weights* 和 *self.bias*。
- **自动跟踪：**通过从 nn.Module 继承，所有内部参数都会自动注册，使系统能够跟踪这些参数以进行梯度计算，而无需人工干预。

torch.optim

手动参数更新步骤（需要显式 **torch.no_grad()** 上下文和每个参数梯度归零）已被 **torch.optim** 框架取代

- **model.parameters()**：此方法可充当桥梁，将所有已注册的模型参数传递给优化器。优化器会维护这些参数指针的注册表以进行高效更新。
- **opt.step()**：该方法将更新逻辑抽象化。该方法迭代所有已注册的参数，并应用所选的优化算法（例如 SGD），以根据当前梯度更新权重。
- **opt.zero_grad()**：这提供了一种集中机制，可同时清除所有优化参数的梯度，从而保持下一次训练迭代的完整性。

使用优化器，可以通过修改一行代码实现不同优化算法之间的无缝切换（例如，从 SGD 切换到 Adam），而无需不改变底层模型架构。

示例脚本

下载 Tiny ML 工作流程示例 Python 脚本包，并使用其中包含的“04_test_pytorch_module_refactor.py”脚本。使用终端执行脚本后，通过监测日志来验证训练进度：

```
--- Pre-training Model State (Optimized Version) ---
Initial Loss: 2.3087
------------------------------

--- Commencing Training with torch.optim ---
...
Epoch 10/10 completed.

--- Post-training Performance ---
```


English Document: SDAA429
Copyright © 2026 Texas Instruments Incorporated

```
Final Training Loss: 0.0554
Validation Accuracy: 0.9245
```

5.3.3 数据管道优化

虽然前面各节依赖手动索引计算和对训练和标签张量的显式分片，但通过实现 Dataset 和 DataLoader 抽象，可以实现更强大的项目工作流程。这些抽象可以有效地将数据存储与批处理分离，从而提高可扩展性和效率。

TensorDataset

将特征和标签作为单独的实体管理需要仔细同步，以确保索引在重排或分片过程中保持对齐。TensorDataset 通过将多个张量“压缩”成单个数据集对象来解决此问题。一旦封装，访问索引会自动返回包含相应特征张量和标签的元组，从而在整个管道中保持结构完整性。

DataLoader

数据集处理存储时，DataLoader 负责管理数据交付的物流。DataLoader 提供了几种基本工程能力：

- **数据随机化 (shuffle = True)**：为防止模型学习数据序列（这会导致过拟合），DataLoader 会在每个轮次开始时重新打乱数据集，从而保持小批量选择的随机性。
- **并行加载 (num_workers)**：在 I/O（磁盘读取）成为瓶颈的大规模应用中，**DataLoader** 可以利用多进程数据加载。这使得 CPU 能够在 GPU 处理当前批次时在后台预取并准备下一批次，从而更大限度地提高硬件利用率。

通过利用这些工具，将训练循环从复杂的索引运算简化为简洁的可迭代接口。

示例脚本

下载 Tiny ML 工作流程示例 Python 脚本包，并使用其中包含的“05_test_data_pipeline_refactor.py”脚本。使用终端执行脚本后，通过监测日志来验证训练进度：

```
--- Commencing Training ---
Epoch 0: Val Loss = 0.4051 | Val Accuracy = 87.68%
Epoch 1: Val Loss = 0.3202 | Val Accuracy = 90.73%
Epoch 2: Val Loss = 0.2923 | Val Accuracy = 91.63%
Epoch 3: Val Loss = 0.2960 | Val Accuracy = 91.53%
Epoch 4: Val Loss = 0.2821 | Val Accuracy = 92.19%
Epoch 5: Val Loss = 0.3038 | Val Accuracy = 91.28%
Epoch 6: Val Loss = 0.3009 | Val Accuracy = 91.40%
Epoch 7: Val Loss = 0.3131 | Val Accuracy = 91.14%
Epoch 8: Val Loss = 0.2663 | Val Accuracy = 92.78%
Epoch 9: Val Loss = 0.3008 | Val Accuracy = 91.59%

--- Final Test Set Evaluation ---
>> Test Loss: 0.3143
>> Test Accuracy: 91.03%
```

5.3.4 过渡到 CNN 架构

借助强大的模块化训练基础设施，现在可以升级流水线，通过用卷积神经网络 (CNN) 代替线性架构来支持高级计算机视觉 (CV) 任务。

线性模型的主要限制是无法保留空间层次；将 28x28 图像平展成 784 元素向量会破坏像素之间的局部几何关系。CNN 架构通过两种机制解决了这一问题：

- **张量重塑：**使用 *xb.view(-1, 1, 28, 28)*，将输入重塑为四维张量（批次、通道、高度、宽度）。这将恢复卷积操作所需的空间网格。
- **分层特征学习：**模型使用 **nn.Conv2d** 层序列，通道深度不断增加。这使网络能够提取特征的层次结构：从第一层的低级边缘到最终卷积层中复杂高级的数字轮廓。

为提高收敛速度和稳定性，SGD 优化器升级为包含动量 = 0.9 的项。在标准 SGD 中，更新可能会在损失曲面的平坦区域中停滞或剧烈振荡。动量项引入了过去梯度的累积，提供了一种物理“惯性”，帮助优化器更高效地穿越平坦区域和狭窄的低谷区域。

由于训练逻辑（loss_batch 和 fit）与网络架构完全分离，因此这种过渡无需对核心训练循环进行修改。

English Document: SDAA429
Copyright © 2026 Texas Instruments Incorporated

示例脚本

下载 Tiny ML 工作流程示例 Python 脚本包，并使用其中包含的“06_test_cnn_model_refactor.py”脚本。使用终端执行脚本后，通过监测日志来验证训练进度：

```
--- Commencing CNN Training ---
Epoch 0: Val Loss = 0.3287 | Val Accuracy = 0.9031
Epoch 1: Val Loss = 0.2477 | Val Accuracy = 0.9233
Epoch 2: Val Loss = 0.2205 | Val Accuracy = 0.9352
Epoch 3: Val Loss = 0.2174 | Val Accuracy = 0.9335
Epoch 4: Val Loss = 0.1661 | Val Accuracy = 0.9522
Epoch 5: Val Loss = 0.1702 | Val Accuracy = 0.9488
Epoch 6: Val Loss = 0.1590 | Val Accuracy = 0.9537
Epoch 7: Val Loss = 0.1423 | Val Accuracy = 0.9606
Epoch 8: Val Loss = 0.1265 | Val Accuracy = 0.9622
Epoch 9: Val Loss = 0.1303 | Val Accuracy = 0.9611

--- Final Test Set Evaluation ---
>> Test Loss: 0.1389 | Test Accuracy: 0.9568 <<
```

