参考資料

Application Note

エッジ人工知能アプリケーションでの *MSPM0G5187* の使用方法

Sal Ye

概要

このアプリケーション ノートでは、MSPM0G5187 マイコンにエッジ人工知能 (AI) アプリケーションを実装する方法について説明します。このマイコンは、TinyEngine™ ニューラル プロセッシング ユニット (NPU) を搭載しており、リソースに制約のある組込みシステムで効率的なニューラル ネットワーク推論を実現するよう設計された専用のハードウェア アクセラレータです。この資料では、MSPM0G5187 のハードウェアアーキテクチャの概要と NPU の主な機能について説明し、その後に TI のエッジ AI ソフトウェア エコシステムについての包括的な紹介が続きます。その中には、CCSTUDIO™ Edge AI Studio、Tiny ML Tensorlab、ニューラル ネットワーク コンパイラ (NNC) などが含まれます。次に、MSPM0G5187 のエッジ AI 機能を示すために、2 つの代表的なアプリケーション例を紹介します。1 つは、LeNet–5 CNN モデルに基づく手書きの数字を認識する設計、もう 1 つは時系列 CNN モデルに基づく波形分類の設計です。NPU/CPU ベースの推論の実装を比較した定量的性能ベンチマークが提供されており、推論レイテンシとエネルギー効率における NPU ハードウェア アクセラレーションの大きな利点が示されています。これらの結果は、組込みプラットフォームでエッジ AI 設計を開発するエンジニアにとって、実践的なガイダンスと設計リファレンスになります。

目次

1 概要 3
2 TinyEngine™ NPU を搭載した MSPM0G5187 4
3 エッジ AI ツールチェーン 5
3.1 TI Edge AI Studio 6
3.2 TI Tiny ML Tensorlab 7
3.3 TI ニューラル ネットワーク コンパイラ 7
4 TI Edge AI Studio による迅速なプロトタイピング 9
4.1 プロジェクトの設定 9
4.2 データ キャプチャ 11
4.3 データのインポート 13
4.4 データ アノテーション 16
4.5 モデルのトレーニング 17
4.6 パフォーマンス レポート 21
4.7 モデルのエクスポート 24
5 TI Tiny ML Tensorlab による高度な開発 26
5.1 環境設定 26
5.2 データ収集 27
5.3 モデル アーキテクチャの反復リファクタリング 29
5.4 モデルのエクスポート 33
5.5 量子化対応トレーニング 33
5.6 モデルのコンパイル 34
6 エッジ AI アプリケーション：数字認識 37
6.1 LeNet–5 バリアント CNN モデル 37
6.2 NPU/CPU の性能比較 38
7 エッジ AI アプリケーション：波形分類 39
7.1 特徴抽出 40
7.2 時系列分類モデル 42
7.3 モデル メモリに関する検討事項 43
7.4 NPU/CPU の性能比較 44

Copyright © 2026 Texas Instruments Incorporated

8 まとめ.....45
9 参考資料.....45
10 改訂履歴.....45

商標

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.
すべての商標は、それぞれの所有者に帰属します。

Copyright © 2026 Texas Instruments Incorporated

1 概要

エッジ人工知能 (AI) は、データが生成されるデバイス、またはデータ ソースの近くにあるデバイス上で推論モデルをローカルに実行する技術です。クラウドベースの AI と比較して、エッジ AI には、レイテンシ低減による応答性の向上、帯域幅要件の低減による運用効率の向上、クラウド依存性の低減、ネットワーク中断時の継続的な運用によるシステムの信頼性の向上、データ転送の最小化とプライバシー保護の強化など、大きな利点があります。

TinyEngine NPU を搭載した MSPM0G5187 を活用することにより、コンシューマ エレクトロニクス、スマートハウス デバイス、産業用システム、車載システムなど、さまざまなドメインにおけるエンド デバイスに AI 機能を直接統合することができます。専用のハードウェア NPU が並列動作するので、ハードウェア アクセラレーションが搭載されていない同等のマイコンに比べて、推論レイテンシを最大 90 分の 1 に短縮し、1 回当たりの推論に使用するエネルギー消費量を 120 分の 1 まで低減できます。

ただし、エッジ AI は単なるハードウェアではありません。ユーザーの開発と展開を真の意味で加速するためには、包括的なエコシステムが必要です。一般的な分類タスクから、回帰タスク、予測タスクに至るまで、TI の包括的なソフトウェア エコシステムを活用することで、エンジニアは最小限の複雑さで AI ソリューションの迅速なプロトタイピング、最適化、展開を実現できます。

- **TI Edge AI Studio**:モデル開発、トレーニング、コンパイル、展開のための直観的なグラフィカル環境を提供し
- **TI Tiny ML Tensorlab**:コマンド ライン ツールを介して、モデルのトレーニング、量子化、コンパイル、展開を網羅するエンド ツー エンドのワークフローを提供
- **TI ニューラル ネットワーク コンパイラ**:シンプルな構成により、NPU または CPU にシームレスに展開が可能

MSPM0 SDK には、すぐに使用できるエッジ AI の豊富なサンプル プロジェクト集が含まれており、開発の迅速化に貢献します。開発者は、リファレンス コードをマイコンに迅速に展開したり、再トレーニング用の事前ビルド済みモデルをインポートしたり、ユーザー開発したカスタム AI モデルをアプリケーションに統合したりできます。

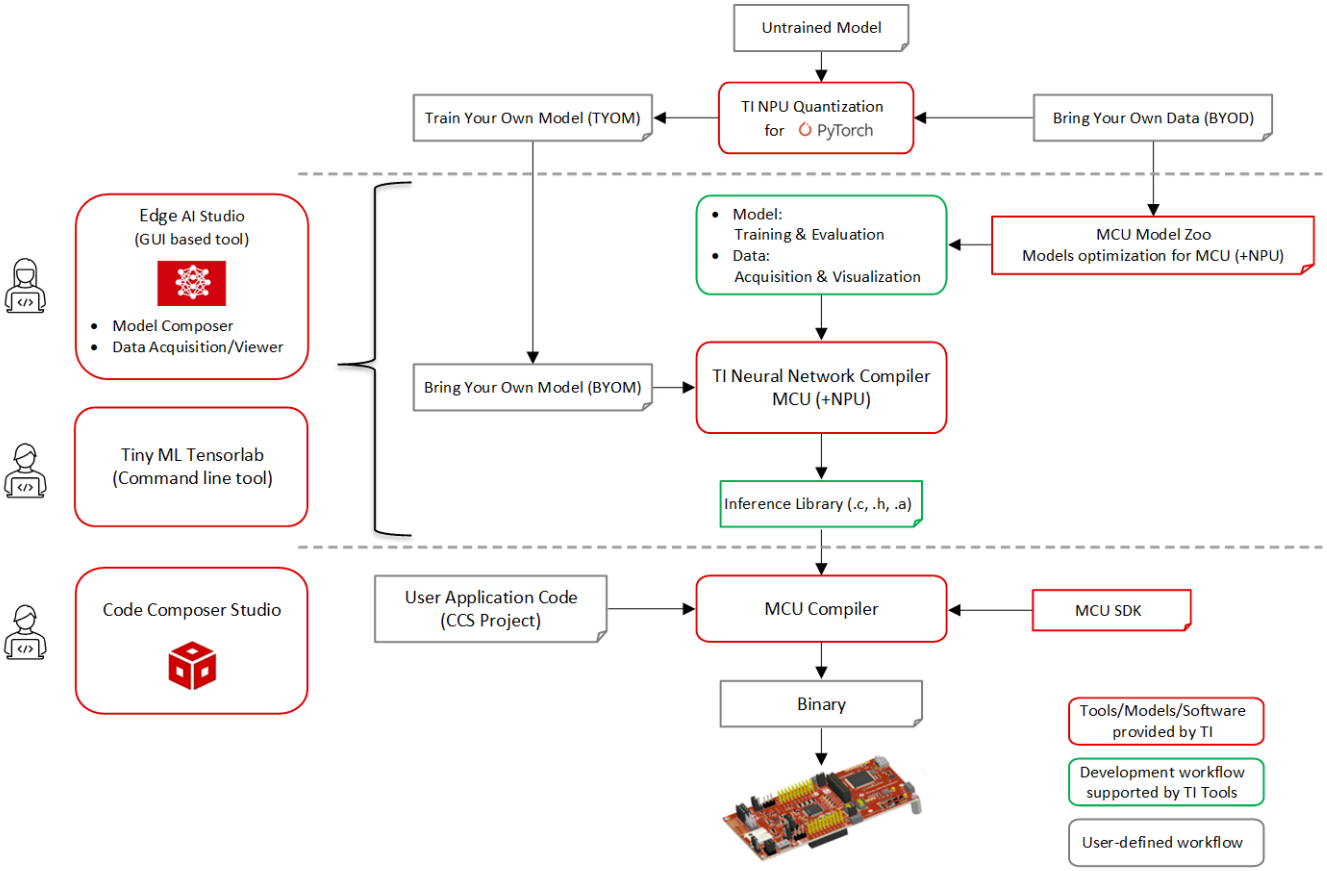


図 1-1. MSPM0G5187 へのエッジ AI モデルの導入

Copyright © 2026 Texas Instruments Incorporated

2 TinyEngine™ NPU を搭載した MSPM0G5187

MSPM0G5187 マイクロコントローラ (MCU) は、最大 80MHz の周波数で動作する、拡張された Arm® Cortex®-M0+ 32 ビット コア プラットフォームをベースとする、MSP 高集積超低消費電力 32 ビット MCU ファミリの製品です。これらのマイコンは、小型パッケージまたはピン数の多いパッケージ (最大 64 ピン) で最大 128KB のフラッシュ メモリを必要とするアプリケーション向けで、コストの最適化と柔軟な設計を両立できます。これらのデバイスには、USB2.0-FS インターフェイス、デジタル オーディオ インターフェイス、TinyEngine NPU が含まれており、動作温度範囲全体にわたって優れた低消費電力性能を実現します。

ハードウェア NPU は、深い畳み込みニューラル ネットワーク (CNN) 向けに高度に最適化されたコアで、事前トレーニング済みのモデルを使用した機械学習推論をサポートしています。オンチップ CPU との組み合わせで動作して、CNN 推論で高い性能を発揮して、消費電力を低減します。640 ～ 2560MOPS (1 秒あたりオペレーション数) の性能を備えた NPU は、ソフトウェア実装に比べて 90 分の 1 のレイテンシを実現し、スタンバイモード時の消費電流は 2µA 未満です。

図 2-1 に、チップ内のモジュールのトップレベル構成を示します。

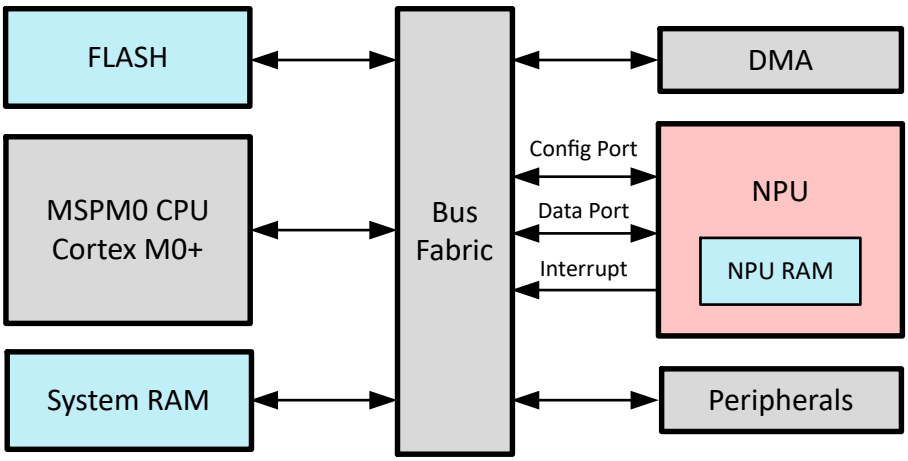


図 2-1. MSP CPU/NPU システム構造

NPU システムを搭載した MSPM0 は、不揮発性フラッシュ メモリと揮発性 RAM メモリを含むオンチップ メモリを共有しています。システム RAM は、CPU または NPU による読み取り / 書き込みをサポートしています。さらに、NPU には、CNN 推論専用 RAM が搭載されています。エッジ AI モデルは、利用可能なフラッシュ メモリの容量内に収まる必要があります。また、RAM の最大使用量も利用可能なシステム RAM の容量内に収まる必要があります。

Copyright © 2026 Texas Instruments Incorporated

3 エッジ AI ツールチェーン

エッジ AI の実装は、図 3-1 に示すように、オフライン トレーニング ワークフローと、オンデバイス ランタイム推論という 2 つの異なるフェーズに分かれています。

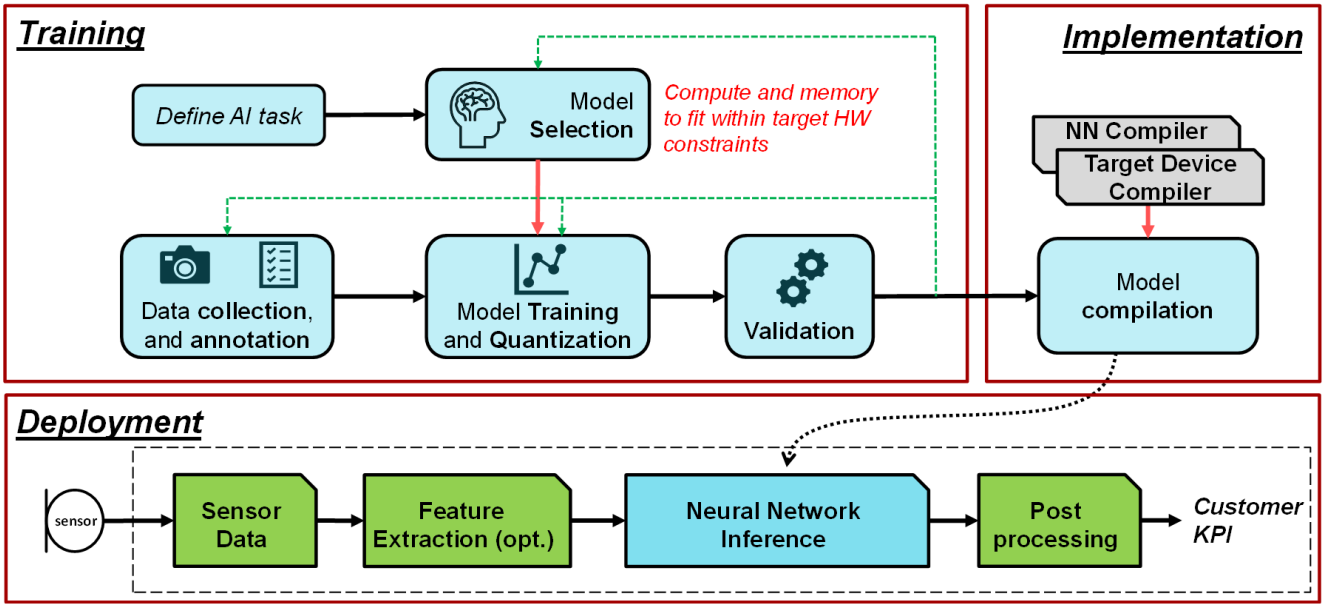


図 3-1. エッジ AI ワークフロー

モデル開発ワークフロー

開発パイプラインは、アプリケーションの定義から始まり、4 つのシーケンシャル・ステージを通じて最終的なハードウェア コンパイルに進みます。

- **タスク定義とモデル選択:** コア AI ワークロードは、特定のアプリケーション要件と運用上の制約に基づいて定義されます。開発者は、TI Model Zoo から構造的に最適化され、高度にパラメータ化されたリファレンス モデルを選択することも、独自のカスタム ネットワーク アーキテクチャをインポートすることもできます。
- **データセット管理:** 実際の環境で汎用性の高いモデルを実現するには、忠実度の高いデータセットが不可欠です。Edge AI Studio のグラフィカル ユーザーインターフェイス (GUI) は、データ アクイジションとアノテーションを効率的に行う環境を提供します。このプラットフォームでは、ユーザー独自のデータセットを取り込むこともできます。
- **トレーニングと定量化:** モデルは PyTorch® フレームワーク内でトレーニングされ、標準 ONNX 形式でエクスポートされます。ハードウェア効率を最大化するため、このパイプラインでは、TI が最適化したトレーニング後量子化 (PTQ) または量子化対応トレーニング (QAT/QDQ) の手法を活用しています。
- **コンパイル:** コンパイル済みモデルは、高度に最適化された C コードまたはバイナリ ライブラリに変換されます。このアーティファクトにより、ダウンストリームの Code Composer Studio (CCS) ファームウェア プロジェクトにシームレスなゼロ オーバーヘッド統合が可能になります。

オンデバイス ランタイム データ フロー

物理ハードウェアへの導入が成功すると、システムは確定的なパイプラインを介してリアルタイム処理およびエッジ推論を実行します。

- **信号収集:** 物理トランスデューサ (センサ) が、加速度、電圧過渡、音響波形などの環境変数を連続的にサンプリングし、それらをデジタル化されたセンサ データに変換します。
- **特徴抽出:** 利用可能な計算オーバーヘッドに応じて、未加工データに対してローカルで特徴抽出を実行することもできます。このステージはデータの次元を縮小し、ダウンストリームにある推論エンジンの計算負荷を低減します。
- **ニューラルネットワーク推論:** このコア実行フェーズは、高速な計算と分類を実行します。ネットワークのワークロードは、CPU 処理システム オーケストレーションと制御フロー管理により、遅延を最小限に抑えるハードウェア ニューラル プロセッシング ユニット (NPU) によって大幅に高速化されています。
- **後処理:** 未加工のテンソル出力がデコードされ、正規化され、高レベルの実用的なメトリックと顧客に対する主要業績評価指標 (KPI) に変換されます。