使用 nn.Sequential 简化架构

以前实现的 CNN 架构使用 nn.Module 来定义定制层，*forward* 方法仍然需要显式手动定义数据流。这种方法虽然高度灵活，但需要手动编排类结构内的每个张量变换。

为简化此过程，PyTorch 提供了 nn.Sequential 容器。此实用程序允许开发人员将多个层封装到一个模块化单元中，数据按照定义的确切顺序流经这些层。这种“流水线”方法显著减少了样板代码，并且无需为标准前馈架构手动实施 **forward** 方法。

使用 nn.Sequential 时的一个常见挑战是，要求执行不属于标准原生层的中间张量操作，例如在后续处理之前执行自定义切片操作或调整特定形状。在标准 nn.Module 中，这可以在 *forward* 函数中轻松处理。但是，在严格的 nn.Sequential 流水线中，每个步骤都必须封装为有效的 nn.Module 对象。为实现这一点而无需为每个不同的转换编写全新的独立类，可以使用 Lambda 封装器。

示例脚本

下载 Tiny ML 工作流程示例 Python 脚本包，并使用其中包含的“07_test_cnn_sequential_refactor.py”脚本。使用终端执行脚本后，通过监测日志来验证训练进度：

```
--- Commencing Sequential CNN Training ---
Epoch 0: Val Loss = 0.3077 | Val Accuracy = 0.9077
Epoch 1: Val Loss = 0.2371 | Val Accuracy = 0.9303
Epoch 2: Val Loss = 0.1858 | Val Accuracy = 0.9456
Epoch 3: Val Loss = 0.1682 | Val Accuracy = 0.9503
Epoch 4: Val Loss = 0.1618 | Val Accuracy = 0.9546

--- Final Test Set Evaluation ---
>> Test Loss: 0.1694 | Test Accuracy: 0.9501 <<
```

5.4 模型导出

在训练阶段，模型达到所需的精度后，就可以从 PyTorch 开发环境转换为部署就绪格式 — 将模型转换为开放神经网络交换 (ONNX) 格式。

ONNX 是一种可互操作的中间表示，支持跨各种平台（包括 C++、Android、iOS 和 Web 浏览器）进行评估推理，可使用 *ONNX Runtime* 等运行时，而无需完全安装 PyTorch 的开销。

以下脚本演示了如何扩展现有的训练工作流程，以包括强大的模型导出步骤。该实现包括目录管理和特定配置，以确保导出的模型与下游推理引擎兼容。

示例脚本

下载 Tiny ML 工作流程示例 Python 脚本包，并使用其中包含的“08_test_export_onnx.py”脚本。使用终端执行脚本后，通过监测日志来验证训练进度：

```
--- Commencing Sequential CNN Training ---
Epoch 0: Val Loss = 0.3461 | Val Accuracy = 0.8950
```


English Document: SDAA429
Copyright © 2026 Texas Instruments Incorporated

```
Epoch 1: Val Loss = 0.3137 | Val Accuracy = 0.9107
Epoch 2: Val Loss = 0.2044 | Val Accuracy = 0.9445
Epoch 3: Val Loss = 0.1736 | Val Accuracy = 0.9490
Epoch 4: Val Loss = 0.1684 | Val Accuracy = 0.9502

--- Final Test Set Evaluation ---
>> Test Loss: 0.1698 | Test Accuracy: 0.9475 <<

--- Commencing ONNX Model Exportation ---
ONNX model successfully exported to: ...\model\mnist\mnist_model.onnx
```

5.5 量化感知训练

量化是在资源受限的边缘设备上部署机器学习模型的关键优化技术。量化是指将模型权重和激活值的精度从 32 位浮点数 (FP32) 降至较低的位宽，例如 8 位 (INT8)、4 位 (INT4) 或 2 位 (INT2) 整数。

此过程为边缘 AI 带来三个主要优势：

- **模型压缩：**显著减少存储器占用量（例如，从 FP32 转换为 INT8 可减少 4 倍）。
- **提升推理速度：**在专用硬件上，整数运算比浮点运算效率更高。
- **硬件加速：**TI 的神经处理单元 (NPU) 在架构上针对基于整数的计算进行了优化，因此量化成为实现高性能执行的先决条件。

随着位宽降低，模型效率提高，代价是可能会降低精度。表 5-1 量化期间观察到的典型性能特征。

表 5-1. 精度与效率权衡

精密	尺寸减小	速度提升	典型精度下降
Float32	1 倍（基线）	1 倍（基线）	0%
INT8	4x	2x – 4x	< 1 %
INT4	8x	3x – 6x	1% – 5%
INT2	16x	4x – 8x	5% – 15%

为了在低位宽下实现高精度，建议进行量化感知训练 (QAT)。QAT 会在训练过程中模拟量化误差，使模型权重能够适应降低的精度。有关更多详细信息，请访问

- Tiny ML Tensorlab 用户指南：量化
- Tiny ML Tensorlab 用户指南：独立量化示例

示例脚本

下载 Tiny ML 工作流程示例 Python 脚本包，并使用其中包含的“09_test_cnn_quantization_model.py”脚本。该脚本演示了完整的处理流程：从 FP32 训练到 INT8 量化感知训练，最后到 ONNX 导出。