Copyright © 2026 Texas Instruments Incorporated

要約すると、このデカップリングされたアーキテクチャにより、物理データの認識、アルゴリズムによる推論、決定論的意思決定に及ぶループ全体をネットワーク エッジで実行できるため、クラウドへの依存がなく、確定的な応答時間が最適化されます。

3.1 TI Edge AI Studio

TI Edge AI Studio は、テキサス・インスツルメンツの幅広いプロセッサ、マイコン、レーダー センサを対象に AI モデルの開発、学習、コンパイル、展開を簡素化するための、グラフィカル ツールおよびコマンド ライン ツールからなる包括的なスイートです。

さまざまな開発環境とセキュリティ要件に対応できるように、Edge AI Studio には次の 2 つの異なるバージョンがあります。

- クラウド バージョン：ローカル インストールを必要としない完全な Web ベースの環境。このバージョンは、最小限の構成で、迅速なプロトタイピングと起動を実現します。
- デスクトップ バージョン：ユーザーのマシン上で実行されるローカル インストール。このバージョンは、データの機密性に対して厳格な要件があるプロジェクト向けに最適化されており、すべてのデータセットとトレーニング済みモデルがオンプレミスに存在し、TI からアクセスされることはありません。

Edge AI Studio は、以下のタスクに対応する堅牢なワークフローを提供します。

- **データ管理：**シームレスなデータキャプチャ、アノテーション、視覚化。
- **モデルのトレーニング：**
  - ユーザーが用意したデータ セットを用いて、TI が最適化したモデルのトレーニング (BYOD：Bring Your Own Data、独自データの導入)
  - トレーニング パラメータのカスタマイズによるモデル性能のファイン チューニング。
- **モデルのデプロイ：**
  - ターゲット ハードウェア向けのトレーニング済みモデルの自動コンパイル。
  - リアルタイムの性能検証を可能にするライブ推論プレビュー。

現在、一部の高度なカスタマイズ ワークフローは Edge AI Studio でサポートされていません。

- ユーザー定義モデルのトレーニング (TYOM：Train Your Own Model、独自モデルのトレーニング)
- TI モデルのアーキテクチャ変更
- 特徴抽出パイプラインまたは量子化フローのカスタマイズ
- 事前トレーニング済みまたはお客様が用意したモデルの直接コンパイル

注

高度なモデル カスタマイズ (TYOM やアーキテクチャ変更など) が必要なワークフローについては、セクション 3.2 を参照してください。これらの機能の拡張サポートについて記載されています。

Edge AI Studio は、C7™ NPU 搭載プロセッサ、AI アクセラレーション デバイス (TinyEngine 搭載マイコンなど)、専用アクセラレータのない AI サポート デバイスなど、複数の特化ファミリにわたる 22 個以上の TI マイコンをサポートしています。詳細については、EDGE-AI-STUDIO の製品ページを参照してください。

Copyright © 2026 Texas Instruments Incorporated

3.2 TI Tiny ML Tensorlab

Tiny ML Tensorlab は、リソースに制約のあるマイコン (MCU) に対して機械学習モデルの開発、トレーニング、最適化、デプロイを行うためのテキサス・インスツルメンツの包括的な AI ツールチェーンです。このツールチェーンを使用すると、以下のことを実行できます。

- 時系列タスクと画像分類タスクに関する機械学習モデルのトレーニング
- 組込みデバイスへの展開を目的とする、(2 ビット、4 ビット、8 ビット) の量子化を用いたモデルの最適化
- TI のマイコンでモデルを効率的に実行するためのコンパイル (オプションとして、NPU アクセラレーションも使用可能)
- Code Composer Studio (CCS) を使用したモデルの展開

このツールチェーンは、未加工データから、TI のマイコンで実行されるコンパイル済みモデルまでを網羅する、エンド ツー エンドのワークフローを提供します。ツールチェーンは次の処理を行います。

- **データの準備と特徴抽出:** 未加工信号のフォーマット、前処理、および高次元の特徴量への変換を自動化します。
- **モデルのトレーニング:** 組込みリソースの制約を考慮して特別に設計されたニューラル ネットワークのトレーニング。
- **モデルの最適化:** 量子化対応トレーニング (QAT) またはトレーニング後の量子化 (PTQ) を実装し、2 ビット、4 ビット、8 ビットの重み精度をサポートします。
- **コンパイル:** TI ニューラル ネットワーク コンパイラを使用して、高度に最適化されたデバイス固有のコードを生成します。
- **パフォーマンス分析:** ハードウェアの導入前にモデルの精度と推論指標を包括的に評価します。

TI Tiny ML Tensorlab は、次の機械学習 (ML) タスク タイプをサポートしています。

表 3-1. Tensorlab でサポートされているタスク タイプ

タスク タイプ	説明
時系列分類	時系列データを個別のクラスに分類 (故障検出、アクティビティ認識など)
時系列回帰	時系列入力から連続値を予測 (トルク推定など)
時系列予測	過去のパターンに基づいた将来値の予測 (温度予測など)
異常検知	オートエンコーダベースのモデルを使用した異常パターンの特定 (機器のモニタリングなど)
画像分類	画像をクラスに分類 (目視検査、数字認識など)

Tensorlab は、**C2000 DSP**、**MSPM0**、**MSPM33**、**AM13x/AM26x**、**コネクティビティ**など複数の特化型ファミリにわたり、TI のマイコンを幅広くサポートしています。レイテンシが重要で低消費電力のアプリケーションでは、ハードウェア アクセラレーション推論向けにテキサス・インスツルメンツの統合ニューラル プロセッシング ユニット (NPU) を搭載したデバイス (F28P55、AM13E2、MSPM0G5187 など) を選択してください。

注

新しいハードウェアが統合されると、サポートされるデバイスのリストが変更される場合があります。サポートされている型番の最新かつ包括的なリストについては、『Tiny ML Tensorlab ユーザー ガイド』の「サポートされるデバイス」を参照してください。

その他資料

- 『Tiny ML Tensorlab GitHub リポジトリ』
- 『Tiny ML Tensorlab ユーザー ガイド』

3.3 TI ニューラル ネットワーク コンパイラ

TI ニューラル ネットワーク コンパイラ (NNC) は、TVM コンパイラ フレームワーク上に構築された、マイコン デバイス向けの TI 製組込み AI コンパイル ツールチェーンです。TI NNC を使用すると、シンプルなコンパイル構成で AI モデルをハードウェア NPU アクセラレータまたはホスト CPU に柔軟に展開できます。アプリケーション コードやモデル側に変更を加える必要はありません。推論は常に統合型インターフェイスを経由して呼び出されるため、開発者は基盤となるハードウェア アーキテクチャに関する深い知識がなくても、組込みアプリケーションに AI 推論を統合できます。

Copyright © 2026 Texas Instruments Incorporated

表 3-2. NPU/CPU 展開構成の比較

対象の展開先	構成シンボル	初期化要件	推論の呼び出し
ホスト CPU	TVMGEN_DEFAULT_TI_NPU_SOFT	あり。推論の前に NPU を初期化	tvmgen_default_run()
ハードウェア NPU	TVMGEN_DEFAULT_TI_NPU	なし	