以下日志显示了脚本的完整执行过程，包括基线 FP32 训练、后续的量化感知训练 (QAT) 阶段，以及量化模型的成功导出。

```
--- Stage 1: FP32 Training ---
--- Commencing Model Training ---
Epoch 0: Val Loss = 0.3535 | Val Accuracy = 0.8992
Epoch 1: Val Loss = 0.2269 | Val Accuracy = 0.9410
Epoch 2: Val Loss = 0.2205 | Val Accuracy = 0.9359
Epoch 3: Val Loss = 0.1928 | Val Accuracy = 0.9449
Epoch 4: Val Loss = 0.1590 | Val Accuracy = 0.9593
--- Final Test Set Evaluation ---
>> Test Loss: 0.1732 | Test Accuracy: 0.9522 <<

--- Stage 2: INT8 QAT ---
--- Commencing Model Training ---
Freezing BN for subsequent epochs
Epoch 0: Val Loss = 0.1289 | Val Accuracy = 0.9617
Epoch 1: Val Loss = 0.1197 | Val Accuracy = 0.9641
Freezing ranges for subsequent epochs
Epoch 2: Val Loss = 0.1499 | Val Accuracy = 0.9530
Epoch 3: Val Loss = 0.1278 | Val Accuracy = 0.9623
Epoch 4: Val Loss = 0.0985 | Val Accuracy = 0.9713
```


English Document: SDAA429
Copyright © 2026 Texas Instruments Incorporated

```
--- Final Test Set Evaluation ---
>> Test Loss: 0.1041 | Test Accuracy: 0.9686 <<

--- Stage 3: Exporting to ONNX ---
ONNX model successfully exported: ...\model\mnist\mnist_model_int8.onnx
```

5.6 模型编译

模型经过训练并导出为 ONNX 格式后，开发生命周期的最后一步是编译。此过程可将高级 ONNX 计算图转换为针对目标微控制器 (MCU) 进行优化的机器可读代码。

TI 神经网络编译器 (NNC) 用于弥合通用 ONNX 模型与嵌入式器件之间的差距。NNC 根据您的项目要求将模型转换为两种主要部署格式之一：

- **静态库 (.a)：**使用交叉编译器（例如 tiarmclang）将该模型编译到静态库中。这是标准生产方法，其中该库直接链接到 Code Composer Studio™ (CCS) 项目中。
- **源代码包 (.tar)：**该模型被解压缩到人类可读的 C 源文件中。此方法非常适合深度检查生成的内核或限制直接链接二进制文件的环境。此模式在编译阶段不需要外部交叉编译器。

输出文件结构

编译器生成的输出文件遵循标准化的目录结构：

```
root_path/
├── lib0.c                # Optimized source code lib0
├── lib1.c                # Optimized source code lib1
├── lib2.c                # Optimized source code lib2
├── mod.a                 # Compiled static library file (Link this to CCS)
└── tvmgen_default.h      # C header file containing inference function signatures
```

命令行界面 (CLI)

对于高级用户，*tvmc* 命令行界面提供了对编译过程的精细控制。有关详细文档，请参阅 TI MCU 神经网络编译器用户指南：编译说明。

要将 ONNX 模型编译为 MSPM0 (Cortex-M0+) 目标的静态库，请在终端中执行以下命令：

```
tvmc compile \
    --target="c, ti-npu skip_normalize=true output_int=true" \
    --target-c-mcpu=cortex-m0plus \
    .\model.onnx \
    -o artifacts_m0/mod.a \
    --cross-compiler="tiarmclang" \
    --cross-compiler-options="-Os -mcpu=cortex-m0plus -march=thumbv6m -mtune=cortex-m0plus -mthumb
-mfloat-abi=soft -I. -Wno-return-type"
```

示例脚本

下载 Tiny ML 工作流程示例 Python 脚本包，并使用其中包含的“10_test_cnn_quan_compile_npu_lib.py”脚本。该脚本围绕 *tvmc* 命令行界面提供了一个通用封装器。该脚本使开发人员能够轻松管理目标特定编译器标志。

使用终端执行脚本后，通过监测日志来验证训练进度：

```
================================================================
Output Mode    : LIB
Target Device  : MSPM0 (cortex-m0plus)
NNC Target     : c, ti-npu skip_normalize=true output_int=true
Input Model    : C:\ti\tinyml-tensorlab\UserTestFolder\model\mnist\mnist_model_int8.onnx
Output Dir     : C:\ti\tinyml-tensorlab\UserTestFolder\model\mnist\outputLib
================================================================
WARNING:ti_mcu_nnc:GCONV layer NOT offloaded to TI-NPU
WARNING:ti_mcu_nnc:GCONV layer config: 8-bit input [9, 9, 16], 8-bit output [4, 4, 10], 8-bit
kernel [3, 3], strides [2, 2], padding [(0, 0), (0, 0)], data layout NHWC, kernel layout HWIO
WARNING:ti_mcu_nnc:GCONV does not support 10 output channels. Please change to multiples of 4.
WARNING:ti_mcu_nnc:AVGPOOL layer NOT offloaded to TI-NPU
WARNING:ti_mcu_nnc:Global AVGPOOL layer config:8-bit input [4, 4, 10], 8-bit output [10]
```


English Document: SDAA429
Copyright © 2026 Texas Instruments Incorporated

```
WARNING:ti_mcu_nnc:Global AVGPOOL does not support 10 input channels. Please change to multiples of
4.
INFO:ti_mcu_nnc: ============================================================
INFO:ti_mcu_nnc: TI NPU Offloading Report
INFO:ti_mcu_nnc: ============================================================
INFO:ti_mcu_nnc:
INFO:ti_mcu_nnc: LAYER/OPERATOR COUNT SUMMARY
INFO:ti_mcu_nnc: ============================================================
INFO:ti_mcu_nnc:
INFO:ti_mcu_nnc: Layer Patterns Offloaded
INFO:ti_mcu_nnc: ------------------------------------------------------------
INFO:ti_mcu_nnc: Layer Pattern                                      Count
INFO:ti_mcu_nnc: ------------------------------------------------------------
INFO:ti_mcu_nnc: FCONV                                              1
INFO:ti_mcu_nnc: PAD                                                2
INFO:ti_mcu_nnc: GCONV                                              1
INFO:ti_mcu_nnc:
INFO:ti_mcu_nnc: Operators not Offloaded
INFO:ti_mcu_nnc: ------------------------------------------------------------
INFO:ti_mcu_nnc: Operator                                           Count
INFO:ti_mcu_nnc: ------------------------------------------------------------
INFO:ti_mcu_nnc: reshape                                            1
INFO:ti_mcu_nnc: nn.pad                                             2
INFO:ti_mcu_nnc: qnn.conv2d                                         1
INFO:ti_mcu_nnc: cast                                               4
INFO:ti_mcu_nnc: nn.bias_add                                        1
INFO:ti_mcu_nnc: multiply                                           2
INFO:ti_mcu_nnc: right_shift                                        2
INFO:ti_mcu_nnc: clip                                               2
INFO:ti_mcu_nnc: sum                                                1
INFO:ti_mcu_nnc: add                                                1
INFO:ti_mcu_nnc:
INFO:ti_mcu_nnc: ============================================================
INFO:ti_mcu_nnc:
INFO:ti_mcu_nnc:
INFO:ti_mcu_nnc: MODEL PARTITION - TOPOLOGICAL VIEW
INFO:ti_mcu_nnc: ============================================================
INFO:ti_mcu_nnc: Layer/Operator                                     Offloaded
INFO:ti_mcu_nnc: ------------------------------------------------------------
INFO:ti_mcu_nnc: Input
INFO:ti_mcu_nnc: reshape                                            [ ]
INFO:ti_mcu_nnc: nn.pad                                             [ ]
INFO:ti_mcu_nnc: nn.pad                                             [ ]
INFO:ti_mcu_nnc: FCONV                                              [X]
INFO:ti_mcu_nnc: PAD                                                [X]
INFO:ti_mcu_nnc: GCONV                                              [X]
INFO:ti_mcu_nnc: PAD                                                [X]
INFO:ti_mcu_nnc: qnn.conv2d                                         [ ]
INFO:ti_mcu_nnc: cast                                               [ ]
INFO:ti_mcu_nnc: nn.bias_add                                        [ ]
INFO:ti_mcu_nnc: multiply                                           [ ]
INFO:ti_mcu_nnc: right_shift                                        [ ]
INFO:ti_mcu_nnc: clip                                               [ ]
INFO:ti_mcu_nnc: cast                                               [ ]
INFO:ti_mcu_nnc: cast                                               [ ]
INFO:ti_mcu_nnc: sum                                                [ ]
INFO:ti_mcu_nnc: add                                                [ ]
INFO:ti_mcu_nnc: multiply                                           [ ]
INFO:ti_mcu_nnc: right_shift                                        [ ]
INFO:ti_mcu_nnc: clip                                               [ ]
INFO:ti_mcu_nnc: cast                                               [ ]
INFO:ti_mcu_nnc: Output
INFO:ti_mcu_nnc: ============================================================
INFO:ti_mcu_nnc:
INFO:ti_mcu_nnc: ============================================================
INFO:ti_mcu_nnc: TI Model Library Memory Usage (mod.a)
INFO:ti_mcu_nnc: ============================================================
INFO:ti_mcu_nnc: Code:                      1952 bytes (    1.91 KB)
INFO:ti_mcu_nnc: RO Data:                   7061 bytes (    6.90 KB)
INFO:ti_mcu_nnc: RW Data:                   9540 bytes (    9.32 KB)
INFO:ti_mcu_nnc: Total:                    18553 bytes (   18.12 KB)
INFO:ti_mcu_nnc: ============================================================
```


English Document: SDAA429
Copyright © 2026 Texas Instruments Incorporated

6 边缘 AI 应用：数字识别

数字识别技术是一项基础功能，在嵌入式和个人消费市场中具有广泛的应用。然而，在嵌入式系统上部署可靠的识别功能面临重大挑战：传统模板匹配方法难以应对人类手写笔迹的自然变异性。边缘 AI 的出现提供了一种颇具吸引力的设计方案，能够直接在设备本身上实现准确、实时的推理，无需依赖云端连接。基于 CPU 的推理流水线需要大量的计算资源，超出了资源受限微控制器通常可提供的处理能力。

MSPM0G5187 微控制器通过集成专用的 TinyEngine NPU 来解决这一挑战，该 NPU 旨在以极低的功耗和内存开销加速机器学习工作负载。该设计演示了将单个数字字符图像分类为 10 个类别 (0-9) 之一的功能。应用通过 UART 反向通道接收 28 × 28 灰度图像作为像素数据，使用片上 NPU 进行硬件加速的模型推理，并将识别出的数字类别发送回主机 GUI 进行显示。

借助片上 NPU，该数字识别系统实现了约 99% 的分类准确率，推理延迟仅为 6.05ms，同时仅占用 73KB 的闪存和 10.9KB 的 RAM，如表 6-1 所示。这些结果证明了在资源受限的微控制器上进行器件端机器学习的可行性。

表 6-1. 数字识别边缘 AI 设计性能

Metric	值
精度	约 99%
闪存使用情况	73KB
RAM 使用情况	10.9KB
推理延迟 (NPU)	6.05ms
推理能量(1) (AVG)	170.38uJ

(1) 推理能量是在重复执行 tvmgen_default_run() 期间使用 EnergyTrace™ 软件测量的。

备注

要在 LP-MSPM0G5187 上保持准确的推理能量测量，请移除 J6 跳线。这可以防止板载麦克风和音频 ADC 产生额外的电流消耗，否则可能会导致结果出现偏差。

表 6-2 显示了模型主要信息。

表 6-2. 数字识别 AI 模型信息

属性	值
模型架构	CNN
参数数量	60,000
输入形状	(1、28、28)
输出类	10
量化	INT8

6.1 LeNet-5 变体 CNN 模型

LeNet 是一系列开创性的卷积神经网络 (CNN) 架构，最初设计用于从小尺寸的灰度图像中识别手写数字和字符。数字识别边缘 AI 模型是通过 TI 神经网络编译器编译为 NPU 原生指令的 LeNet-5 变体 CNN 模型。

图 6-1 展示了其工作流程。该设计接受一个扁平化的 28×28 灰度图像作为 784 个元素的 float32 输入，并输出一个包含 10 个元素的 float32 值，表示每个类别的置信度分数。该网络由两个连续的卷积块组成，每个块包含一个带偏置的 Conv2D 层、ReLU 激活函数和 2×2 最大池化。这些模块在每个阶段逐步提取空间特性，同时将空间分辨率减半。生成的特征图随后被展平，并通过一系列全连接层，将高维特性映射到 10 类输出。由于所有层均使用 INT8 量化权重和激活操作来运行，在最后一个全连接层之后会执行去量化后处理步骤，将输出重新缩放为 float32 类型。预测的数字由主机端对 10 个输出分数取 argmax 来确定。