MSPM0 シリーズのモデルをコンパイルするには、以下のように環境を構成し、TVMC コンパイル コマンドを実行します。

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"
```

> バイナリ出力を必要としない場合 (たとえば、C コードのみを生成する場合)、ARM Clang コンパイラは不要です。コマンドでクロスコンパイラ引数を省略できます。

詳細については、『マイコン用 TI ニューラル ネットワーク コンパイラ ユーザー ガイド』を参照してください。

Copyright © 2026 Texas Instruments Incorporated

4 TI Edge AI Studio による迅速なプロトタイピング

Edge AI Studio は、プロトタイピングの迅速化と、TI が最適化した各種リファレンス モデルの迅速な検証を実現できるように設計されており、開発オーバーヘッドを大幅に低減できます。

このセクションでは、開発環境の概要を紹介します。表 4-1 に、このワークフローに必要なツール (それぞれのバージョンとアクセス リンクを含む) を示します。セットアップの詳細な手順を次に示します。

表 4-1. 開発ツールの要件

ツール	バージョン	機能	アクセス リンク
Edge AI Studio	1.7.0	ML モデル開発用 GUI ツール	EDGE-AI-STUDIO-MCU
MSPM0 SDK	2.11	TI MSPM0 マイコン向けソフトウェア開発キット (SDK)	MSPM0-SDK
ARM CLANG コンパイラ	5.1.1	TI MSPM0 マイコン向けのデバイス コンパイラ	ARM-CGT-CLANG
Code Composer Studio IDE	21.0	MSPM0 マイコン向け統合開発環境	CCStudio™ IDE ツール

注

本書で説明する手順は、Edge AI Studio のデスクトップ バージョンに基づいていますが、ワークフローはクラウド バージョンと同様です。

4.1 プロジェクトの設定

Edge AI Studio で新しい AI プロジェクトを開始するには、「*Workspace*」(ワークスペース) メニューに移動します。このプラットフォームには、2 つの主要なワークフローがあります。図 4-1 に示すように、標準ワークフローでは、「*Create New Project*」(新規プロジェクトの作成) オプションを選択します。

このプロセスでは、次の 4 つの特殊な AI タスク カテゴリのいずれかを選択して、アプリケーションのスコープを定義する必要があります。

- **分類:** カテゴリの識別とパターン認識を含むタスク。
- **回帰:** 連続する数値の予測。
- **異常検出:** データセット内の異常または不規則なパターンの識別。
- **予測:** 過去の傾向に基づく将来のデータ ポイントの予測。

プラットフォームは、選択されたタスクの特定の演算要件を満たすために基盤となるトレーニング方法論と最適化アルゴリズムを動的に構成します。このため、AI タスクの選択は重要です。

最適な結果を得るために、「*Sensor Selection*」(センサの選択) ページで汎用センサを選択し、その後、「*Home*」(ホーム) ページでそのタイプを構成することをお勧めします。アプリケーションで複数のソースからのデータが必要な場合は、「*Data Format Definition*」(データ形式の定義) ページで「*Multi-sensor*」(マルチセンサ) オプションを選択します。

プロジェクトを構成したら、プロジェクトの名前を変更して設定を完了し、Edge AI 開発ワークフローを開始します。

Copyright © 2026 Texas Instruments Incorporated

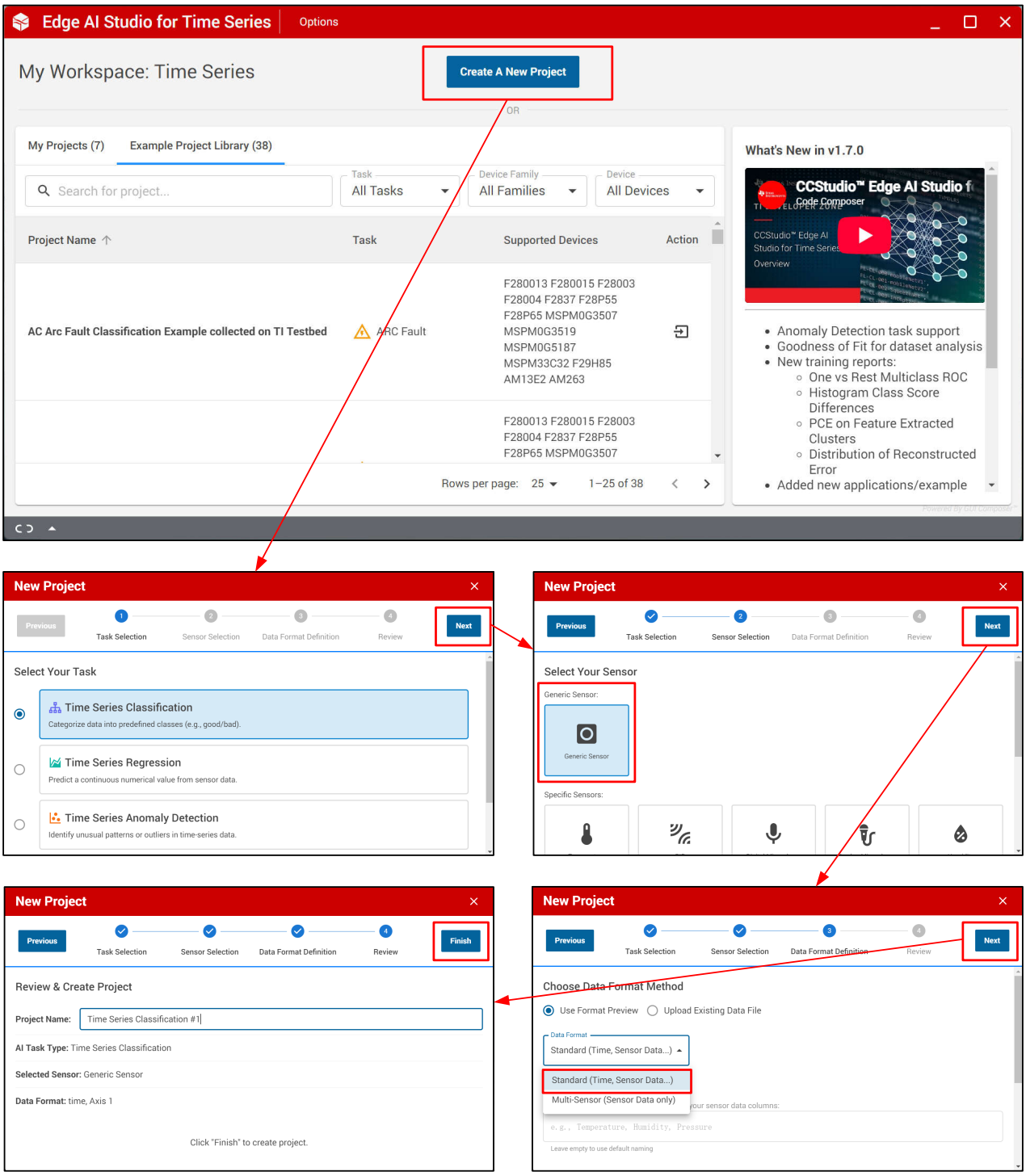


図 4-1. Edge AI Studio を使用した新しいプロジェクトの作成

また、事前構成済みのサンプル プロジェクトで構成された TI のライブラリを活用すると、開発サイクルを迅速化できます。現在、このプラットフォームは 38 の専門プロジェクトをサポートしており、ターゲット アプリケーション、特定のデータセット、または個別のモデルアーキテクチャ別に分類されます。

これらの例は、産業用とユーザー向けのさまざまな使用事例で基本的な参考資料として使用できます。たとえば、*アークフォルト検出*ドメインでは、いくつかの実装パスから選択できます。

Copyright © 2026 Texas Instruments Incorporated

- **TI のテスト データの実装:** TI 独自のデータセットを活用し、性能を最適化。
- **パブリック データセットの実装:** コミュニティの標準化されたデータセットを利用して一般的なベンチマークを実施。
- **異常検知の実装:** 特殊な*異常検知 AI タスク*を利用して、不規則な電気シグネチャを識別。

既存のサンプルを利用するには、*検索*機能を使用して、特定のキーワードを含むプロジェクトを検索します。見つかったら、「*Import*」(インポート) ボタンを選択して、プロジェクトをアクティブな*ワークスペース*に直接追加し、すぐに開発とテストを始めることができます。

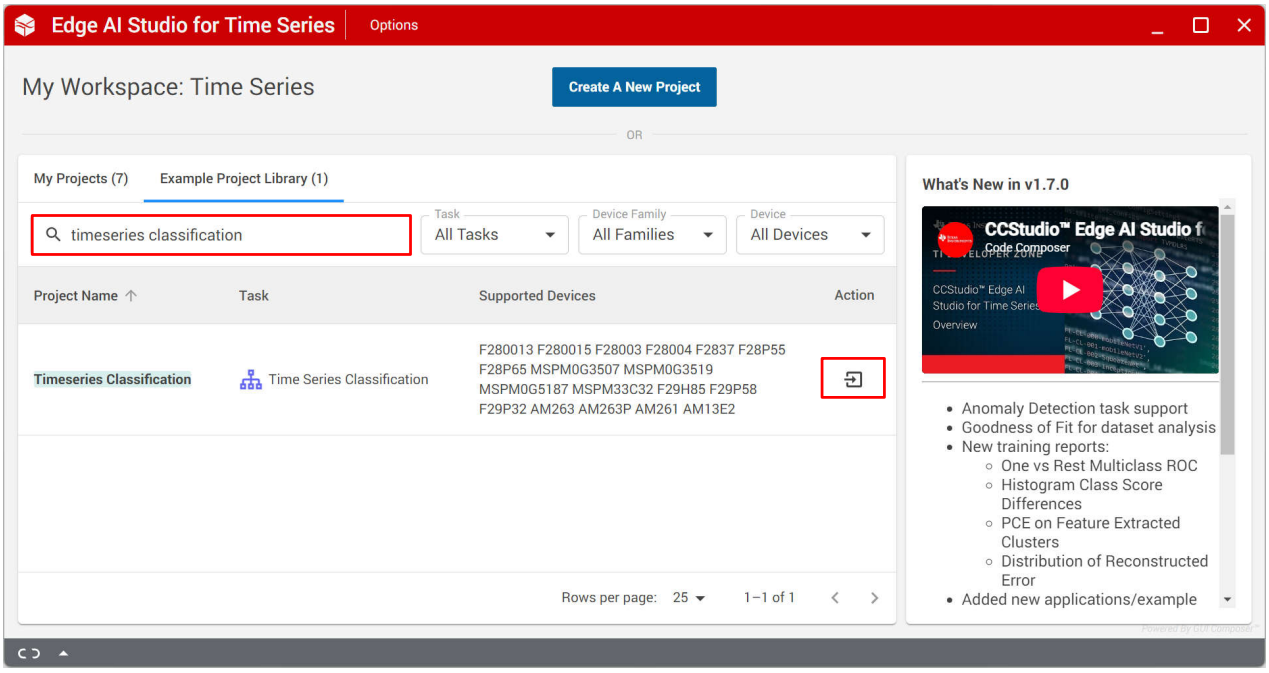


図 4-2. Edge AI Studio を使用したリファレンス プロジェクトのインポート

4.2 データ キャプチャ

このセクションでは、TI のマイコン (MCU) を使用してリアルタイム センサ データを取得し、Edge AI Studio で AI モデル開発用に構成する手順について説明します。

データ キャプチャでは、次の 2 つの基本的なワークフローがサポートされています。

- **リアルタイム データ キャプチャ:** このワークフローでは、互換性のあるセンサ評価基板 (TIDA-010997 など) と接続した TI のマイコン (LP-MSPM0G5187 など) を利用します。このシステムは標準化された GUI 通信プロトコルに従い、動作時にリアルタイム センサ データをキャプチャしてストリーミングします。
- **オフライン データのアップロード:** 他のハードウェア プラットフォームから取得したキャプチャ済みのデータセットをインポートできます。データのインポート プロセスの詳細については、セクション 4.3 を参照してください。

4.2.1 ハードウェア設定

シームレスなデータ ストリーミングを行い、データの破損を防ぐには、ターゲットのハードウェアおよびソフトウェア環境を次の手順に従って構成する必要があります。

1. **プロジェクトのインポート: Code Composer Studio (CCS)** を開き、次の SDK ディレクトリからプロジェクトをインポートします。

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

2. **ビルドとフラッシュ:** プロジェクトをコンパイルし、結果として得られたバイナリを LP-MSPM0G5187 ターゲット ボードにフラッシュします。

Copyright © 2026 Texas Instruments Incorporated

注

データのオーバーフローやパケット損失を防ぐため、ソフトウェア内のシリアル通信パラメータがファームウェア構成と一致させる必要があります。

接続を確立する前に、ファームウェアとホスト アプリケーションの両方でボーレートの設定を確認してください。

1. **ファームウェアの検証 (SysConfig)**:CCS プロジェクトの *SysConfig* ツールを開き、UART 構成モジュールにプログラムされているボーレートを確認します。
2. **Edge AI Studio の設定:**
   a. 「*Options*」(オプション) タブに移動します
   b. 「*Select Port*」(ポートの選択) を選択して、MCU に割り当てられているアクティブな COM ポートを確認します。
   c. *SysConfig* ツールで確認した値と一致するように、*ボーレート*を設定します

パラメータを同期したら、次の手順に従ってデータ アクイジションを開始します。

1. 「*Capture*」(キャプチャ) タブに移動します
2. 「*Connect To Device*」(デバイスに接続) を選択します。
3. 左下のステータス バーに「*Hardware Connected*」(ハードウェア接続済み)が表示されていることを確認します。

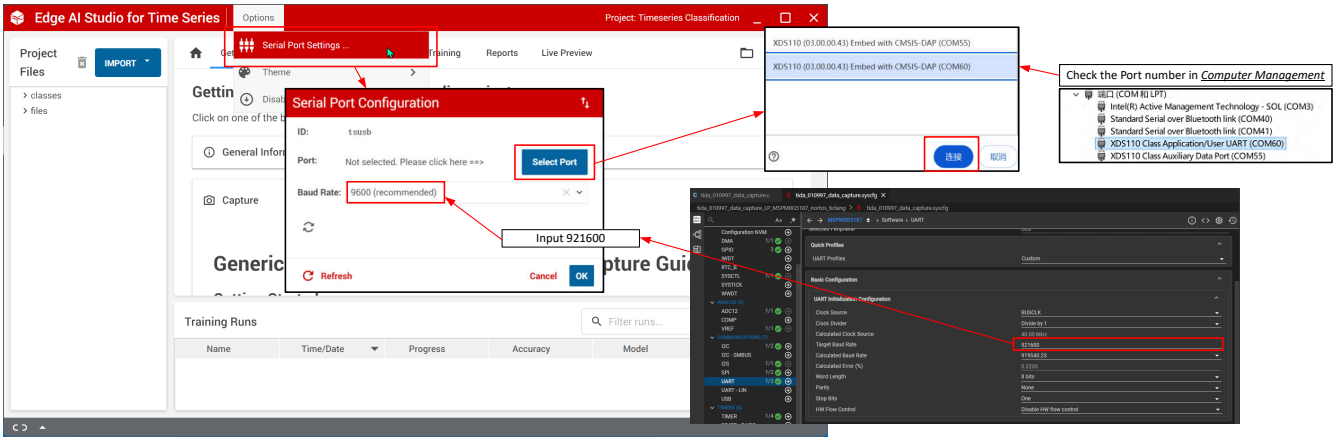


図 4-3. ハードウェアを Edge AI Studio に接続します。

4.2.2 機能の概要

Edge AI Studio の**データ キャプチャ** インターフェイスは、センサ ハードウェアの構成とデータ アクイジション パラメータの管理を行うための包括的なツールを提供します。このプラットフォームは、幅広いセンサ モジュールをサポートしています。たとえば、TIDA-010997 評価基板は、以下のような複数の高性能センサを搭載しています。

- モーション
  - デュアル素子 PIR センサとフレネル レンズ (IRA-S210ST01 + IML-0685)
  - 低消費電力 6 軸加速度計とジャイロスコープ (BMI270)
  - 高精度、リニア 3D ホール エフェクト センサ (TMAG5170)
- 環境:
  - デジタル湿度および温度センサ (HDC3020)
  - 高速、高精度周辺光センサ (OPT4001)
  - デジタル圧力センサ (BMI384)
- オーディオ:
  - デジタル I2S 出力、ボトム ポート マイク (ICS-43434)
  - 超低ノイズ、差動アナログ マイク (ICS-40740)

データ キャプチャ プロセス中に、次のパラメータを構成して、体系的で再現可能なデータ収集を行うことができます。

- **データ ラベル:**データセットにカスタム識別子を割り当てます。これは、データを効率的に管理するために、ファイルの自動命名規則で使用されます。

Copyright © 2026 Texas Instruments Incorporated

- **サンプル数:** 1 つのキャプチャ セッションで収集するサンプル数を定義します。
- **キャプチャ間隔:** 連続するキャプチャ サイクルの数と、各サイクル間の遅延時間を設定します。これにより、ハードウェアの正確な同期とテストが可能になります。

アクイジションが完了したら、「*Project Files*」(プロジェクト ファイル) →「*files*」(ファイル) の順に選択して、キャプチャした時系列データを確認できます。

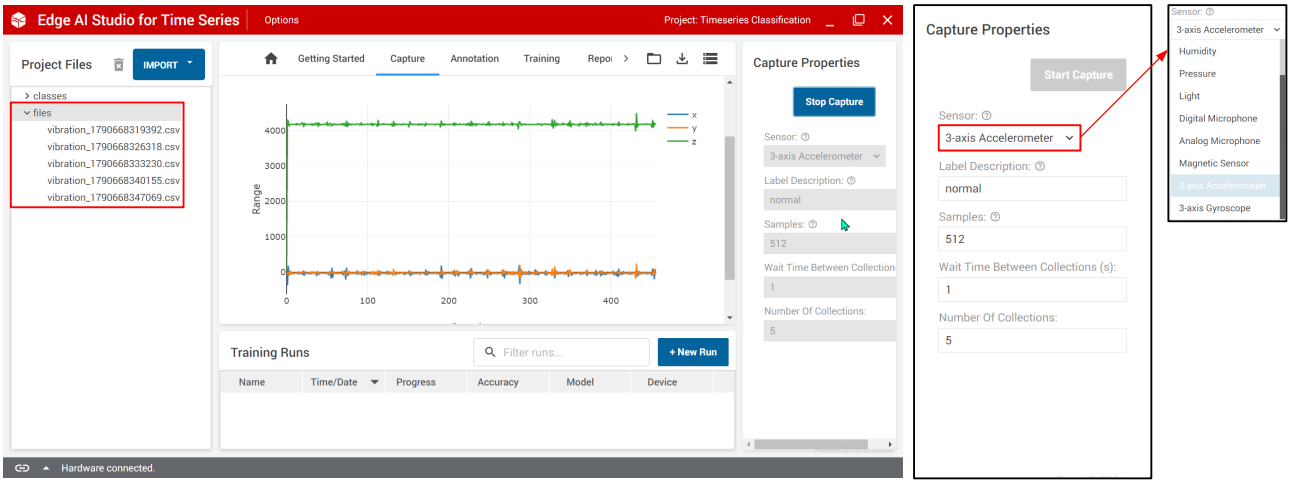


図 4-4. キャプチャ ページの概要

4.3 データのインポート

モデルを直接トレーニングする場合、図 4-5 に示すように、キャプチャ済みのデータセットをローカル PC からインポートできます。

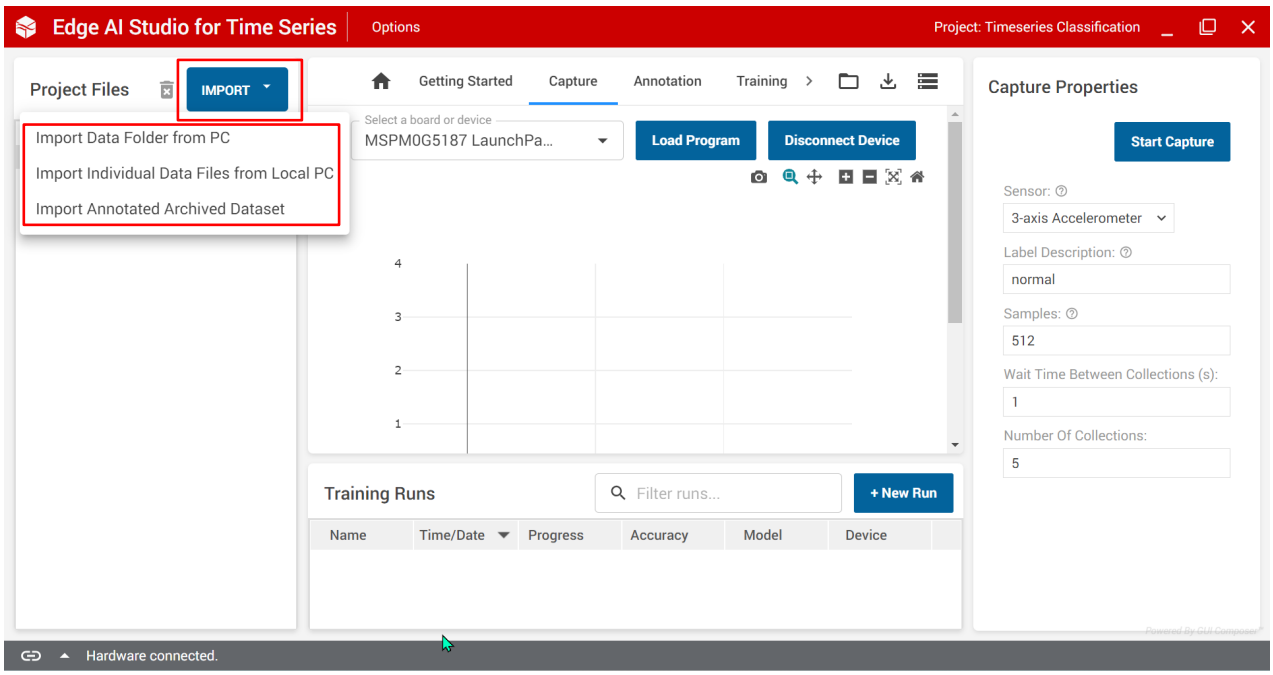


図 4-5. ローカル PC からのデータのインポート

Copyright © 2026 Texas Instruments Incorporated

データの取り込みを成功させるには、アップロードするファイルが、各タスクで必要な形式になっている必要があります。データの準備に関する包括的なガイダンスについては、『TI Tiny ML Tensorlab ユーザー ガイド』の「BYOD：独自データの導入」を参照してください。

次のセクションでは、各種の AI タスクでサポートされているデータ形式について詳しく説明します。

4.3.1 分類タスク

データ フォーマット

A. ヘッダーなしの形式 (シンプル)

この形式は、単一変数のデータセットで実用的です。ヘッダーなしの形式は、未加工の数値から構成され、各行に 1 つの測定値が含まれています。

例：

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

B. ヘッダー付きの形式 (複数の変数がある場合に推奨)

マルチチャネルのセンサ データにはこの形式を推奨します。先頭行に列名が含まれている必要があります。

例：

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

C. 時間列の処理

Edge AI Studio は時系列のタイムスタンプを自動的に検出して除外し、特徴量として扱われないようにします。**time** という文字列 (大文字小文字の区別なし) を含む列は、インポート時に自動的に除外されます。

例：

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

4.3.2 回帰タスク

データ フォーマット

回帰タスクは、複数の入力特徴量に基づいて連続数値を予測します。回帰タスクの場合、*目標値 (ラベル) がデータセットの最終列に存在する必要があります*。ファイルは、先頭行に特徴名と目標変数が定義されているヘッダー付きの形式にする必要があります。ファイル形式は次のようになります。

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

例：モーター トルク予測

- 列 1 ～ 6：入力特徴量

Copyright © 2026 Texas Instruments Incorporated

- 列 7 (最後): 予測目標値

```
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
```

タイムスタンプが入力特徴量として誤って処理されないように、Edge AI Studio は Time を含む列を自動的に識別して削除します。

ラベリング ロジック

注

回帰タスクには、*単純なウィンドウ*変換が必要です。

Edge AI Studio は、ウィンドウ付きの平均化プロセスを目標値に適用します。このプロセスにより、時系列モデリングのターゲット ラベルが安定化されます。

- **ウィンドウ抽出:** サイズ N のウィンドウ (*frame_size* パラメータで定義) が未加工の目標列から抽出されます。
- **平均計算:** そのウィンドウ内の目標値の平均が計算されます。
- **ラベルの割り当て:** この平均値がウィンドウ全体の単一のターゲット ラベルになります

計算例 (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 予測タスク

予測タスクでは、過去の観測結果に基づいて、指定された 1 つまたは複数の変数の将来値を予測します。

予測では、すべての変数は列に編成されます。ターゲットが専用の列である回帰とは異なり、予測はデータセット内の既存の変数を利用します。入力特徴量として機能する列と、構成パラメータで予測される特徴として機能する列を明示的に指定する必要があります。

例: 温度予測

```
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
...
```

この例では、データセットは、最初の 3 つの列が入力特徴量として機能し、最後の列が予測の目的変数を表すように構造化されています。列の対応は次のとおりです。

- 列 0 (**ambient**): 入力特徴量
- 列 1 (**coolant**): 入力特徴量
- 列 2 (**current**): 入力特徴量
- 列 3 (**pm_temp**): 予測目標

回帰タスクと予測タスクのどちらも連続的な数値目標を使用していますが、表 4-2 に示すように、時間論理と目標の定義は根本的に異なります。

表 4-2. 回帰タスクと予測タスクの比較

特長	回帰タスク	予測タスク
主な目的	現在または最近のウィンドウ特徴量に基づいて連続値を予測します。	過去の時間的パターンと傾向を特定することで、将来の値を予測します。
目標の定義	ウィンドウ平均: 目標は、現在の観察ウィンドウ内の平均値として定義されます。	将来値: 目標は、事前に定義された将来のタイムステップにおける特定の値です。
データ要件	標準のウィンドウ変換が必要です。	予測地平線を明示的に設定する必要があります。

Copyright © 2026 Texas Instruments Incorporated

4.3.4 アノテーション リスト

正確なモデル トレーニングと評価を容易にするため、Edge AI Studio では、専用のアノテーション ファイルを使用してデータセットを明示的に分割することができます。トレーニング、検証、テスト用のサブセットを定義することで、堅牢なモデル性能を実現できます。

> より小規模なデータセットや迅速なプロトタイピングでは、Edge AI Studio のグラフィカル ユーザー インターフェイス (GUI) で直接ラベルとパーティションを手動で割り当てることができます。

プロジェクト ディレクトリの構造

必要なディレクトリ構成は、AI タスクの種類 (分類と回帰 / 予測) によって異なります。以下の構造を参照してください。

分類タスクの場合: データセットは、各ターゲット クラスを表すサブディレクトリに編成する必要があります。

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

回帰と予測タスクの場合: データは、フラットなディレクトリ構造内の個々のファイルとして保存されます。