Copyright © 2026 Texas Instruments Incorporated

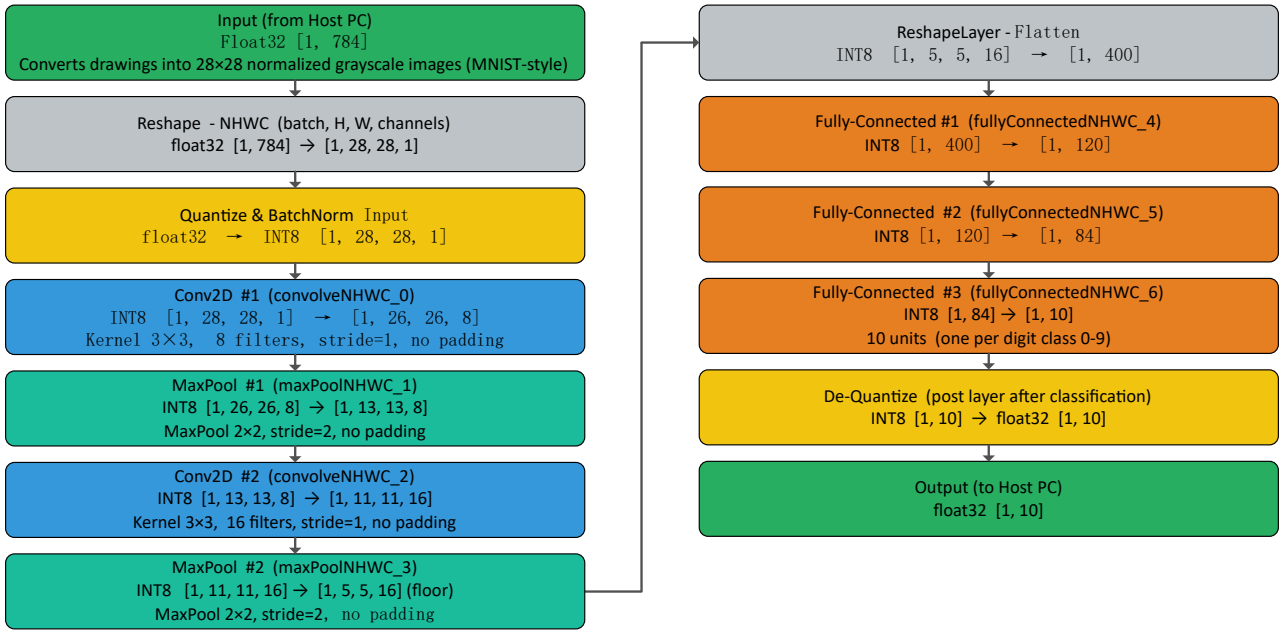


图 6-1. 数字识别 AI 模型流程图

6.2 NPU/CPU 性能比较

边缘 AI 模型可通过 TI 神经网络编译器部署到硬件上，目标可以是专用的硬件 NPU，也可以是主机 CPU。TinyEngine NPU 是一种专门设计用于执行神经网络计算的专用硬件加速器，能够在保证高效运算的同时，相比通用 CPU 大幅减少推理延迟并节能。

为便于性能评估，MSPM0 SDK 同时提供了基于 NPU 和基于 CPU 的实现示例，供用户进行基准测试，参见：

- 基于 NPU 的数字识别边缘 AI 示例
- 基于 CPU 的数字识别边缘 AI 示例

表 6-3 总结了基于 NPU 和 CPU 的边缘 AI 设计在数字识别应用中的性能对比。

表 6-3. NPU/CPU 性能比较

性能指标	基于 NPU 的设计	基于 CPU 的设计
精度	约 99%	约 99%
闪存使用情况	73KB	68KB
RAM 使用情况	10.9KB	8.1KB
推理延迟	6.05ms	89.81ms
推理能量 (AVG)	170.38uJ	2,447.45uJ

基于 NPU 的设计相比基于 CPU 的实现，推理延迟大约降低 15 倍，且每次推理的能耗有效减少约 96%，使其成为对功耗敏感的边缘应用场景中极具效率的选择。

English Document: SDAA429
Copyright © 2026 Texas Instruments Incorporated

7 边缘 AI 应用：波形分类器

MSPM0 提供了一种实验性波形分类器设计，能够对输入信号进行实时、连续的检测和分类，识别正弦波、三角波和锯齿波等常见波形类型。该设计基于一个 13K 参数的时间序列分类模型，具有较强的可扩展性和可移植性。

该波形分类器系统实现了 100%（R 平方）的分类准确率，推理延迟仅为 5.84ms，同时仅占用 21.5KB 的闪存和 3.3KB 的 RAM，如表 7-1 所示。

表 7-1. 波形分类器边缘 AI 解决方案性能

Metric	值
精度	100%
闪存使用情况	21.5KB
RAM 使用情况	3.3KB
推理延迟 (NPU)	5.84ms
推理能量(1) (AVG)	236.97uJ

(1) 推理能量是在重复执行 *tvmgen_default_run()* 期间使用 EnergyTrace 测量的。

备注

要在 LP-MSPM0G5187 上保持准确的推理能量测量，请移除 J6 跳线。这可以防止板载麦克风和音频 ADC 产生额外的电流消耗，否则可能会导致结果出现偏差。

表 7-2 显示了模型主要信息。

表 7-2. 波形分类器 AI 模型信息

属性	值
模型架构	CNN
参数数量	14,124
输入形状	张量 [(1, 1, 128, 1)]
输出类	3（锯齿波、正弦波、方形波）
量化	INT8

Copyright © 2026 Texas Instruments Incorporated

7.1 特征提取

特性提取是常见 EdgeAI 应用的关键组成部分，尤其是在音频和时间序列信号处理中。特性提取模块提供针对嵌入式系统的高效信号处理功能，利用 ARM 库以实现最佳性能。可用的特征提取函数包括：

- 实数快速傅里叶变换 (RFFT)：将时域信号转换为频域表示
- 幅度缩放：应用缩放将信号幅度归一化到适当范围，并防止后续处理中出现上溢/下溢
- 复数幅度计算：计算 FFT 所得复数的幅度
- 直流分量去除：消除信号直流偏置，以提高特征准确性
- 频带计算：对定义频带内的频率分量求平均，以实现降维