```
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
```

アノテーション ファイルの形式

アノテーション ファイル (.txt) には相対パスのリストが含まれており、各パスが一意の行に記述されている必要があります。

例:

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

4.4 データ アノテーション

データ アクイジションまたはインポートが成功すると、ワークフローはデータ アノテーションとデータセットの分割に進みます。これらのプロセスは、Edge AI Studio GUI を使用して実行し、モデルのトレーニングと評価に使用する未加工データを準備します。

Copyright © 2026 Texas Instruments Incorporated

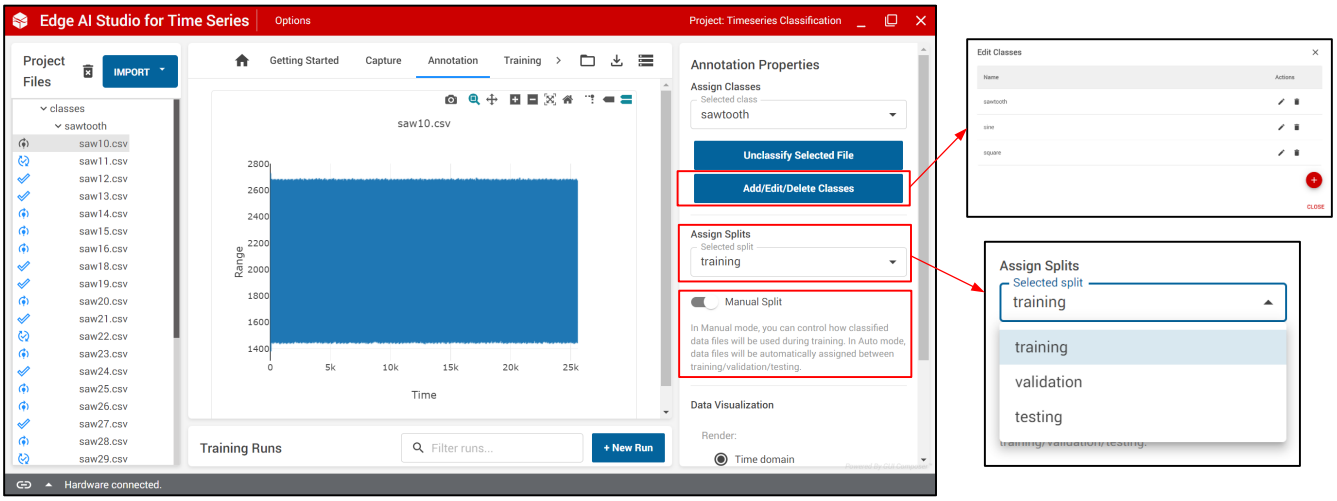


図 4-6. アノテーション ページの概要

アノテーション ワークフロー

「**Data Annotation**」(データ アノテーション) インターフェイスでは、次の操作を実行して、未加工データを意味のあるクラスに編成します。

- **クラス管理:** ターゲット アプリケーションの要件に合わせて新しいクラス カテゴリを作成したり、既存のクラスの名前を変更したりできます。
- **データの割り当て:** 個々のファイルを選択して特定のクラスに割り当てることも、既存のカテゴリ間で移動してデータセットを絞り込むこともできます。

データセットの分割

クラスが割り当てられたら、データセットを 3 つの異なるサブセット (トレーニング、検証、テスト) に分割する必要があります。表 4-3 に、各サブセットの要件と目的をまとめます。

表 4-3. データセット要件

サブセット	要件	目的
トレーニング データセット	必須	モデルで特徴を学習し、重みを最適化するために使用されます。
検証データセット	必須	トレーニング中にハイパーパラメータを調整し、オーバーフィットを監視するために使用します。
テスト データセット	オプション	未知のデータで最終モデルの性能を評価するために使用します。

Edge AI Studio には**手動分割**モードがあり、サンプルの分布を詳細に制御できます。このモードは、特定のエッジケースをテスト用に予約する必要がある特殊なアプリケーションに推奨されます。

モデルの評価に統計的な有意性を与えるため、テスト データセットの分配は慎重に検討する必要があります。

注

最も信頼性の高いパフォーマンス メトリックを取得するには、テストセットがデータセット全体のかなりの部分を表す必要があります。ほとんどのアプリケーションでは、合計データの 50% を超えるテスト セットを推奨します。評価結果が統計的に堅牢で、実際の動作条件を表すようにするため、データセット全体のサイズが大きくなるにつれて、テストセットの割合も増やしていく必要があります。

4.5 モデルのトレーニング

Edge AI Studio のモデル トレーニング インターフェイスは、モデル アーキテクチャの構成とハイパーパラメータのトレーニングに適した効率的なワークフローを提供します。このセクションでは、設定手順とその基礎となるコンパイル・プロセスの概要について説明します。

Copyright © 2026 Texas Instruments Incorporated

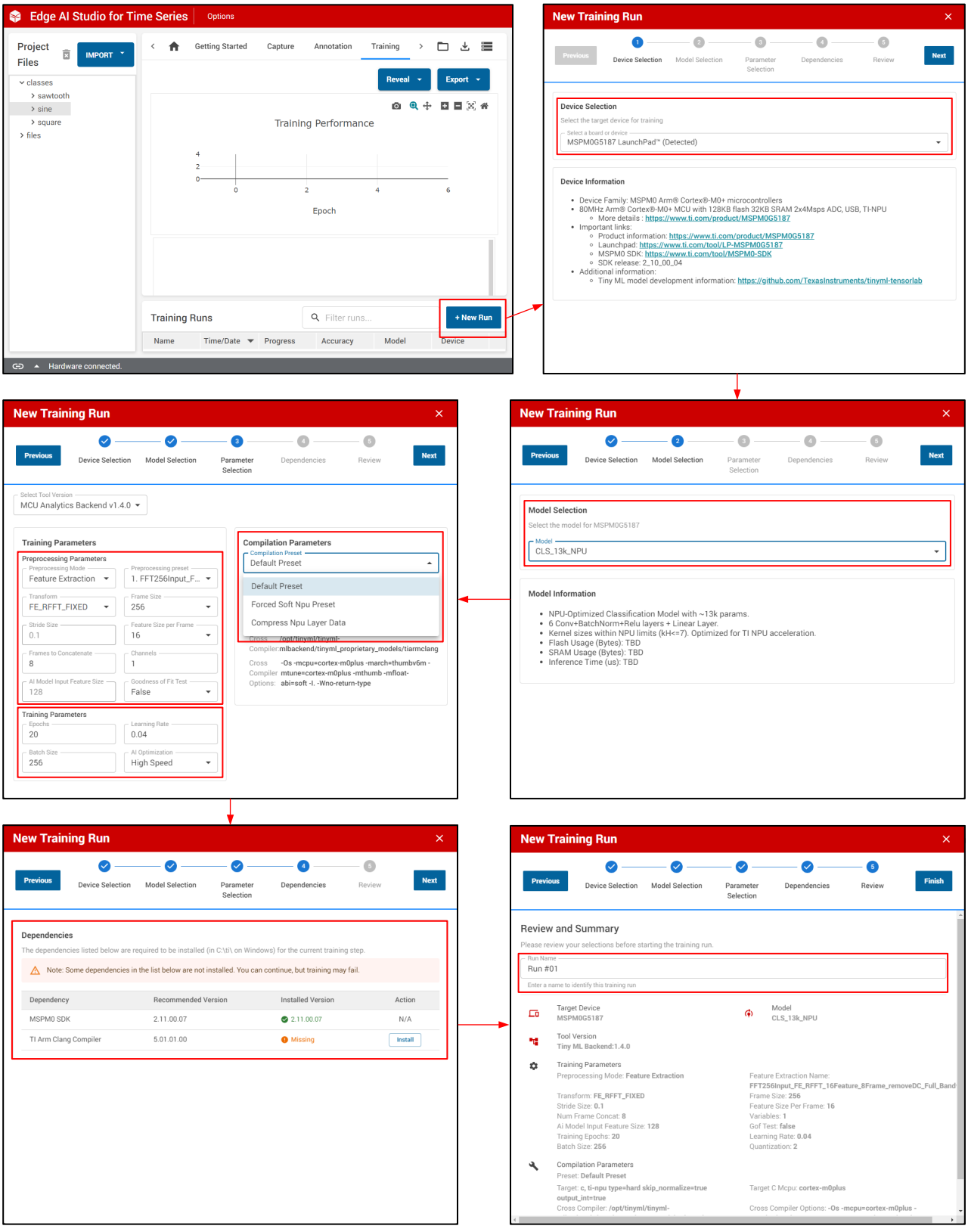


図 4-7. トレーニング ページの概要

Copyright © 2026 Texas Instruments Incorporated

モデルおよびハードウェア構成

トレーニング プロセスを開始するには、ターゲット ハードウェアとモデルアーキテクチャを定義する必要があります。

1. ターゲット デバイスの選択：提供されたドロップダウン メニューからターゲット マイコン (MCU) を選択する必要があります。Edge AI Studio は、20 種類以上のマイコン シリーズから構成された多様なエコシステムをサポートしています。このクロスプラットフォーム機能により、単一の AI モデル アーキテクチャを、特定の電力および性能要件に基づいて、さまざまなハードウェア プラットフォームに移植および導入できます。
2. **AI モデルの選択：**分類タスク向けに、Edge AI Studio は TI が最適化した 24 種類以上のモデル アーキテクチャを提供しています。モデルを選択する際には、計算の複雑さと予測精度のトレードオフを考慮する必要があります。
   a. **モデルの複雑さ：** より多くのレイヤーとパラメータを持つ大きなモデルは、通常より高い精度をもたらします。
   b. **リソースの制約：**パラメータ数が増加すると、メモリの消費量 (SRAM / フラッシュ) と推論の遅延時間が増加します。

モデルを選択し、トレーニング パラメータを定義した後 (パラメータの詳細については、セクション 4.5.1 を参照)、実行ターゲットを指定する必要があります。

- **デフォルト プリセット (NPU)：**ニューラル プロセッシング ユニット (NPU) を使用して、モデルをハードウェア アクセラレーションに最適化します。
- **ソフト NPU 強制プリセット (CPU)：**推論ワークロードを CPU にルーティングします。通常、デバッグに使用されるか、専用の NPU ハードウェアがないデバイスに使用されます。

トレーニングとコンパイル パイプライン

トレーニング済みモデルからターゲットに対応したバイナリへの移行は、厳密なマルチステージ パイプラインに従います。このプロセスは、トレーニング フェーズが完了し、モデル アーティファクトが生成された後に開始されます。

1. **トレーニング フェーズ：**提供されたデータセットに基づいてモデル パラメータを最適化します。正常に完了すると、Edge AI Studio は高レベルのモデル ファイルを ONNX 形式で生成します。この ONNX ファイルは、以降の変換フェーズで標準化された中間表現として機能します
2. **ONNX からソースコードへの変換：**生成された ONNX モデルが **TI NNC** によって処理されます。NNC は、グラフの最適化と演算子マッピングを実行して、高レベルの ONNX 表現を最適化されたソースコードに変換します
3. **ソース コードからバイナリへの変換：**最終段階で、最適化されたソース コードが**デバイス コンパイラ** (たとえば、ARM Clang) に渡されます。このコンパイラは、低レベルのマシンコードを生成し、特定のターゲット マイコン アーキテクチャに合わせてカスタマイズされた最終的なバイナリ ライブラリ (.a ファイル) を生成します

注

デバイス コンパイラが PC システムにインストールされていない場合、コンパイル パイプラインは致命的なエラーで終了します。トレーニングはコンパイル ツールチェーンから独立しているため、前のモデル トレーニング フェーズは正常に完了します。

4.5.1 トレーニング パラメータ

トレーニング パラメータの設定は、前処理パラメータとトレーニング ハイパーパラメータの 2 つの段階に分かれます。

前処理パラメータ

Edge AI Studio は、モデル トレーニング用のデータを準備する 2 つの前処理ワークフローをサポートしています。

- **シンプル ウィンドウ：**スライド式のウィンドウ メカニズムを使用して、データセットの未加工の時系列データを利用します。ウィンドウ サイズは 0.01 ～ 1.0 の範囲で設定できます
- **特徴抽出：**このワークフローは高度なデータ変換を実行します。特徴抽出モジュールでは、次の 2 つの処理モードから選択できます。
  - **FE_RFFT_FIXED：**FFT を使用して周波数ドメイン情報を抽出します。サポートされている FFT サイズは 2 のべき乗である必要があり、出力サイズは「フレーム サイズ / 2」と定義されています。その後、FFT 出力は、「フレームあたりの特徴量サイズ」パラメータで定義されている特定の数のビンに集約されます
  - **RAW：**周波数変換なしで、未加工の時系列データを利用します

Copyright © 2026 Texas Instruments Incorporated

特徴抽出ワークフローを設定するには、次のパラメータを使用します。これらの設定は、**FFT** と **未加工** の両方の処理モードに適用できます。

表 4-4. 特徴抽出パラメータ

パラメータ	説明
フレーム サイズ	処理フレームあたりの入力サンプル数を指定します。
フレームあたりの特徴サイズ	フレームごとに生成されるビン (または特徴) の数を定義します。
連結するフレーム	バッファから取得する前のフレームのビン数を指定します。このメカニズムにより、モデルは周波数スペクトル全体の時間変化を捕捉することができます。
チャネル	同時に処理されるセンサ チャネル数を定義します。

包括的な実装例については、セクション 7.1 (波形分類アプリケーション) を参照してください。

トレーニング ハイパーパラメータ

トレーニング ハイパーパラメータは、最適化プロセスと最終的なモデルの特性を制御します。

- **エポック:**トレーニング データセットによる完了パスの合計数を定義します。一般に、エポック数を増やすと、モデル パラメータが最適な設計に近づくことができますが、エポックが多すぎるとオーバーフィッティングにつながる可能性があります。
- **学習率:**各エポックの最急降下法に従ってパラメータを更新する際のステップサイズを決定します。学習率が高いほど、グローバル最適化への収束を加速させることができますが、振動や不安定性を引き起こす可能性があります。逆に、学習率が低いほど収束はより安定しますが、より多くのエポックが必要になり、局所解になる可能性があります。
- **バッチ サイズ:**1 回の反復で処理されるトレーニング サンプルの数を指定します。バッチ サイズを大きくすると、ホスト PC の計算負荷とメモリ圧力が大きくなることに注意してください。
- **最適化 (精度モード):**トレーニング済みモデルのターゲット精度を定義します。
  - **高精度:**フル精度の浮動小数点モデルをトレーニングします。
  - **高速:**量子化対応トレーニング (QAT) を利用して、高速推論用に最適化された量子化固定小数点モデルを作成します。

4.5.2 ツールチェーンの依存関係

必要なデバイス コンパイラ (ARM Clang) と MSPM0 SDK がホスト システムにすでにインストールされていても、Edge AI Studio のモデル コンパイル プロセスが失敗する場合があります。この問題は主に、Edge AI Studio の以前のバージョンで確認されています。

一部のバージョンでは、Edge AI Studio が必要なツールのインストール パスを自動的に検出できず、バックエンドがコンパイル段階でコンパイラを正しく起動しないことがあります。

ツール検出の問題でコンパイルが失敗した場合は、次の手順でツールのインストール パスを手動で指定できます。

1. 次のディレクトリに移動します ([version] は、特定のインストール バージョンに置き換えます)。

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

2. ファイルを開きます。

```
constants.py
```

3. 次のコード ブロックを探して、TOOLS_PATH 変数を変更します。

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

4. 通常は以下のように、TI のインストール ディレクトリを指定するように、このディレクトリを直接指定します。

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


Copyright © 2026 Texas Instruments Incorporated

5. コンパイラのバージョンを確認し、MSPM0_CGT_VERSION がインストール済みのバージョンと一致していることを確認します。例:

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

4.6 パフォーマンス レポート

トレーニング プロセスが完了すると、「*Reports*」(レポート) インターフェイスにアクセスして、モデルの予測パフォーマンスを評価できます。「*Reports*」(レポート) インターフェイスには、混同行列、多クラス ROC 曲線、ペアワイズ クラス予測差という 3 つの異なる分析次元が用意されており、図 4-8 に示すように、精度、堅牢性、決定マージンを包括的に評価することができます。

Copyright © 2026 Texas Instruments Incorporated

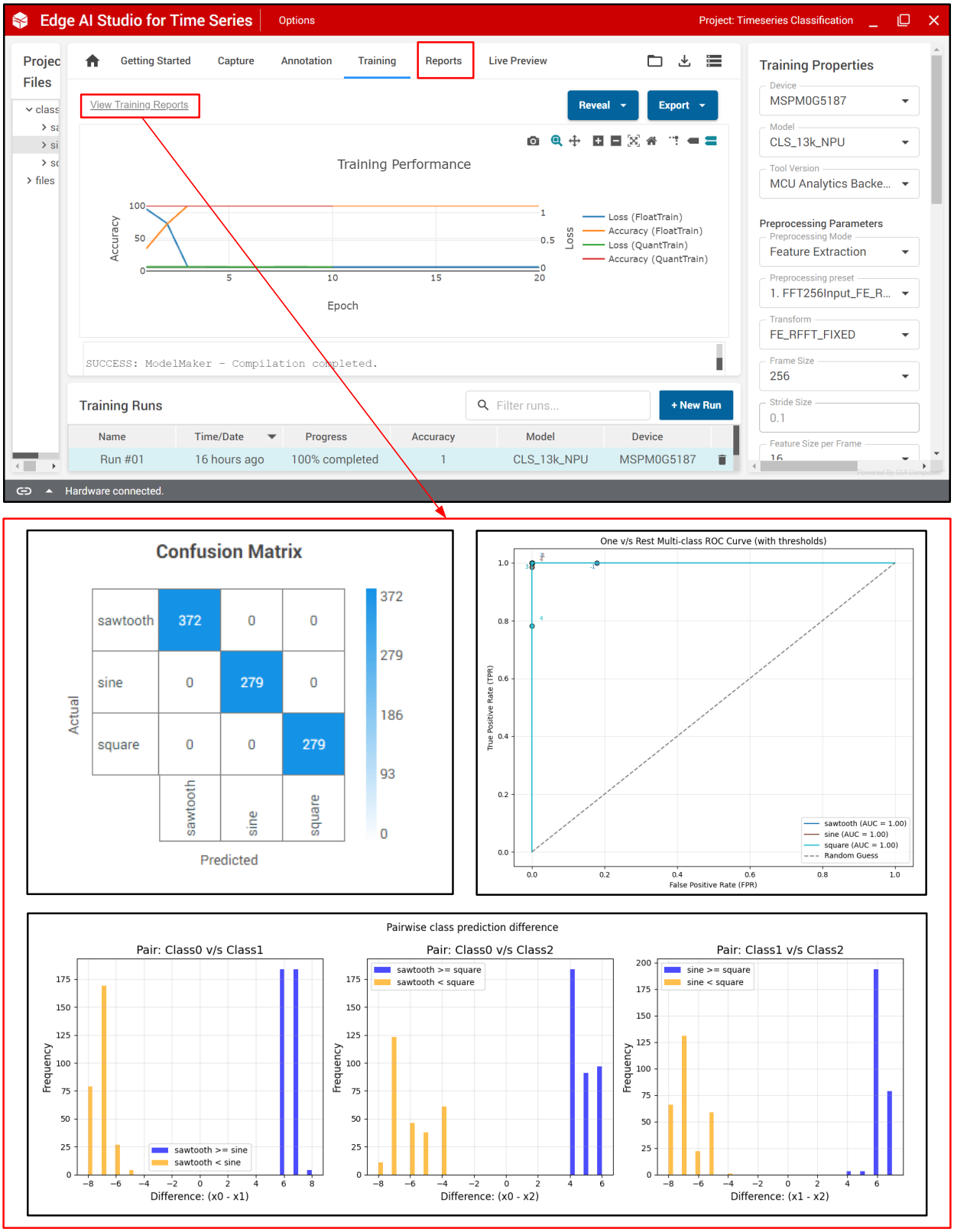


図 4-8. レポート ページの概要

Copyright © 2026 Texas Instruments Incorporated

混同行列

混乱行列は、教師あり学習アルゴリズムのパフォーマンスを特定の決定スレッショルドで可視化するための基本的なツールです。混乱行列は、モデルが実際のデータセット全体に予測をどのように分散するかを詳細に示します。

行列は、次のようなグリッドに編成されます。

- **行 (Y 軸):**テスト データセットの**実際/真のクラス** ラベルを表します。
- **列 (X 軸):** モデルによって生成された**予測クラス** ラベルを表します。
- **対角成分:正しい予測**を表します (予測されたラベルは実際のラベルと一致します)。
- **非対角成分:分類の誤り** (エラー) を表します。

混同行列は、モデルの信頼性を評価するための多面的な診断ツールとして機能します。主に、対角線上に値が集中していることは、モデル全体の精度と、個々のクラスにおける適合率や再現率を視覚的に表しています。対角線が濃いほど、モデルが各クラスの潜在的なパターンを正確に把握していることを示しています。

全体的な精度以外にも、非対角成分を分析することで、誤りのパターンの高度な診断を行うことができます。誤分類が集約される場所を観察することで、特徴の混同が起きている箇所を特定できます。たとえば、クラス A が頻繁にクラス B と誤認されている場合、現在の特徴セットには、この 2 つの特定のカテゴリを分離するために必要な識別能力がないことを示しています。このようなインサイトは、アップストリームの特徴エンジニアリングまたはデータ前処理を最適化するための具体的な方向性を示しています。

さらに、この行列は、データセットの整合性に対する重要な検証に役立ちます。個々の行または列にわたる要素の合計を計算することにより、テストセット内のサンプルの分布を検証できます。このステップは、パフォーマンス指標がクラスの不均衡によって歪んでいないことを確認するために不可欠であり、報告された精度がモデルの真の一般化能力を客観的に反映していることを保証できます。

多クラス ROC 曲線 (一対他)

受信者操作特性 (ROC) 曲線は、偽陽性率 (FPR) に対して真陽性率 (TPR) をプロットすることで、モデルのクラス間の識別能力を評価します。Edge AI Studio では、多クラス タスクに一対他 (OvR) 戦略を採用しています。このアプローチでは、個々のクラスが「正」クラスとして扱われ、他のクラスはすべて単一の「負」クラスに集約されます。これにより、モデル内のすべてのクラスに対して個別の ROC 曲線を生成できます。

多クラス ROC 曲線は、さまざまな動作条件におけるモデルの識別能力を包括的に表しています。この評価の主な指標は曲線下面積 (AUC) です。AUC 値が 1.0 に近づくと、対応する曲線が左上に著しく引き寄せられ、ターゲット クラスとデータセットの残りの部分がほぼ完全に分類されていることを表します。逆に、AUC が 0.5 に近い場合、モデルは予測的有用性を欠いており、ランダムな予測と変わらないことを示しています。

ROC 曲線は、単一のスカラー値を超えて、スレッショルドの堅牢性評価に欠かせないツールです。この曲線は、さまざまな分類信頼スレッショルドにわたって性能を追跡するため、曲線の安定性、つまり、広範囲で偽陽性率 (FPR) を最小限に抑えながら高い真陽性率 (TPR) を維持することが極めて重要となります。明示的なスレッショルド アノテーションを曲線に直接統合することで、特定のスコアが真陽性率および偽陽性率にどのように対応しているのかを視覚的に特定できます。

リアルタイム組込みシステムにモデルを導入する場合、ROC 曲線が安定していることは、感度や誤警報許容値などの特定のアプリケーション要件を満たすように分類スレッショルドを動的に調整しても、モデルの信頼性が維持されることを意味しています。

ペアワイズ クラス予測差

ペアワイズ クラス予測差のヒストグラムは、決定マージン、つまり、モデルが任意の 2 つのクラスを識別する際の信頼性を分析します。このヒストグラムは、2 クラス間の未加工の決定スコア (または予測される確率) の差の分布をプロットしたものです。ヒストグラムは、最終的なハードラベルではなく、モデルの出力分布の統計的挙動を捕捉します。

従来の指標は正誤の判定に焦点を当てていますが、ペアワイズ クラス予測差のヒストグラムでは、モデルの決定マージンと確率的な確実性について、より詳細な分析が可能になります。2 クラス間のスコア差の分布を調べることにより、モデルの分類境界を定義する統計的なギャップを定量化できます。

Copyright © 2026 Texas Instruments Incorporated

高性能モデルは、2 つのピークが大きく離れている明確なバイモーダル分布を示すのが特徴です。このような構造は、モデルが広い決定マージンを持ち、その出力に高い信頼性を提供することを示しています。最も重要なことは、ゼロ点付近 **(曖昧領域)** の分布の密度がノイズ耐性の重要な指標として機能する点です。

ゼロ点付近の密度が最小または「真空」であるヒストグラムは、このモデルが小さな信号の摂動に対して非常に高い耐性を備えていることを示しています。このモデルは、過渡ノイズや環境の変動による誤った状態遷移を引き起こす可能性が統計的に低いため、実際の導入において優れた干渉耐性があることを意味します。

4.7 モデルのエクスポート

トレーニング フェーズが完了したら、トレーニング済みモデルをエクスポートして、カスタマイズされた開発ワークフローに統合できます。Edge AI Studio では、エッジ側でのコンパイル済みモデルのエクスポートもサポートされています。バージョン 1.7.1 でサポートされているエクスポート形式は、エッジ デバイス用に最適化されたバイナリ ファイルに限定されています。

図 4-9 に、Edge AI Studio のモデル エクスポート ワークフローを示します。

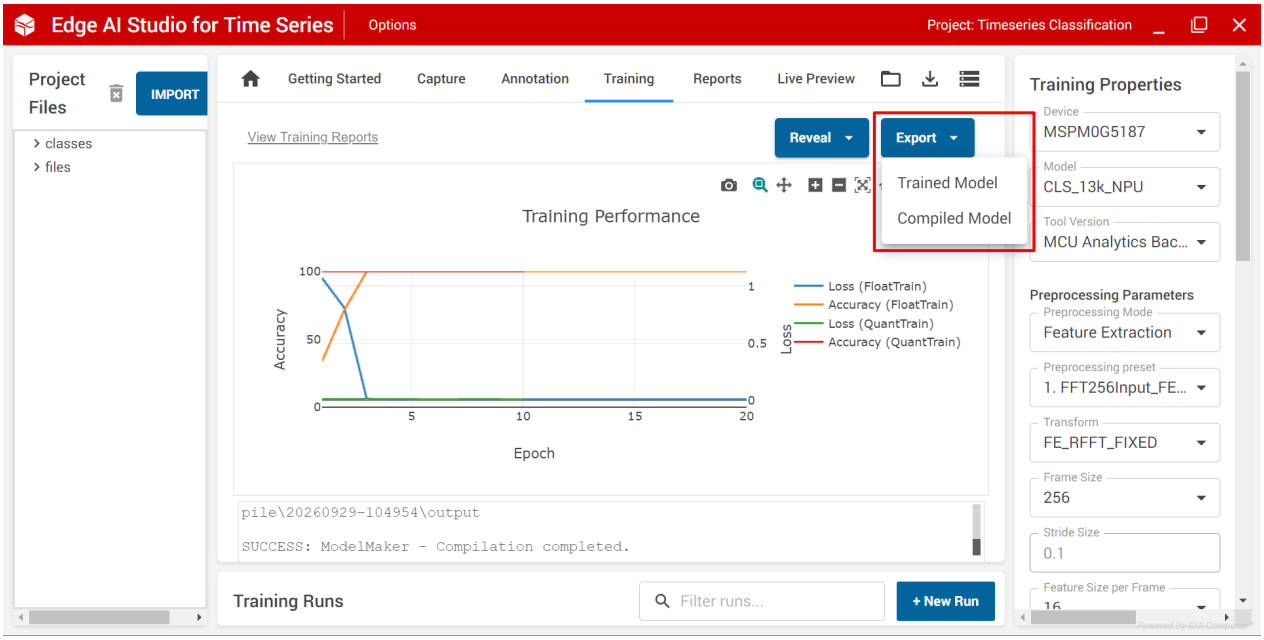


図 4-9. Edge AI Studio からのモデルのエクスポート

エクスポートが成功すると、**トレーニング済みの** NN モデル ファイルと**コンパイル済みの** NN モデル ファイルという 2 つの異なるタイプのアーティファクトが生成されます。詳細なディレクトリ構造とエクスポートされたファイルの具体的な内容については、それぞれ 図 4-10 と 図 4-11 を参照してください。

Copyright © 2026 Texas Instruments Incorporated

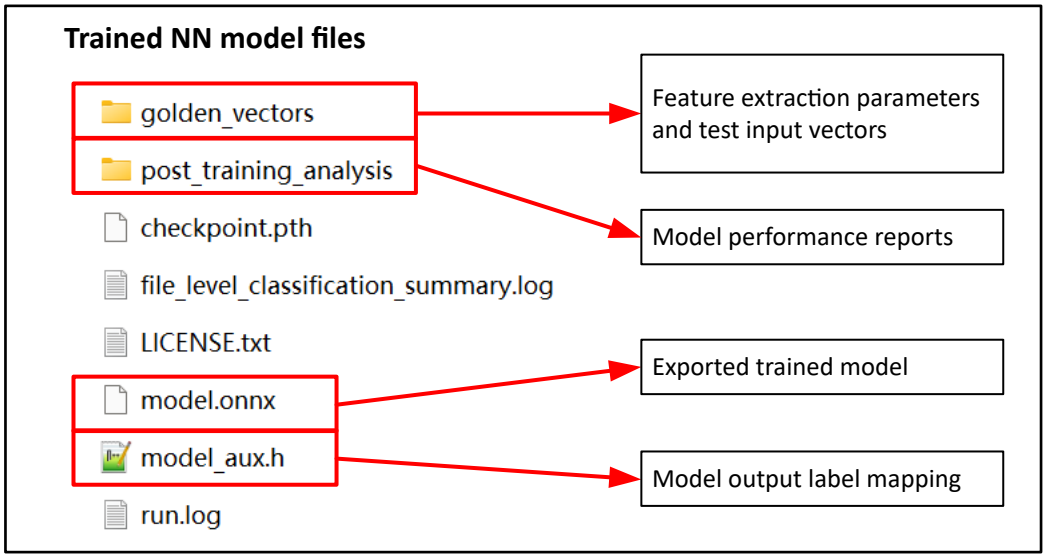


図 4-10. トレーニング済みの NN モデル ファイル

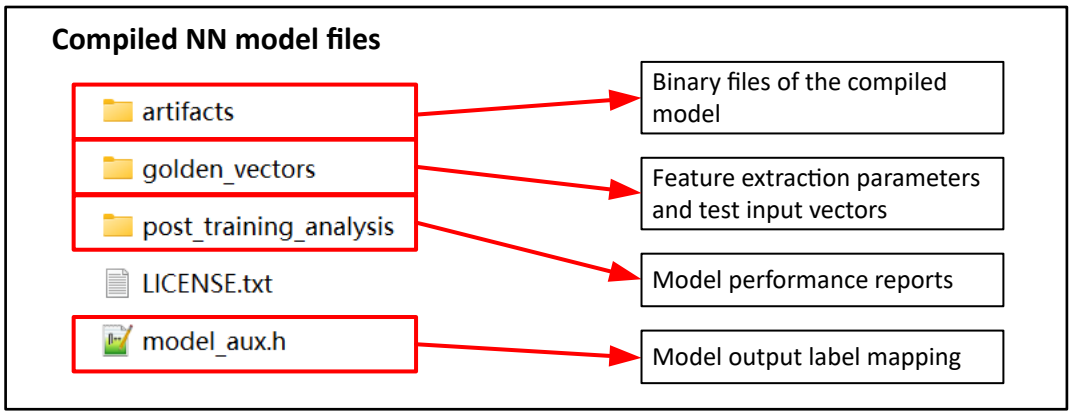


図 4-11. コンパイル済みの NN モデル ファイル

Copyright © 2026 Texas Instruments Incorporated

5 TI Tiny ML Tensorlab による高度な開発

このセクションでは、PyTorch ベースのモデル トレーニング用に最適化されたワークフロー指向ツールセットである Tensorlab について説明します。Tensorlab では、PyTorch のコア モジュールとクラスを活用することで、ニューラル ネットワークのエンド ツー エンドの設計とトレーニングを容易に行うことができます。

高レベルの抽象化ツールとは異なり、Tensorlab は機械学習パイプラインのきめ細かいエンド ツー エンド制御を必要とするプロの開発者向けに設計されています。Tensorlab は、高度にカスタマイズされた開発に必要な柔軟性を提供し、データ アクイジション、モデル アーキテクチャ、トレーニング ハイパーパラメータ、コンパイル ワークフローの正確な操作を可能にします。

以下の実装では、MNIST (数字認識) データセットを基礎ベンチマークとしてニューラル ネットワークをゼロから構築する例を示します。技術的な進歩は、ベースラインの実装から始まり、PyTorch エコシステムの高度な機能を段階的に組み込んでいきます。ワークフローは、ONNX 形式でのモデルのエクスポート、TVM (TI NNC) を使用したコンパイル、PC 側での推論という、デプロイのライフサイクル全体を網羅しています。最後に、TI 固有の NPU 量子化技術を使用してエッジでのモデル性能を最適化する量子化対応トレーニング (QAT) の実装について説明します。

その他資料

- 『エッジ AI ユーザー ガイド (MSPM0-SDK)』
- 『Tiny ML Tensorlab ユーザー ガイド』
- PyTorch の Web ページ:What is torch.nn really?
- Tiny ML ワークフロー サンプル Python スクリプト パッケージ

5.1 環境設定

インストール

Tiny ML Tensorlab をシステムにインストールするには、『Tiny ML Tensorlab ユーザー ガイド』の「インストール」を参照してください。ここで説明するワークフローは、開発者がインストールを行うことを前提に検証されています。この点に注意して、開発環境に合わせてインストール パスを選択してください。

仮想環境 (venv) の構成

環境を分離し、依存関係の競合を防ぐには、すべての Python スクリプトを Tensorlab 仮想環境内で実行する必要があります。すべての操作は、ルート インストール ディレクトリから開始する必要があります。

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

使用するターミナルのインターフェイスに応じて、次のコマンドで環境をアクティブにします。

A. PowerShell

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

B. Git Bash

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

VS Code 構成

開発とデバッグのために仮想環境を Visual Studio Code と統合するには、次の手順を実行します。

1. 次のキーを押して、コマンド パレットを開きます

   ```
   Ctrl + Shift + P
   ```

2. 次のものを検索して選択します。

   ```
   Python: Select Interpreter
   ```


Copyright © 2026 Texas Instruments Incorporated

3. 結果のリストから、venv ディレクトリに関連付けられているインタープリタを選択します

依存関係の確認

環境がアクティブ化されたら、次のコマンドを実行して、インストールされているパッケージを確認し、必要な依存関係がすべて存在することを確認します。

```
pip list
```

5.2 データ収集

モジュール化された検証可能な開発ワークフローを提供するため、データ アクイジション プロセスは、データ取得とデータ検証の 2 つの段階に分かれています。このモジュラー方式により、ネットワークの接続性、ファイル システムの整合性、PyTorch テンソルの互換性の分離テストが可能になります。

5.2.1 データの取得

最初の段階では、MNIST データセットをダウンロードし、ローカル プロジェクト構造内で整理するスクリプトを実行します。実行中にパス関連のエラーが発生しないように、このスクリプトは、現在のファイルの場所に基づいて絶対パスを使用します。これにより、

```
data/
```

ディレクトリが、ターミナルの現在の作業ディレクトリに関係なく、常にスクリプトのディレクトリを基準に作成され、アクセスされます。

1. 次の名前の新しいファイルを作成します

   ```
   01_test_download_data.py
   ```

   作成先は、プロジェクトのルート ディレクトリです。

2. 次のスクリプトを実装します

   ```
   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. ターミナルを使用してスクリプトを実行します。正常に完了すると、ターミナルに次のログが表示されます。