图 7-1 展示了特性提取处理流水线。

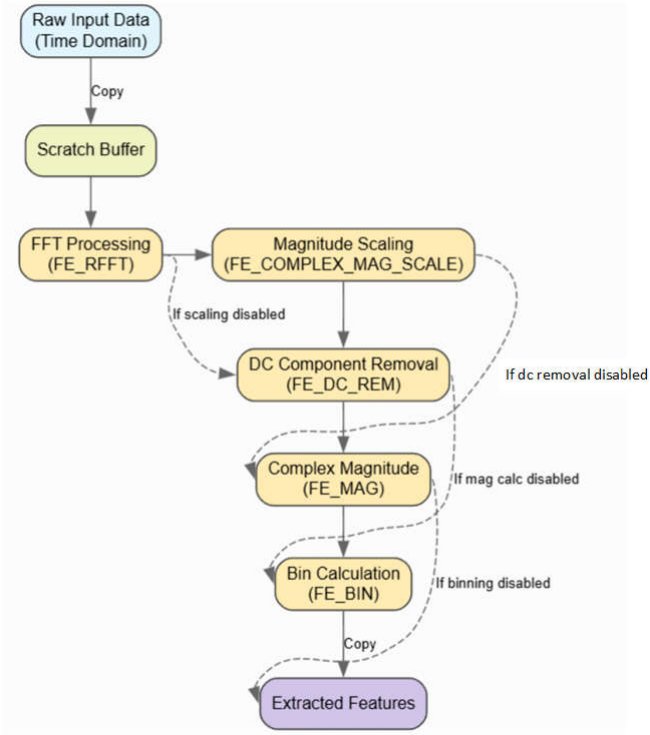


图 7-1. 特性提取处理流水线

表 7-3 展示了专为波形分类器设计的特性提取处理流水线配置。当相应的配置符号被定义时，其关联的任务将在处理流水线内执行。

表 7-3. 波形分类器的流水线配置

正在处理任务	配置符号	是否已定义
RFFT	FE_RFFT	Y
幅度缩放	FE_COMPLEX_MAG_SCALE	Y
直流分量去除	FE_DC_REM	Y
复幅度计算	FE_MAG	Y
频带计算	FE_BIN	Y

English Document: SDAA429
Copyright © 2026 Texas Instruments Incorporated

表 7-4 展示了专为波形分类器设计的特性提取配置参数。对于每个推理任务，将八个帧进行拼接，每个帧处理 256 个数据点以生成 16 个特征维度，总共产生 128 个特征维度输入到波形分类器 AI 模型中。

表 7-4. 波形分类器特征提取参数

Metric	配置符号	值
输入帧大小	FE_FRAME_SIZE	256
每帧特性大小	FE_FEATURE_SIZE_PER_FRAM	16
频率频带大小	FE_BIN_SIZE	8
归一化频带值	FE_BIN_NORMALIZE	1
要拼接的帧	FE_NUM_FRAME_CONCAT	8
输入特性大小	FE_STACKING_FRAME_WIDTH	128

备注

表 7-4 中配置的特性提取参数必须与模型训练期间使用的参数一致，以确保结果的可比性。

性能注意事项

- 优化的计算：利用 ARM 优化的库进行高效处理
- 存储器效率：缓冲区交换可更大限度地减少存储器复制开销
- 降维：频带计算可降低特征维度，从而降低后续阶段的存储器使用和计算量
- 参数调优：配置参数必须根据可用的存储器和计算资源进行调优

Copyright © 2026 Texas Instruments Incorporated

7.2 时序分类模型

时序分类用于识别时序数据中的特定类别、运行状态或预定义标签，广泛适用于医疗健康、无线监控、智能安防和自动化测试等领域。

本波形分类器设计实现了一个轻量级、含 13K 参数的时间序列分类模型 (CLS_13k_NPU)，专为在嵌入式神经处理单元 (NPU) 上高效部署而设计。该设计采用 256 点实数快速傅里叶变换 (RFFT) 特征提取流水线，并结合每次 AI 推理任务中的 8 帧拼接，接受 128 元素的一维时间序列特征向量作为输入，输出 3 元素的输出向量，其中每个元素表示对应类别的置信度得分。