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

5.2.2 データの検証

第 2 段階では、ダウンロードしたファイルの整合性を検証し、未加工データを可視化して、NumPy 配列を PyTorch テンソルに変換します。

Copyright © 2026 Texas Instruments Incorporated

1. 次の名前の新しいファイルを作成します

```
02_test_load_visual_data.py
```

2. 次の検証スクリプトを実装します

```
import pickle
import gzip
from pathlib import Path
from matplotlib import pyplot
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. スクリプトを実行します。図 5-1 に示すように、MNIST セットの最初の手書きの数字がウィンドウに表示されます。画像ウィンドウを閉じると、変換されたテンソル形状と成功メッセージがターミナルに表示されます。正常に実行されると、データセットが正しく読み込まれ、環境が適切に設定され、PyTorch 依存関係が機能しています。

Copyright © 2026 Texas Instruments Incorporated

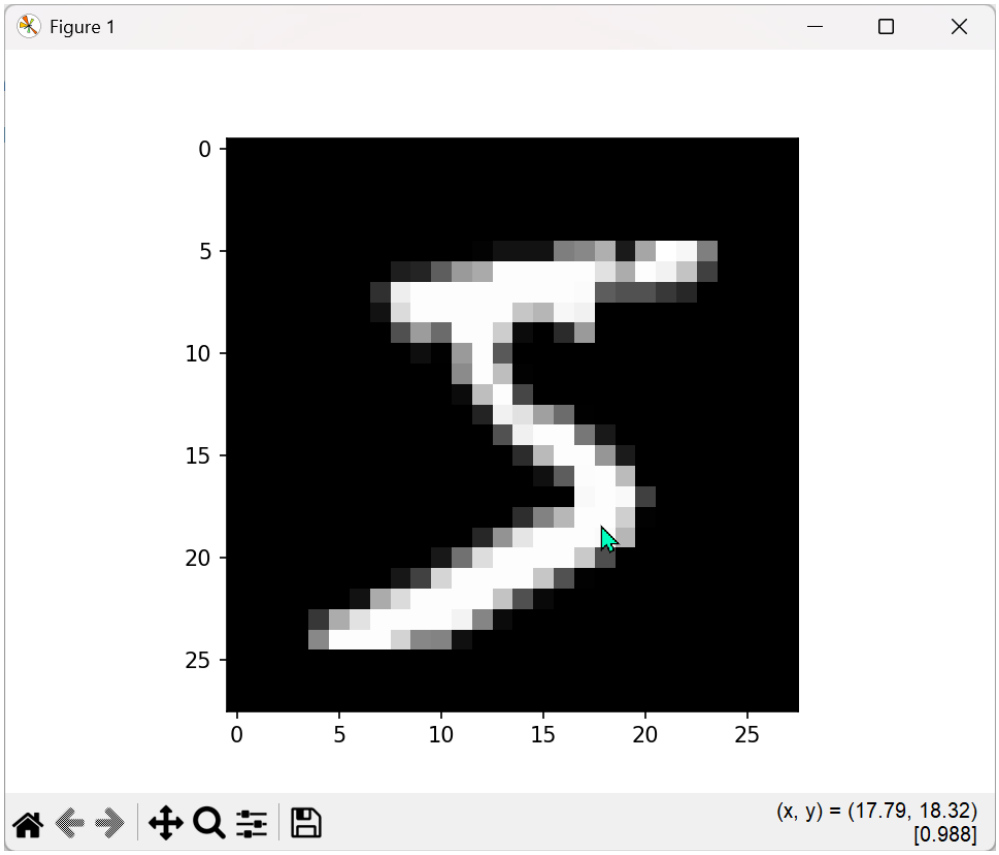


図 5-1. 手書きの数字の画像

注

MNIST データセットは、[0,1] の浮動小数点値の範囲にスケーリングされています。

5.3 モデル アーキテクチャの反復リファクタリング

開発パイプラインが成熟するにつれて、モデル アーキテクチャは基本的な数学的抽象化から複雑なデータ パターンを学習できる高度で高性能な構造へと進化しなければなりません。このセクションでは、線形分類モデルから畳み込みニューラル ネットワーク (CNN) への移行と、PyTorch エコシステムにおけるモジュール性の利点について解説します。

5.3.1 ベースライン実装

最小の線形分類モデルをゼロから実装し、ベースラインを確立します。これにより、*torch.nn* のような高度な抽象化ではなく、基本的な PyTorch テンソル演算を利用することで、ニューラル ネットワークの数学的なメカニズムを理解します。

この実装では、次の 2 つの PyTorch コア機能を利用します。

1. **テンソル演算 :** テンソルとしての重みとバイアスの手動管理。数値的安定性のため、スケーリングされた初期化を適用します。
2. **Autograd エンジン :**

```
requires_grad=True
```

を有効にすることにより、PyTorch がすべてのテンソル操作を追跡できるようになり、バックプロパゲーション時に勾配を自動的に計算するための動的計算グラフが作成されます。

Copyright © 2026 Texas Instruments Incorporated

Python の標準的なロジックによるフォワード パス、活性化関数 (log_softmax)、損失関数 (NLL) の定義は、PyTorch のベクトル化バックエンドにより、カスタマイズされたアーキテクチャを完全に最適化できることを示しています。

サンプル スクリプト

Tiny ML Workflow サンプル Python スクリプト パッケージをダウンロードして、付属の 03_test_train_from_scratch.py スクリプトを使用します。ターミナルでスクリプトを実行した後、ログを監視してトレーニングの進行状況を確認します。

```
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)
```

注

観測された 100% という精度は、検証セットを用意せずにトレーニング セットのみでモデルを評価した結果であり、**オーバーフィッティング**の可能性があります。

5.3.2 PyTorch 組み込みモジュールによるリファクタリング

このセッションでは、PyTorch の高レベルな抽象化機能 nn.Module と torch.optim を活用して、手動テンソル操作から標準化されたワークフローに移行します。このリファクタリングにより、開発パラダイムを低レベルの数学的実装からオブジェクト指向プログラミング (OOP) アプローチに移行します。

nn.Module

モジュール性を向上させるために、モデルは nn.Module から継承したクラス内にカプセル化されます。これには、次のような重要な利点があります。

- **パラメータ管理**: *self.weights* と *self.bias* を手動で定義して初期化する代わりに、*nn.Linear* モジュールがパラメータの初期化と登録を自動的に処理します。
- **自動追跡**: nn.Module を継承することで、すべての内部パラメータが自動的に登録されるため、システムは手動介入なしで、これらのパラメータを勾配計算用に追跡できるようになります。

torch.optim

手動のパラメータ更新ステップ (明示的な **torch.no_grad** コンテキストとパラメータごとの勾配ゼロ化が必要) は、**torch.optim** フレームワークに置き換わります。

- **model.parameters()**: このメソッドはブリッジとして機能し、登録されているすべてのモデル パラメータをオプティマイザに渡します。オプティマイザは、効率的な更新のために、これらのパラメータ ポインタの登録を維持します。
- **opt.step()**: このメソッドは更新ロジックを抽象化します。このメソッドは、登録されているすべてのパラメータを反復処理し、選択した最適化アルゴリズム (SGD など) を適用して、現在の勾配に基づいて重みを更新します。
- **opt.zero_grad()**: 最適化するすべてのパラメータの勾配を一元的な方法でクリアし、次のトレーニング ステップの整合性を維持します。

オプティマイザを使用すると、基盤となるモデル アーキテクチャを変更することなく、コードを 1 行変更するだけで、別の最適化アルゴリズム (たとえば SGD から Adam) にシームレスに切り替えることができます。

Copyright © 2026 Texas Instruments Incorporated

サンプル スクリプト

Tiny ML Workflow サンプル Python スクリプト パッケージをダウンロードして、付属の 04_test_pytorch_module_refactor.py スクリプトを使用します。ターミナルでスクリプトを実行した後、ログを監視してトレーニングの進行状況を確認します。

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

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

--- Post-training Performance ---
Final Training Loss: 0.0554
Validation Accuracy: 0.9245
```

5.3.3 データ パイプラインの最適化

これまでのセクションでは、インデックスを手動で計算し、トレーニングとラベル テンソルを明示的にスライス化していましたが、データセットと DataLoader の抽象化を実装することで、より堅牢なエンジニアリング ワークフローを実現できます。これらの抽象化により、データ ストレージがバッチ処理から効果的に分離され、スケーラビリティと効率性の両方が向上します。

TensorDataset

特徴量とラベルを別々のエンティティとして管理するには、シャッフルまたはスライス中にインデックスが整列していることを確認するため、慎重に同期する必要があります。TensorDataset は、複数のテンソルを 1 つのデータセット オブジェクトにまとめることで、この問題を解決します。カプセル化された後、インデックスにアクセスすると、対応する特徴テンソルとラベルを含むタプルが自動的に返され、パイプライン全体の構造的整合性が維持されます。

DataLoader

データセットがストレージを処理する間、DataLoader はデータ配信のロジスティクスを管理します。DataLoader には、次のような基本的なエンジニアリング機能が用意されています。

- **データ シャッフル (shuffle = True)**: モデルがデータのシーケンスを学習しないようにするため (オーバーフィットの原因となります)、DataLoader は各エポックの開始時にデータセットを再シャッフルし、ミニバッチ選択のランダム性を維持します。
- **並列読み込み (num_workers)**: I/O (ディスク読み取り) がボトルネックになる大規模アプリケーションでは、**DataLoader** はマルチプロセス データ読み込みを利用できます。これにより、GPU が現在のバッチを処理している間、CPU はバックグラウンドで次のバッチをプリフェッチして準備できるため、ハードウェアを最大限活用できます。

これらのツールを利用することで、複雑なインデックス演算からクリーンで反復可能なインターフェースまで、トレーニング ループが簡略化されます。

サンプル スクリプト

Tiny ML Workflow サンプル Python スクリプト パッケージをダウンロードして、付属の 05_test_data_pipeline_refactor.py スクリプトを使用します。ターミナルでスクリプトを実行した後、ログを監視してトレーニングの進行状況を確認します。

```
--- 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 ---
```


Copyright © 2026 Texas Instruments Incorporated

```
>> Test Loss: 0.3143
>> Test Accuracy: 91.03%
```

5.3.4 CNN アーキテクチャへの移行

堅牢なモジュール型のトレーニング インフラを導入したことで、線形アーキテクチャを畳み込みニューラル ネットワーク (CNN) に置き換え、高度なコンピュータ ビジョン (CV) タスクをサポートするようにパイプラインをアップグレードできます。

線形モデルの主な限界は空間階層を保持できないことであり、28x28 の画像を 784 要素のベクトルにフラット化すると、ピクセル間の局所的な幾何学的関係が破壊されます。CNN アーキテクチャは、次の 2 つのメカニズムでこの課題を解決します。

- **テンソルの形状変換:** *xb.view(-1, 1, 28, 28)* を使用して、入力を 4D テンソル (バッチ、チャンネル、高さ、幅) に変換します。これにより、畳み込み操作に必要な空間グリッドが復元されます。
- **階層型特徴学習:** このモデルは、チャンネル数を増やした一連の **nn.Conv2d** レイヤーを使用します。このネットワークは最初のレイヤーの低レベル エッジから、最終的な畳み込みレイヤーの複雑で高レベルの数字の輪郭まで、特徴の階層を抽出することができます。

収束速度と安定性を向上させるために、SGD オプティマイザは momentum = 0.9 項を含むようにアップグレードされます。標準 SGD では、損失ランドスケープの平坦な領域で更新が停止したり、大きく振動したりする可能性があります。モメンタムは、過去の勾配の蓄積によって物理的な「慣性」をもたらし、オプティマイザが平坦部を横断し、狭い谷をより効率的に通過できるようにします。

トレーニング ロジック (loss_batch および fit) はネットワーク アーキテクチャから完全に分離されているため、この移行ではコア トレーニン グループに対する変更は不要です。

サンプル スクリプト

Tiny ML Workflow サンプル Python スクリプト パッケージをダウンロードして、付属の 06_test_cnn_model_refactor.py スクリプトを使用します。ターミナルでスクリプトを実行した後、ログを監視してトレーニングの進行状況を確認します。

```
--- 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 <<
```

nn.Sequential によるアーキテクチャの合理化

以前に実装された CNN アーキテクチャでは、nn.Module を使用してカスタム レイヤーを定義していますが、*forward* メソッドでは、データフローを手動で明示的に定義する必要があります。このアプローチは柔軟性に優れていますが、クラス構造内のすべてのテンソル変換を手動で制御しなければなりません。

このプロセスを効率化するために、PyTorch は nn.Sequential コンテナを提供しています。このユーティリティを使用すると、開発者は複数のレイヤーを 1 つのモジュール単位にカプセル化でき、データは定義された正確な順序でレイヤーを通過するようになります。このパイプライン アプローチにより、ボイラープレート コードが大幅に削減され、標準的なフィード フォワード アーキテクチャの **forward** メソッドを手動で実装する必要がなくなります。

nn.Sequential を使用する場合の一般的な課題は、カスタム スライス操作を実行したり、後続の処理の前に特定の形状を調整したりするなど、標準のネイティブ レイヤーとしては存在しない中間テンソル操作を実行する必要がある点です。標準の nn.Module では、*forward* 関数内で簡単に処理できます。ただし、厳密な nn.Sequential パイプラインでは、すべてのステップを有効な nn.Module オブジェクトとしてカプセル化する必要があります。これを実現するために、全く新しい独立したクラスを記述する必要はありません。それぞれの変換に Lambda ラッパーを利用できます。

Copyright © 2026 Texas Instruments Incorporated

サンプル スクリプト

Tiny ML Workflow サンプル Python スクリプト パッケージをダウンロードして、付属の 07_test_cnn_sequential_refactor.py スクリプトを使用します。ターミナルでスクリプトを実行した後、ログを監視してトレーニングの進行状況を確認します。

```
--- 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 モデルのエクスポート

トレーニング フェーズ中に目的の精度に達したら、モデルを Open Neural Network Exchange (ONNX) 形式に変換し、PyTorch 開発環境からデプロイ可能な形式に移行できます。

ONNX は、相互運用可能な中間表現として機能します。*ONNX Runtime* などのランタイムを使用すると、PyTorch 全体をインストールすることなく、C++、Android、iOS、Web ブラウザなどのさまざまなプラットフォームで評価推論を実現できます。

次のスクリプトは、既存のトレーニング ワークフローを拡張して、堅牢なモデル エクスポート ステップを含める方法を示しています。この実装には、ディレクトリ管理と特定の構成が含まれており、エクスポートされたモデルがダウンストリーム推論エンジンと互換性があることを確認できます。

サンプル スクリプト

Tiny ML Workflow サンプル Python スクリプト パッケージをダウンロードして、付属の 08_test_export_onnx.py スクリプトを使用します。ターミナルでスクリプトを実行した後、ログを監視してトレーニングの進行状況を確認します。

```
--- Commencing Sequential CNN Training ---
Epoch 0: Val Loss = 0.3461 | Val Accuracy = 0.8950
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 量子化対応トレーニング

量子化は、リソースに制約のあるエッジ デバイスに機械学習モデルを導入するための重要な最適化手法です。量子化では、モデルの重みと活性化の精度を低下させ、32 ビット浮動小数点 (FP32) から 8 ビット (INT8)、4 ビット (INT4)、2 ビット (INT2) 整数などの低ビット幅に変換します。

このプロセスにより、エッジ AI に以下の 3 つの主な利点が得られます。

- **モデルの圧縮:** メモリ フットプリントが大幅に削減されます (たとえば、FP32 から INT8 への変換では 4 分の 1 に縮小されます)。
- **推論速度の向上:** 専用ハードウェア上での浮動小数点演算よりも、整数演算のほうが計算効率が高くなります。
- **ハードウェア アクセラレーション:** TI のニューラル プロセッシング ユニット (NPU) は、整数ベースの計算用にアーキテクチャが最適化されているため、量子化は高速実行の前提条件となります。

ビット幅が小さくなると、精度が低下する可能性がありますが、モデルの処理効率は向上します。表 5-1 に、量子化で見られる代表的な性能特性を示します。

Copyright © 2026 Texas Instruments Incorporated

表 5-1. 精度と効率のトレードオフ

精度	サイズの低減	速度の向上	標準的な精度低下
Float32	1x (ベースライン)	1x (ベースライン)	0%
INT8	4x	2x ～ 4x	1% 未満
INT4	8x	3x ～ 6x	1% ～ 5%
INT2	16x	4x ～ 8x	5% ～ 15%

低いビット幅で高い精度を実現するには、量子化対応トレーニング (QAT) を推奨します。QAT はトレーニング プロセス中の量子化誤差をシミュレートし、モデルの重みを精度低下に適応させることができます。詳細については、以下を参照してください。

- 『Tiny ML Tensorlab ユーザー ガイド』の「量子化」
- 『Tiny ML Tensorlab ユーザー ガイド』の「スタンドアロン量子化の例」

サンプル スクリプト

Tiny ML Workflow サンプル Python スクリプト パッケージをダウンロードして、付属の 09_test_cnn_quantization_model.py スクリプトを使用します。FP32 トレーニングから INT8 量子化対応トレーニング、ONNX 形式でのエクスポートまでの完全なパイプラインを確認できます。

次のログは、ベースライン FP32 トレーニング、量子化対応トレーニング (QAT) フェーズ、量子化モデルの正常なエクスポートまでのスクリプト全体の実行結果を示しています。

```
--- 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
--- 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 モデルのコンパイル

モデルのトレーニングと ONNX 形式へのエクスポートが完了した後、開発ライフサイクルの最後のステップはコンパイルです。このプロセスでは、高水準の ONNX 計算グラフを、ターゲット マイコン (MCU) 用に最適化されたマシン コードに変換します。

TI のニューラル ネットワーク コンパイラ (NNC) は、汎用 ONNX モデルと組み込みデバイスの間のギャップを埋める役割を果たします。NNC は、プロジェクトの要件に応じて、モデルを 2 つの主要な展開形式のいずれかに変換します。

- **静的ライブラリ (.a):** モデルは、クロスコンパイラ (たとえば tiarmclang) を使用して静的ライブラリにコンパイルされます。これは量産化における標準的な方法であり、ライブラリを Code Composer Studio™ (CCS) プロジェクトに直接リンクします。
- **ソース コード パッケージ (.tar):** モデルは、可読形式の C ソース ファイルに分解されます。このアプローチは、生成されたカーネルの詳細な検査や、バイナリの直接リンクが制限されている環境に最適です。このモードでは、コンパイル フェーズ中に外部のクロス コンパイラは必要ありません。

Copyright © 2026 Texas Instruments Incorporated

出力ファイルの構造

コンパイラによって生成される出力ファイルは、標準化されたディレクトリ構造に従います。

```
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
```

コマンドライン インターフェイス (CLI)

上級ユーザーの場合は、*tvmc* コマンドライン インターフェイスを使用してコンパイル プロセスを詳細に制御できます。詳細な資料については、『マイコン向け TI ニューラル ネットワーク コンパイラ ユーザー ガイド』の「コンパイルの説明」を参照してください。

ONNX モデルを MSPM0 (Cortex-M0+) ターゲット用の静的ライブラリにコンパイルするには、ターミナルで次のコマンドを実行します。

```
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"
```

サンプル スクリプト

Tiny ML Workflow サンプル Python スクリプト パッケージをダウンロードして、付属の 10_test_cnn_quan_compile_npu_lib.py スクリプトを使用します。このスクリプトは、*tvmc* コマンドライン インターフェイスの汎用ラッパーを提供します。このスクリプトを使用すると、開発者は最小限の労力でターゲット固有のコンパイラ フラグを管理できます。

ターミナルでスクリプトを実行した後、ログを監視してトレーニングの進行状況を確認します。

```
============================================================
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]
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: ------------------------------------------------------------
```


Copyright © 2026 Texas Instruments Incorporated

```
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: ==========================================================
```


Copyright © 2026 Texas Instruments Incorporated

6 エッジ AI アプリケーション：数字認識

数字認識は、組込み機器市場やコンシューマ市場の幅広いアプリケーションで活用される基盤技術です。しかし、信頼性の高い認識を組込みシステムに展開することは容易ではありません。従来のテンプレート マッチング方式では、人の手書き文字に発生しがちなバラツキに十分対応できないためです。そうした中で登場したエッジ AI により、有力な設計が可能になりました。クラウド接続に依存することなく、デバイス上で直接、高精度かつリアルタイムの推論を実行できるようになったのです。CPU ベースの推論パイプラインは膨大な演算リソースを必要とするため、リソース制約のある一般的なマイコンの処理能力では対応できません。

MSPM0G5187 マイコンはこの課題に対応するため、最小限の電力とメモリ オーバーヘッドで ML のワークロードを高速化するように設計された専用の TinyEngine NPU を採用しています。この設計では、1 桁の数字が書かれている画像を 10 種類のカテゴリ (0 ～ 9) のいずれかに分類する処理を実現します。このアプリケーションは、UART バック チャネル経由で 28×28 のグレイスケール画像をピクセル データとして受信し、オンチップの NPU を使用してハードウェア アクセラレーション モデルの推論を実行し、認識された数字クラスをホスト GUI に送信して表示します。

オンチップ NPU を活用することで、この数字認識システムは、約 99% の分類精度と 6.05ms の推論レイテンシを実現しています。また、メモリの消費量はフラッシュ メモリでわずか 73KB、RAM で 10.9KB です (表 6-1 を参照)。これらの結果から、リソース制約のあるマイコン上でもオンデバイス機械学習が実用可能であることが分かります。

表 6-1. 数字認識エッジ AI 設計のパフォーマンス

メトリック	値
精度	約 99%
フラッシュの使用率	73KB
RAM の使用法	10.9KB
推論レイテンシ (NPU)	6.05ms
推論エネルギー(1) (AVG)	170.38uJ

(1) この推論エネルギーは、tvmgen_default_run () を繰り返し実行している間に、EnergyTrace™ ソフトウェアを使用して測定しました。

注

LP-MSPM0G5187 で正確な推論エネルギー測定を維持するには、J6 ジャンパを取り外します。これにより、オンボード マイクおよびオーディオ ADC による追加の消費電流が防止されます。この処理を行わないと、結果が偏る可能性があります。

表 6-2 に、モデルの主な情報を示します。

表 6-2. 数字認識 AI モデルの情報

特性	値
モデル アーキテクチャ	CNN
パラメータ数	60,000
入力形状	(1, 28, 28)
出力クラス	10
量子化	INT8

6.1 LeNet–5 バリアント CNN モデル

LeNet は、小さなグレイスケール画像から手書きの数字と文字を認識するために設計された、畳み込みニューラル ネットワーク (CNN) アーキテクチャの先駆的なシリーズです。数字認識エッジ AI モデルは、TI のニューラル ネットワーク コンパイラを使用して NPU ネイティブ命令へコンパイルされた LeNet-5 派生の CNN モデルです。

図 6-1 に、そのワークフローを示します。この設計では、784 要素の float32 入力としてフラット化された 28×28 グレイスケール画像を受け取り、クラスごとの信頼スコアを表す 10 要素の float32 出力を生成します。このネットワークは、2 つの連続した畳み込みブロックで構成され、それぞれバイアス付きの Conv2D レイヤ、ReLU 活性化、および 2×2 MaxPooling で構成されています。これらのブロックは、空間分解能を段階ごとに半分にしながら、空間的特徴を徐々に抽出します。その後、得られた特徴マップはフラット化され、一連の完全に接続されたレイヤを通過して渡されます。これ

Copyright © 2026 Texas Instruments Incorporated

により、高次元の特徴が 10 クラスの出力にマッピングされます。すべてのレイヤは INT8 の量子化重み付けと活性化で動作するため、完全に接続された最終レイヤの後に量子化の後処理ステップが適用され、出力を float32 に再スケーリングします。予測される数字は、ホスト側で 10 個の出力スコアに対して argmax を適用することで決定します。

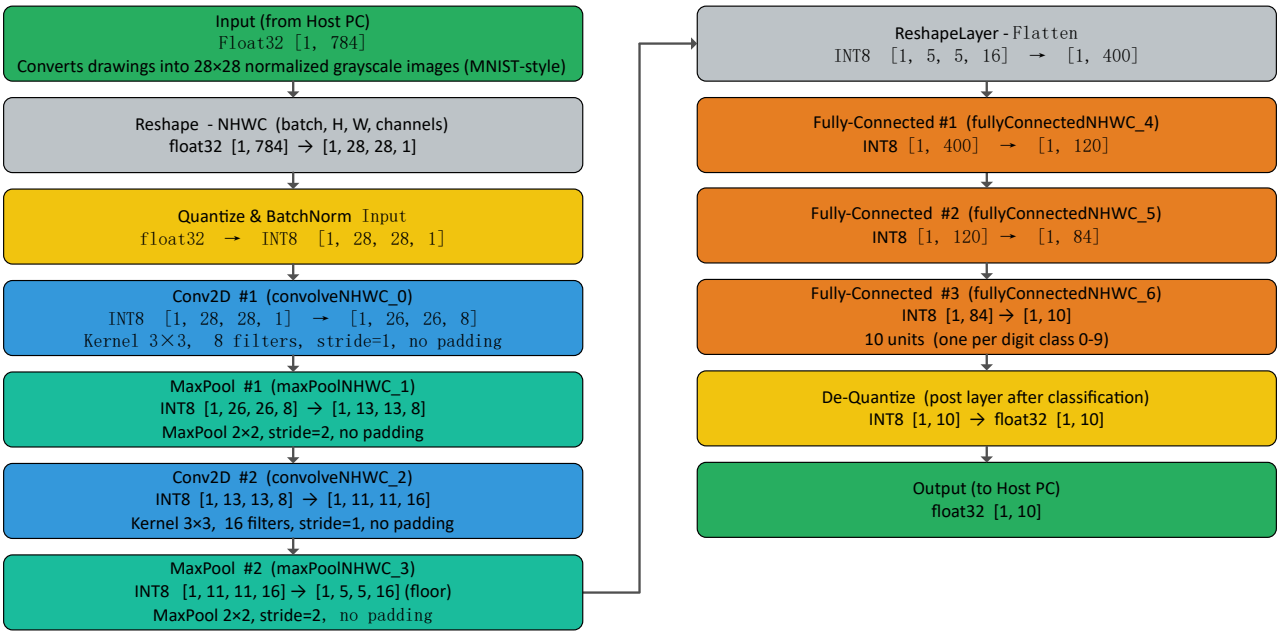


図 6-1. 数字認識 AI モデルのフロー チャート

6.2 NPU/CPU の性能比較

エッジ AI モデルは、TI ニューラル ネットワーク コンパイラを用いてハードウェアにデプロイでき、専用ハードウェア NPU またはホスト CPU のいずれかをターゲットとします。TinyEngine NPU は、高効率でニューラル ネットワーク計算を実行する専用ハードウェア アクセラレータであり、汎用 CPU に比べて推論レイテンシとエネルギーが大幅に低減されます。

性能評価を容易にするため、MSPM0 SDK はユーザー ベンチマーク用として NPU と CPU の両方の実装例を提供しています。以下を参照してください。

- NPU ベースの数字認識エッジ AI の例
- CPU ベースの数字認識エッジ AI の例

表 6-3 に、数字認識アプリケーションにおける NPU ベースと CPU ベースのエッジ AI 設計の性能比較を示します。

表 6-3. NPU/CPU の性能比較

性能指標	NPU ベースの設計	CPU ベースの設計
精度	約 99%	約 99%
フラッシュの使用率	73KB	68KB
RAM の使用法	10.9KB	8.1KB
推論レイテンシ	6.05ms	89.81ms
推論エネルギー (AVG)	170.38uJ	2,447.45uJ

NPU ベースの設計は、CPU ベースの実装に比べて、推論レイテンシが約 15 分の 1 に短縮されます。また、推論 1 回の電力消費が約 96% 低減するという実質的な効果もあるため、電力制約の厳しいエッジ アプリケーションにおいて高い効率性を実現します。

Copyright © 2026 Texas Instruments Incorporated

7 エッジ AI アプリケーション：波形分類

MSPM0 は、入力信号をリアルタイムかつ連続的に検出、分類する機能を備えた実験的な波形分類設計です。正弦波、三角波、のこぎり波などの一般的な波形を識別できます。この設計は、13K パラメータの時系列分類モデルに基づいており、高いスケーラビリティと拡張性を備えています。

この波形分類システムは、100% (R の 2 乗) の分類精度を達成し、推論レイテンシはわずか 5.84ms です。また、メモリの消費量はフラッシュで 21.5KB、RAM で 3.3KB のみです (表 7-1 を参照)。

表 7-1. 波形分類エッジ AI ソリューションの性能

メトリック	値
精度	100%
フラッシュの使用率	21.5KB
RAM の使用法	3.3KB
推論レイテンシ (NPU)	5.84ms
推論エネルギー(1) (AVG)	236.97uJ

(1) 推論エネルギーは、*tvmgen_default_run ()* を繰り返して実行している間に、EnergyTrace を使用して測定しました。

注

LP-MSPM0G5187 で正確な推論エネルギー測定を維持するには、J6 ジャンパを取り外します。これにより、オンボード マイクおよびオーディオ ADC による追加の消費電流が防止されます。この処理を行わないと、結果が偏る可能性があります。

表 7-2 に、モデルの主な情報を示します。

表 7-2. 波形分類 AI モデルの情報

特性	値
モデル アーキテクチャ	CNN
パラメータ数	14,124
入力形状	テンソル [(1, 1, 128, 1)]
出力クラス	3 (のこぎり波、正弦波、方形波)
量子化	INT8

Copyright © 2026 Texas Instruments Incorporated

7.1 特徴抽出

特徴抽出は、一般的なエッジ AI アプリケーション、特にオーディオと時系列の信号処理にとって重要なコンポーネントです。この特徴抽出モジュールは、組込みシステム向けに最適化された効率的な信号処理機能を提供し、ARM ライブラリを活用して最適な性能を実現します。使用可能な特徴抽出機能には、次のものが含まれます。

- 実数高速フーリエ変換 (RFFT)：時間ドメインの信号を周波数ドメイン表現に変換します
- 振幅スケーリング：信号の大きさを適切な範囲に正規化するためのスケーリングを適用し、以後の処理でのオーバーフロー / アンダーフローを防止します
- 複素振幅の計算：FFT により得られる複素数の振幅を計算します
- DC コンポーネントの除去：特徴量の精度を向上させるため、信号の DC バイアスを除去します
- ビン計算：定義されたビン内の周波数成分を平均化し、次元数を削減します

図 7-1 に、特徴抽出処理のパイプラインを示します。

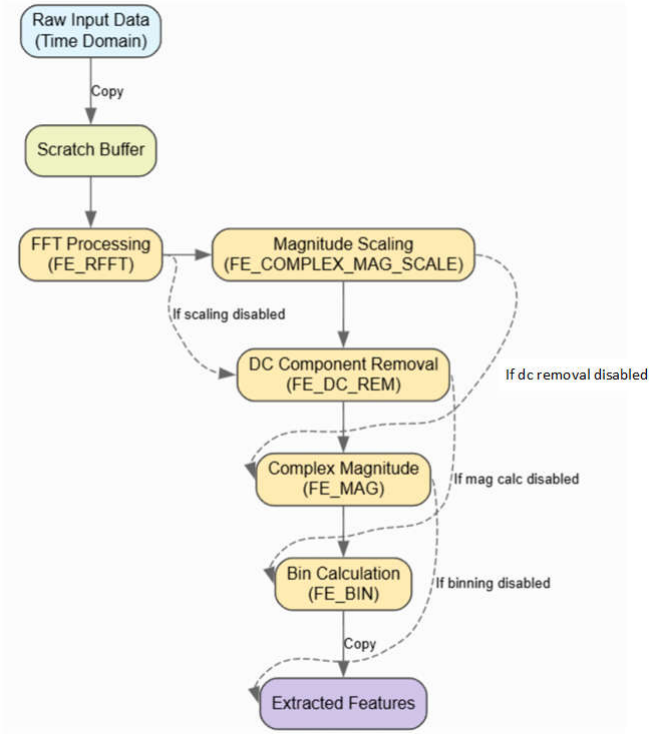


図 7-1. 特徴抽出処理のパイプライン

表 7-3 に、波形分類用に設計された特徴抽出処理パイプライン構成を示します。対応する構成シンボルが定義されると、関連タスクが処理パイプライン内で実行されます。

表 7-3. 波形分類のパイプライン構成

処理タスク	構成シンボル	定義の有無
RFFT	FE_RFFT	Y
振幅スケーリング	FE_COMPLEX_MAG_SCALE	Y
DC コンポーネントの除去	FE_DC_REM	Y
複素振幅の計算	FE_MAG	Y
ビン計算	FE_BIN	Y

Copyright © 2026 Texas Instruments Incorporated

表 7-4 に、波形分類用に設計された特徴抽出構成パラメータを示します。推論タスク 1 回あたり、8 フレームを連結します。各フレームでは 256 個のデータ ポイントを処理して 16 次元の特徴量を生成し、合計 128 次元の特徴量を波形分類 AI モデルに入力します。

表 7-4. 波形分類特徴の抽出パラメータ

メトリック	構成シンボル	値
入力フレーム サイズ	FE_FRAME_SIZE	256
フレームあたりの特徴サイズ	FE_FEATURE_SIZE_PER_FRAM	16
周波数ビン サイズ	FE_BIN_SIZE	8
ビン値を正規化します	FE_BIN_NORMALIZE	1
連結するフレーム	FE_NUM_FRAME_CONCAT	8
入力特徴サイズ	FE_STACKING_FRAME_WIDTH	128

注

表 7-4 で構成されている特徴量抽出パラメータは、モデル学習時のパラメータと同一に設定する必要があります。これにより、一貫した結果を確認できます。

性能に関する検討事項

- 最適化計算：効率的な処理のために、ARM 最適化済みのライブラリを活用してください
- メモリ効率：バッファのスワップにより、メモリ コピーのオーバーヘッドが最小限に抑えられます
- 次元数の削減：ビン計算により、特徴の次元が小さくなり、下流のステージでのメモリ使用量と計算量が減少します
- パラメータの調整：使用可能なメモリと計算リソースに合わせて構成パラメータを調整する必要があります

Copyright © 2026 Texas Instruments Incorporated

7.2 時系列分類モデル

時系列分類は、時系列データ内の特定のカテゴリ、運用状態、または事前定義されたラベルを識別し、ヘルスケア、ワイヤレス モニタリング、スマート セキュリティ、自動テストなどの領域に広く適用されています。

波形分類設計では、軽量な 13K パラメータの時系列分類モデル (CLS_13k_NPU) が実装されており、組込みニューラル プロセッシング ユニット (NPU) への効率的な展開を目的として設計されています。この設計では、256 ポイントの実数高速フーリエ変換 (RFFT) による特徴抽出パイプラインと、AI 推論タスクごとに 8 フレームを連結する方式を採用しており、入力として 128 要素の 1 次元時系列特徴ベクトルを受け取り、出力として 3 要素のベクトルを生成します。各要素は、それぞれ対応するクラスの信頼度スコアを表します。

エクスポートされたモデル ファイルは、完全に接続された最終レイヤから INT8 (8 ビット量子化) テンソルを出力します。生の出力は、量子化された形式のクラス別信頼スコアで構成されているため、最終的な分類結果を導き出すには後処理ステップが必要です。

波形分類モデルのアーキテクチャ

モデル ネットワークはシーケンシャル特徴変換パイプラインとして構成されており、各ステージは前の出力に直接入力として構築され、入力表現を段階的に洗練、抽象化しながら、最終的な分類判断を行います。図 7-2 にワークフローを示します。

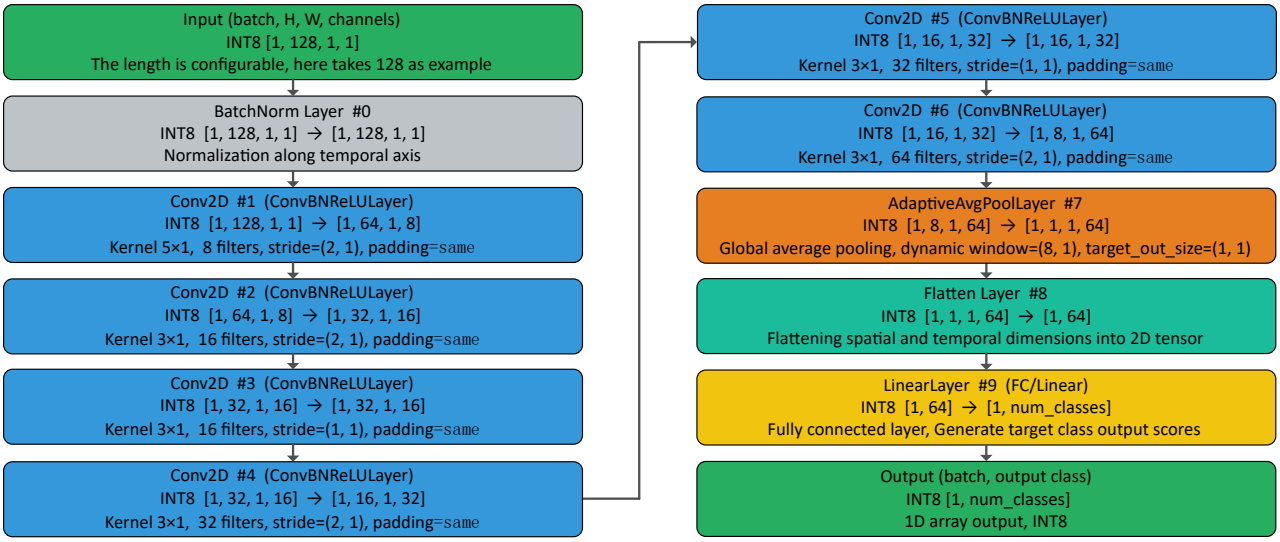


図 7-2. 時系列分類モデル (CLS_13k) のフロー チャート

このパイプラインはバッチ正規化レイヤーから始まります。このレイヤーは生の入力シーケンスをリアルタイムで標準化し、学習済みの変換を適用する前に入力信号の振幅が変動しても一貫した特徴スケーリングを行います。入力が安定すると、ネットワークは 6 つの連続した畳み込みブロックを介して特徴表現を段階的に深めていきます。各ブロックは一貫した内部構造 (バイアスのある Conv2D レイヤー、バッチ正規化、ReLU 活性化) の順で構成されています。この繰り返し構造により、各ステージを経るごとに、抽象度を高めた時系列パターンをキャプチャします。各ブロック内のバッチ正規化は、深層化してもトレーニングの安定性を維持しますが、ReLU 活性化は、モデルが複雑な信号特性を学習するために必要な非線形性を付与します。