导出的模型文件从最后一个全连接层输出 INT8（8 位量化）张量。由于原始输出为量化形式的每类置信度得分，因此需要后处理步骤才能得到最终的分类结果。

波形分类器模型架构

模型网络结构为顺序式的特征变换流水线，其中每个阶段都直接构建在前一阶段的输出之上，逐步精炼并抽象输入表示，最终完成分类决策。图 7-2 展示了工作流程。

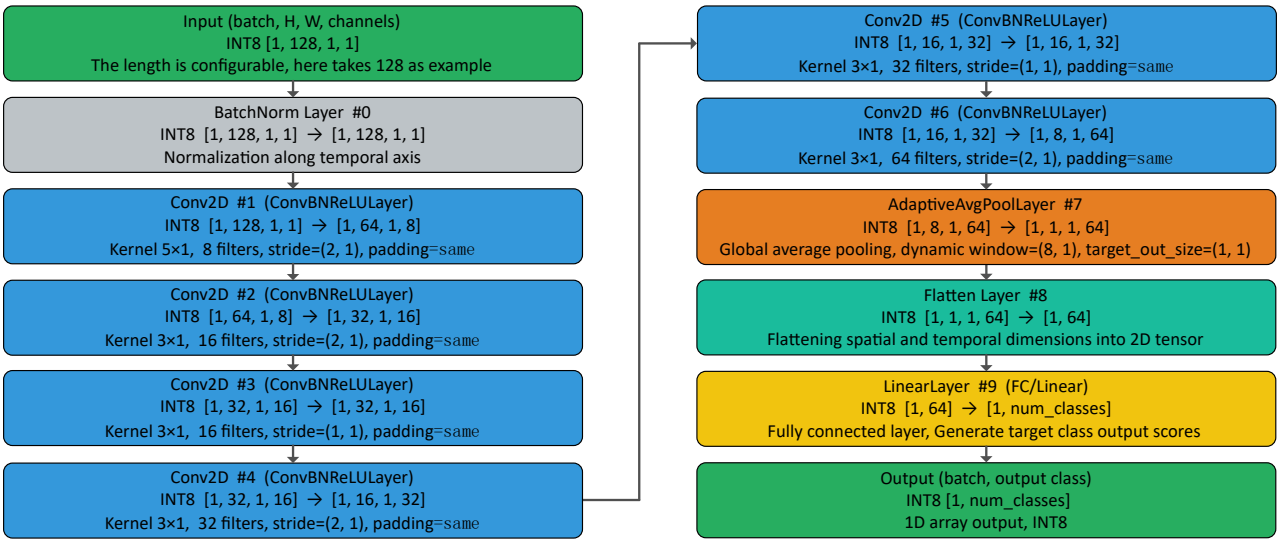


图 7-2. 时间序列分类模型 (CLS_13k) 流程图

流水线起始于批量归一化层，该层对原始输入序列进行实时标准化处理，确保在任何可学习的变换应用之前，输入信号幅值变化不会影响特征缩放的一致性。输入稳定后，网络通过六个连续的卷积块逐步加深特征表示。每个卷积块内部结构一致：一个带偏置的 Conv2D 层，后接批量归一化和 ReLU 激活，形成重复模式，从而在连续的阶段中捕获越来越抽象的时间模式。每个卷积块内的批量归一化有助于在深度增加时维持训练稳定性，而 ReLU 激活则为模型学习复杂信号特征提供了必要的非线性。

经过六个卷积阶段后，所得特征图通过一个自适应平均池化层，将空间分辨率压缩至固定大小 1×1，而与之前的时间长度无关。这一设计尤为关键：通过将模型与固定时序维度解耦，该模型本质上也支持不同长度的时序输入，为未来的部署场景提供了灵活性，无需修改网络架构。

固定尺寸的池化特征图随后被展平为二维张量，并送入最后的全连接 (FC) 层，该层将压缩后的高维特征表示直接映射到目标类别的输出得分。

模型输出和后处理

导出的模型文件从最后一个全连接层输出 INT8（8 位量化）张量。由于原始输出为量化形式的每类置信度得分，因此需要后处理步骤才能得到最终的分类结果。通常，这是通过在输出向量上应用 argmax 操作来实现的，该操作会识别出置信度分数最高的类别索引作为预测的分类结果。

English Document: SDAA429
Copyright © 2026 Texas Instruments Incorporated

7.3 模型内存注意事项

在资源受限的嵌入式平台上开发边缘 AI 设计，需要平衡一个严格的三维优化组合：算法性能（准确性、延迟）、非易失性存储 (FLASH/ROM) 和运行时存储器 (SRAM)。

- **算法性能**：包括推理准确性和延迟，这两者主要由模型架构决定（包括网络深度、层宽度和算子复杂度）。
- **静态存储器（闪存/ROM）**：主要由模型的总参数数量（权重和偏置）决定。在完全量化的 INT8 流水线中，每个参数直接对应一个字节的闪存存储。因此，通道宽度扩展的更深网络会线性地增加静态二进制文件大小。有时，输入特征维度的变化也会影响静态存储器使用（例如，在全连接层中）。
- **动态运行时存储器 (SRAM/RAM)**：由输入特征大小以及中间层产生的激活张量（特征图）决定。当时间序列切片在网络中传播时，处理核心必须分配临时工作区来存储层的输入和输出。更长的输入时间窗口或更高的特性维度会指数级地增加峰值运行时 RAM 需求。

受这些硬件现实驱动，边缘部署迫使人们摆脱传统的“准确性优先”思维。相反，成功的部署需要进行细致的权衡，将原始模型性能直接与静态闪存和峰值动态 SRAM 的硬性物理边界相平衡。

为了说明这种关系，表 7-5 量化了波形分类器在不同模型大小和输入特征配置下的存储器占用和资源利用情况（使用 Edge AI Studio 1.6.1 云版本进行测试）。

表 7-5. 模型内存分析

模型变体（参数）	闪存 (ROM) 大小	输入 = 64 时的 RAM 大小	输入 = 128 时的 RAM 大小	输入 = 256 时的 RAM 大小
TimeSeries_Generic_1K	约 5.6KB	约 2.7KB	约 5.3KB	约 10.4KB
TimeSeries_Generic_4K	约 9.9KB	约 1.2KB	约 1.7KB	约 2.7KB
TimeSeries_Generic_13K	约 21.4KB	约 2.3KB	约 3.3KB	约 5.3KB

在所有输入维度下，最小的模型 (TimeSeries_Generic_1K) 所消耗的运行时 RAM 始终是较大模型（TimeSeries_Generic_4K 和 TimeSeries_Generic_13K）的两到四倍。这种看似不合常理的行为源于参数数量和运行时内存是由截然不同的架构决策所驱动的。1K 模型采用浅层拓扑结构，缺少早期下采样，导致大幅值、高分辨率的特征图在整个中间层持续占用存储器。相比之下，4K 和 13K 模型采用更深层的架构，并在网络前端引入带步长的卷积，能够立即缩减时间维度，从而显著减小中间运行时张量的大小。

Copyright © 2026 Texas Instruments Incorporated

7.4 NPU/CPU 性能比较

边缘 AI 模型可通过 TI 神经网络编译器部署到硬件上，目标可以是专用的硬件 NPU，也可以是主机 CPU。为便于性能评估，MSPM0 SDK 同时提供了基于 NPU 和基于 CPU 的实现示例，供用户进行基准测试，参见：

- 基于 NPU 的波形分类器边缘 AI 示例
- 基于 CPU 的波形分类器边缘 AI 示例

表 7-6 总结了波形分类器应用中基于 NPU 与基于 CPU 的边缘 AI 设计之间的性能对比。

表 7-6. NPU/CPU 性能比较

性能指标	基于 NPU 的设计	基于 CPU 的设计
精度	100%	100%
闪存使用情况	21.5KB	21.7KB
RAM 使用情况	3.3KB	3.1KB
特性提取延迟	10.75ms	10.93ms
特征提取能量(1) (AVG)	386.68uJ	393.15uJ
推理延迟	5.77ms	54.42ms
推理能量 (AVG)	236.97uJ	2039.20uJ

(1) 特征提取能量是在重复执行 FE_process() 期间使用 EnergyTrace 测量的。

在硬件计算效率的对比评估中，NPU 加速架构在性能和功耗指标上均显著优于基准 CPU 实现方案。

在纯推理阶段，基于 NPU 的设计展现出卓越的加速效果，执行延迟降低约 8 倍，同时单次推理能耗降低约 88%。

若将前端特性提取纳入端到端系统分析，由于信号预处理在 CPU 上产生的确定性工作负载，整体效率提升会呈现局部衰减；即便如此，基于 NPU 的流水线仍能保持约 3 倍的延迟压缩，并将单次执行能耗降低约 74%。