6 つの畳み込みステージの後、得られた特徴マップはアダプティブ平均プーリング レイヤーを通過します。このレイヤーは、前の時間的長さに関係なく、空間分解能を 1x1 の固定サイズに縮約します。この設計の大きな特徴は、モデルを固定の時間次元に依存させない点です。これにより、長さの異なる時系列入力との互換性が本質的に確保され、アーキテクチャを変更することなく、将来的にさまざまな展開シナリオに柔軟に対応できます。

固定サイズにプールされた特徴マップは、2D テンソルにフラット化され、最終的な完全接続 (FC) レイヤーを通過します。このレイヤーは、圧縮された高次元特徴表現をターゲット クラスの出力スコアに直接マッピングします。

Copyright © 2026 Texas Instruments Incorporated

モデル出力と後処理

エクスポートされたモデル ファイルは、完全に接続された最終レイヤから INT8 (8 ビット量子化) テンソルを出力します。生の出力は、量子化された形式のクラス別信頼スコアで構成されているため、最終的な分類結果を導き出すには後処理ステップが必要です。通常は、出力ベクトルに argmax 演算を適用し、最も高い信頼度スコアを持つクラスのインデックスを、予測分類結果とします。

7.3 モデル メモリに関する検討事項

リソースに制約のある組込みプラットフォーム上でエッジ AI 設計を開発するには、アルゴリズムの性能 (精度、レイテンシ)、不揮発性ストレージ (フラッシュ / ROM)、ランタイム メモリ (SRAM) という 3 つの要素を厳密にトレードオフしながら最適化する必要があります。

- **アルゴリズムのパフォーマンス**：推論の精度とレイテンシの両方が含まれます。これらは主に、ネットワークの深さ、レイヤ幅、演算子の複雑さなどのモデル アーキテクチャによって形成されます。
- **静的ストレージ メモリ (フラッシュ/ROM)**：これは主にモデルのパラメータの合計数 (重みとバイアス) によって決まります。完全に量子化された INT8 パイプラインでは、各パラメータはちょうど 1 バイトのフラッシュ ストレージに直接マップされます。そのため、チャネル幅が拡張されたより深いネットワークでは、静的バイナリ サイズが線形的に増加します。場合によっては、入力となる特徴の次元の変化が、この静的メモリ使用量に影響を与えることがあります (たとえば、完全に接続されたレイヤー)。
- **動的ランタイム メモリ (SRAM/RAM)**：これは、入力した特徴のサイズと、中間層をまたぐ活性化テンソル (特徴マップ) によって制御されます。時系列スライスがネットワークを介して伝播するとき、処理コアはレイヤの入力と出力を格納するために一時的なワークスペースを割り当てる必要があります。入力時間ウィンドウが長いか、特徴の次元が大きいと、このピーク ランタイム RAM 要件が指数関数的に増大します。

このようにハードウェアの現実的な制約があるため、エッジ環境への展開では従来の精度重視の考え方から脱却する必要があります。展開を成功させるには、静的フラッシュおよびピーク時の動的 SRAM という厳しい物理的な制約と、未加工モデルのパフォーマンスとの間で、きめ細かくバランスを取る必要があります。

この関係を説明するために、表 7-5 では、さまざまなモデル サイズと入力した特徴の構成に対して波形分類のメモリ フットプリントとリソース使用率を定量化しています (Edge AI Studio 1.6.1 クラウド バージョンでテスト)。

表 7-5. モデル メモリ解析

モデル バリアント (パラメータ)	フラッシュ (ROM) のサイズ	RAM サイズ、入力 = 64	RAM サイズ、入力 = 128	RAM サイズ、入力 = 256
TimeSeries_Generic_1K	約 5.6 KB	約 2.7 KB	約 5.3 KB	約 10.4 KB
TimeSeries_Generic_4K	約 9.9 KB	約 1.2 KB	約 1.7 KB	約 2.7 KB
TimeSeries_Generic_13K	約 21.4 KB	約 2.3 KB	約 3.3 KB	約 5.3 KB

すべての入力次元において、最小モデル (TimeSeries_Generic_1K) は、大規模モデル (TimeSeries_Generic_4K および TimeSeries_Generic_13K) の 2 倍から 4 倍のランタイム RAM を消費します。一見直感に反するこの挙動は、パラメータ数とランタイム メモリが根本的に異なるアーキテクチャ設計に基づいているという事実に起因します。1K モデルは、初期にダウン サンプリングしない浅いトポロジを使用しているため、高分解能の大きな特徴マップが中間レイヤー全体のメモリに保持されます。これとは対照的に、4K と 13K のモデルは、ネットワークのフロントエンドにストライド付き畳み込みを用いた深いアーキテクチャを採用しているため、時間次元が直ちに縮減され、中間のランタイム テンソルのサイズが大幅に小さくなります。

Copyright © 2026 Texas Instruments Incorporated

7.4 NPU/CPU の性能比較

エッジ AI モデルは、TI ニューラル ネットワーク コンパイラを用いてハードウェアにデプロイでき、専用ハードウェア NPU またはホスト CPU のいずれかをターゲットとします。性能評価を容易にするため、MSPM0 SDK はユーザー ベンチマーク用として NPU と CPU の両方の実装例を提供しています。以下を参照してください。

- NPU ベースの波形分類エッジ AI の例
- CPU ベースの波形分類エッジ AI の例

表 7-6 に、波形分類アプリケーションにおける NPU ベースと CPU ベースのエッジ AI 設計の性能比較を示します。

表 7-6. NPU/CPU の性能比較

性能指標	NPU ベースの設計	CPU ベースの設計
精度	100%	100%
フラッシュの使用率	21.5KB	21.7KB
RAM の使用法	3.3KB	3.1KB
特徴抽出レイテンシ	10.75ms	10.93ms
特徴抽出エネルギー(1) (AVG)	386.68uJ	393.15uJ
推論レイテンシ	5.77ms	54.42ms
推論エネルギー (AVG)	236.97uJ	2039.20uJ

(1) 特徴抽出エネルギーは、FE_PROCESS() を繰り返し実行している間に EnergyTrace を使用して測定しました。

ハードウェアの演算効率を比較したところ、NPU アクセラレーションを採用したアーキテクチャは、ベースラインの CPU 実装に対して、性能と電力特性の両方で大きな優位性を示しました。

推論処理のみを見ると、NPU ベースの設計は優れた高速化を実現しており、実行レイテンシを約 8 分の 1 に低減するとともに、推論 1 回当たりのエネルギー消費量を約 88% 削減しています。

フロントエンドの特徴抽出処理を含めたエンド ツー エンドのシステム全体で評価すると、CPU が実行する信号前処理の負荷が一定であるため、全体の効率向上効果は一部相殺されます。それでも、NPU ベースのパイプラインでは、レイテンシが約 3 分の 1 に低減し、実行 1 回当たりのエネルギー消費量が約 74% 低減しています。

Copyright © 2026 Texas Instruments Incorporated

8 まとめ

このアプリケーション ノートは、MSPM0G5187 マイコンをベースとするエッジ AI ソリューションの包括的な概要を紹介し、ハードウェア特性、ソフトウェア ツール チェーン、代表的なアプリケーション例を説明します。

特に、モデル開発を効率化する TI Edge AI Studio と、高度なモデル エンジニアリング向けの TI Tiny ML Tensorlab という、最適化された 2 つのワークフローを紹介します。

プラットフォームの性能を実証するために、2 種類のアプリケーションを評価しました。数字認識設計では、約 99% の分類精度を達成し、ハードウェア NPU は CPU ベースの実装と比べて、推論レイテンシを 15 分の 1 に低減する一方で、エネルギー効率を 96% 向上させています。一方、波形分類では 100% の分類精度を達成し、NPU は推論レイテンシを 8 分の 1 に低減する一方で、エネルギー効率を 88% 向上させています。

これらの結果から、TinyEngine NPU を搭載した MSPM0G5187 は、リソース制約のある組込み機器へ AI を効率的に導入するための有力なプラットフォームであることが確認されました。

9 参考資料

1. テキサス・インスツルメンツ、『128kB フラッシュ、32kB SRAM、USB 2.0 FS、I2S、ADC、TinyEngine™ NPU 搭載、80MHz Arm® Cortex®-M0+ マイコン』、製品ページ
2. テキサス インスツルメンツ、MSPM0 G シリーズ 80MHz マイコン、テクニカル リファレンス マニュアル。
3. テキサス・インスツルメンツ、『MSPM0G5187 LaunchPad™ 開発キット評価基板』、製品ページ。
4. テキサス インスツルメンツ、『エッジ AI アクセラレーションを採用した Arm® Cortex®-M0+ マイコンはどのようにエレクトロニクスの脳力を向上させるか』、技術記事
5. テキサス インスツルメンツ、『TI の TinyEngine™ NPU により、多くの組込みシステムでエッジ AI アクセラレーションを実現』、製品概要
6. テキサス・インスツルメンツ、『エッジ AI センサ BoosterPack™ リファレンスデザイン』、製品ページ。
7. テキサス・インスツルメンツ、『TI のマイコン (MCU) による数字認識』、製品ページ
8. テキサス・インスツルメンツ、『エッジ AI 向け CCStudio™ ソフトウェア開発ツール』、製品ページ。
9. テキサス・インスツルメンツ、『Tiny ML Tensorlab GitHub リポジトリ』、Web ページ
10. テキサス・インスツルメンツ、『Tiny ML Tensorlab ユーザー ガイド』、Web ページ
11. テキサス・インスツルメンツ、『マイコン用 TI ニューラル ネットワーク コンパイラ ユーザー ガイド』、Web ページ
12. テキサス インスツルメンツ、『MSPM0 エッジ AI ユーザー ガイド』、Resource Explorer
13. テキサス インスツルメンツ、『MSPM0 低コスト マイコンでのエッジ AI を使用した AC アーク フォルト検出』、アプリケーション ノート
14. テキサス・インスツルメンツ、『Tiny ML ワークフロー サンプル Python スクリプトパッケージ』、E2E™ サポート スレッド。
15. PyTorch、「What is torch.nn really?」、Web ページ

10 改訂履歴

資料番号末尾の英字は改訂を表しています。その改訂履歴は英語版に準じています。

Changes from Revision * (June 2026) to Revision A (October 2026)	Page
• 「はじめに」セクションを更新しました	3
• 「エッジ AI ツールチェーン」セクションを更新	5
• 「エッジ AI ツールチェーン」セクションに 図 3-1 を追加	5
• 「TI Edge AI Studio」セクションを更新	6
• 「TI Tiny ML Tensorlab」セクションを更新	7
• 「TI ニューラル ネットワーク コンパイラ」セクションを更新	7
• 「TI Edge AI Studio による迅速なプロトタイピング」セクションを追加	9
• 「プロジェクトの設定」セクションを追加	9
• 「データ キャプチャ」セクションを追加	11
• 「ハードウェアの設定」セクションを追加	11

Copyright © 2026 Texas Instruments Incorporated

- 「機能の概要」セクションを追加 12
- 「データのインポート」セクションを追加 13
- 「分類タスク」セクションを追加 14
- 「回帰タスク」セクションを追加 14
- 「予測タスク」セクションを追加 15
- 「アノテーション リスト」セクションを追加 16
- 「データ アノテーション」セクションを追加 16
- 「モデルのトレーニング」セクションを追加 17
- 「トレーニング パラメータ」セクションを追加 19
- 「ツールチェーンの依存関係」セクションを追加 20
- 「パフォーマンス レポート」セクションを追加 21
- 「モデルのエクスポート」セクションを追加 24
- 「TI Tiny ML Tensorlab による高度な開発」セクションを追加 26
- 「環境設定」セクションを追加 26
- 「データ アクイジション」セクションを追加 27
- 「データの取得」セクションを追加 27
- 「データの検証」セクションを追加 27
- 「モデル アーキテクチャの反復リファクタリング」セクションを追加 29
- 「ベースライン実装」セクションを追加 29
- 「PyTorch 組み込みモジュールによるリファクタリング」セクションを追加 30
- 「データパイプラインの最適化」セクションを追加 31
- 「CNN アーキテクチャへの移行」セクションを追加 32
- 「モデルのエクスポート」セクションを追加 33
- 「量子化対応トレーニング」セクションを追加 33
- 「モデルのコンパイル」セクションを追加 34
- 「エッジ AI アプリケーション：数字認識」セクションを更新 37
- 「LeNet–5 バリアント CNN モデル」セクションを更新 37
- 「NPU/CPU 性能比較」セクションと 表 6-3 を更新 38
- 「エッジ AI アプリケーション：波形分類」セクションに注を追加 39
- 「特徴抽出」セクションの 表 7-4 のタイトルを更新 40
- 「時系列分類モデル」セクションを更新 42
- 「モデル メモリに関する考慮事項」セクション文章と 表 7-5 を更新 43
- 「NPU/CPU 性能比較」セクションの文章と 表 6-6 を更新 44
- 「まとめ」セクションを更新 45
- *「参照資料」* セクションを更新 45

Copyright © 2026 Texas Instruments Incorporated

重要なお知らせと免責事項

TI は、技術データと信頼性データ (データシートを含みます)、設計リソース (リファレンス デザインを含みます)、アプリケーションや設計に関する各種アドバイス、Web ツール、安全性情報、その他のリソースを、欠陥が存在する可能性のある「現状のまま」提供しており、商品性および特定目的に対する適合性の黙示保証、第三者の知的財産権の非侵害保証を含むいかなる保証も、明示的または黙示的にかかわらず拒否します。

これらのリソースは、TI 製品を使用する設計の経験を積んだ開発者への提供を意図したものです。(1) お客様のアプリケーションに適した TI 製品の選定、(2) お客様のアプリケーションの設計、検証、試験、(3) お客様のアプリケーションに該当する各種規格や、その他のあらゆる安全性、セキュリティ、規制、または他の要件への確実な適合に関する責任を、お客様のみが単独で負うものとします。

上記の各種リソースは、予告なく変更される可能性があります。これらのリソースは、リソースで説明されている TI 製品を使用するアプリケーションの開発の目的でのみ、TI はその使用をお客様に許諾します。これらのリソースに関して、他の目的で複製することや掲載することは禁止されています。TI や第三者の知的財産権のライセンスが付与されている訳ではありません。お客様は、これらのリソースを自身で使用した結果発生するあらゆる申し立て、損害、費用、損失、責任について、TI およびその代理人を完全に補償するものとし、TI は一切の責任を拒否します。

TI の製品は、 TI の販売条件 、 TI の総合的な品質ガイドライン 、 ti.com または TI 製品などに関連して提供される他の適用条件に従い提供されます。TI がこれらのリソースを提供することは、適用される TI の保証または他の保証の放棄の拡大や変更を意味するものではありません。 TI がカスタム、またはカスタマー仕様として明示的に指定していない限り、TI の製品は標準的なカタログに掲載される汎用機器です。

お客様がいかなる追加条項または代替条項を提案する場合も、TI はそれらに異議を唱え、拒否します。

Copyright © 2026, Texas Instruments Incorporated

最終更新日：2025 年 10 月