Copyright © 2026 Texas Instruments Incorporated

8 总结

本应用手册全面介绍了基于 MSPM0G5187 微控制器的边缘 AI 解决方案，涵盖硬件特性、软件工具链以及代表性应用示例。

具体而言，本文档介绍了两种优化的工作流程：用于简化模型开发的 TI Edge AI Studio 和用于高级模型工程的 TI Tiny ML Tensorlab。

通过两个应用验证了该平台的功能：数字识别设计实现了约 99% 的分类准确率，且硬件 NPU 相比基于 CPU 的实现，推理延迟降低 15 倍，能效提升 96%；波形分类器实现了 100% 的分类准确率，NPU 推理延迟降低 8 倍，推理能效提升 88%。

这些结果证实，搭载集成式 TinyEngine NPU 的 MSPM0G5187 是面向资源受限嵌入式器件高效部署 AI 的极具吸引力的平台。

9 参考资料

1. 德州仪器 (TI)，具有 128kB 闪存、32kB SRAM、USB 2.0 FS、I2S、ADC 和 TinyEngine™ NPU 的 80MHz Arm® Cortex®-M0+ MCU，产品页面。
2. 德州仪器 (TI)，*MSPM0 G 系列 80MHz 微控制器*，技术参考手册。
3. 德州仪器 (TI)，MSPM0G5187 LaunchPad™ 开发套件评估模块，产品页面。
4. 德州仪器 (TI)，*边缘 AI 加速的 Arm® Cortex®-M0+ MCU 如何为电子产品注入更强智能*，技术文章。
5. 德州仪器 (TI)，*TI 的 TinyEngine™ NPU 为更多嵌入式系统解锁边缘 AI 加速能力*，产品页面。
6. 德州仪器 (TI)，边缘 AI 传感器 BoosterPack™ 参考设计，产品页面。
7. 德州仪器 (TI)，使用 TI 微控制器 (MCU) 进行数字识别，产品页面。
8. 德州仪器 (TI)，CCStudio™ 边缘 AI 软件开发工具，产品页面。
9. 德州仪器 (TI)，Tiny ML Tensorlab GitHub 存储库，网页。
10. 德州仪器 (TI)，Tiny ML Tensorlab 用户指南，网页。
11. 德州仪器 (TI)，TI MCU 神经网络编译器用户指南，网页。
12. 德州仪器 (TI)，*MSPM0 边缘 AI 用户指南*，资源管理器。
13. 德州仪器 (TI)，*在 MSPM0 低成本微控制器上利用边缘 AI 进行交流电弧故障检测*，应用手册。
14. 德州仪器 (TI)，Tiny ML 工作流程示例 Python 脚本包，E2E™ 支持讨论帖。
15. PyTorch，What is torch.nn really?，网页。

10 修订历史记录

注：以前版本的页码可能与当前版本的页码不同

Changes from Revision * (June 2026) to Revision A (October 2026)	Page
• 更新了 *简介* 部分	3
• 更新了 *边缘 AI 工具链* 一节	5
• 向 *边缘 AI 工具链* 一节添加了图 3-1	5
• 更新了 *TI Edge AI Studio* 一节	5
• 更新了 *TI Tiny ML Tensorlab* 一节	7
• 更新了 *TI 神经网络编译器* 一节	7
• 添加了 *通过 TI Edge AI Studio 进行快速原型设计* 一节	9
• 添加了 *项目设置* 一节	9
• 添加了 *数据捕获* 一节	11
• 添加了 *硬件设置* 一节	11
• 添加了 *特征概述* 一节	12
• 添加了 *数据导入* 一节	13
• 添加了 *分类任务* 一节	13
• 添加了 *回归任务* 一节	14
• 添加了 *预测任务* 一节	15

Copyright © 2026 Texas Instruments Incorporated

- 添加了*注释列表*一节 15
- 添加了*数据注释*一节 16
- 添加了*模型训练*一节 17
- 添加了*训练参数*一节 19
- 添加了*工具链依赖项*一节 20
- 添加了*性能报告*一节 20
- 添加了*模型导出*一节 23
- 添加了*通过 TI Tiny ML Tensorlab 进行高级开发*一节 25
- 添加了*环境设置*一节 25
- 添加了*数据采集*一节 26
- 添加了*数据检索*一节 26
- 添加了*数据验证*一节 26
- 添加了*模型架构迭代重构*一节 28
- 添加了*基线实现*一节 28
- 添加了*使用 PyTorch 内置模块进行重构*一节 29
- 添加了*数据管道优化*一节 30
- 添加了*过渡到 CNN 架构*一节 30
- 添加了*模型导出*一节 31
- 添加了*量化感知训练*一节 32
- 添加了*模型编译*一节 33
- 更新了*边缘 AI 应用：数字识别*一节 35
- 更新了 *LeNet-5 变体 CNN 模型*一节 35
- 更新了“*NPU/CPU* 性能比较”一节和表 6-3 36
- 在*边缘 AI 应用：波形分类器*一节中添加了注释 37
- 更新了*特征提取*一节中表 7-4 的标题 38
- 更新了*时间序列分类模型*一节 40
- 更新了*模型存储器注意事项*一节中的文本和表 7-5 41
- 更新了 *NPU/CPU 性能比较*一节中的文本和表 7-6 42
- 更新了*总结*一节 43
- 更新了*参考资料*部分 43

English Document: SDAA429
Copyright © 2026 Texas Instruments Incorporated

重要通知和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、与某特定用途的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他安全、安保法规或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的相关应用。严禁以其他方式对这些资源进行复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。对于因您对这些资源的使用而对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，您将全额赔偿，TI 对此概不负责。

TI 提供的产品受 TI 销售条款)、TI 通用质量指南 或 ti.com 上其他适用条款或 TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。 除非德州仪器 (TI) 明确将某产品指定为定制产品或客户特定产品，否则其产品均为按确定价格收入目录的标准通用器件。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

版权所有 © 2026，德州仪器 (TI) 公司

最后更新日期：2025 年 10 月