参考資料

Application Note

ニューラル ネットワーク処理ユニット (NPU) ガイド

Pranav Siddappa, Nima Eskandari, Nikhil Dasan, Tushar Sharma, Adithya Thonse

概要

このニューラル ネットワーク プロセッシング ユニット ガイドは、車載 / 産業用アプリケーション専用に設計された F28P55x NPU 上で機械学習ソリューションを導入するための包括的なフレームワークを提供します。このオンチップ ハードウェア アクセラレータを活用することで、C2000™Ware を使用するお客様はリアルタイム推論機能を実装し、予防保守、異常検出、センサ フュージョン、高度な制御システムなどを実現すると同時に、これらの領域で重要なディタミニスティック (確定的) な性能を維持することができます。このガイドではエンジニア向けに、実践的な正弦関数近似の例を紹介し、アーキテクチャ設計からハードウェア検証までのワークフロー全体を説明します。ここでは、メモリや計算上の制約によらない NPU の能力に注目します。この資料では、NPU の効果的な使用に不可欠な量子化手法、TI のツールチェーンを使用したコンパイル手順、CCS プロジェクト内の統合戦略について説明しています。自動車および産業分野のお客様は、特殊な環境下での厳しいタイミング、電力、信頼性要件を満たす効率的な組込み型 ML アプリケーションを開発するために必要な実践的知識を得ることができます。

目次

商標

C2000™ is a trademark of Texas Instruments.
PyTorch® is a registered trademark of Linux Foundation.
すべての商標は、それぞれの所有者に帰属します。

1 概要

1.1 NPU の定義と目的

F28P55x ニューラル プロセッシング ユニット (NPU) は、TI の C2000 マイコン アーキテクチャに統合された専用ハードウェア アクセラレータであり、高効率でニューラル ネットワーク計算を実行するように特別に設計されています。この専用シリコンを使用すると、組込みデバイス上で機械学習が直接推論を実施できるため、外部プロセッサやクラウドの接続が不要になります。C2000 エコシステムの不可欠な要素である NPU は、メイン CPU、アナログ フロントエンド、制御ペリフェラルと連携して動作し、リアルタイム制御システムで洗練されたインテリジェンスを実現します。NPU は、リソースに制約のある環境でも先進的な機械学習機能を実現する、ディタミニスティックな性能が最重要視されている TI の取り組みを表しています。

1.2 主な機能

F28P55x NPU は、C2000 アプリケーションを強化する以下の重要な機能を提供します。

- *アクセラレーションされたニューラル ネットワーク推論:*ハードウェアに最適化されたニューラル ネットワーク演算の実行により、メイン CPU でのソフトウェア実装を大幅に上回る性能を実現しますが、モデルのアーキテクチャによっては最大 70 倍の標準的なアクセラレーション係数があります。
- *整数ベース計算:*効率的な固定小数点数演算に特化しており、組込み上の制約に合わせて最適化済みで電力効率の優れた推論処理を実現できます。
- *リアルタイム処理:*ディタミニスティック (確定的) な実行により、車載および産業用アプリケーションの制御システムに不可欠な、予測可能なタイミング要件を維持します。
- *ペリフェラルの統合:*ADC、DAC、他の C2000 ペリフェラルとシームレスに動作することで、包括的な信号処理と制御ワークフローを実現できます。
- *並列動作:*メイン CPU が他のタスクを処理している間にニューラル ネットワーク計算を実行できるため、システム スループットを最大化できます。
- *車載 / 産業分野:*これらの要求の厳しい領域で必要とされる信頼性、温度範囲、および長期的な可用性に関する厳しい要件を満たすように設計されています。

1.3 技術的な制約

F28P55x NPU は強力ですが、アプリケーション設計に影響を及ぼすいくつかの制約下で動作します。

- *アーキテクチャ上の制約:*ReLu アクティベーションを備えた CNN や MLP などのニューラル ネットワーク トポロジは、LSTM やトランスフォーマーなどの複雑なアーキテクチャに比べてサポートが優れています。
- *高精度のトレードオフ:*NPU の実行に必要な量子化は、浮動小数点実装に比べて精度の低下を招くため、精度を維持するために慎重なトレーニング アプローチが必要になります。
- *開発ワークフローの複雑さ:*モデルのコンパイルおよび導入に固有のツールチェーン要件があるため、標準的なマイコンのプログラミングに比べて開発手順が追加されます。

これらの機能と制限により、車載および産業用組込みシステムにおいて F28P55x NPU が採用している実用的なアプリケーション分野を形成しています。この分野では、実装を成功させるために、計算能力とリソースの制約のバランスを維持することが不可欠です。

2 開発フローの概要

F28P55x NPU アプリケーションの開発ワークフローは、機械学習モデルの開発と、組込みシステムの導入を橋渡しする構造化プロセスに従います。このセクションでは、開発サイクル全体の概要について説明します。開発サイクルの詳細については、以降の章で詳しく説明します。

2.1 モデル開発フェーズ

NPU 開発の過程は、組込み用途向けに特別に最適化されたモデル設計とトレーニングから始まります。

- *モデル アーキテクチャの設計:*アプリケーションの要件と NPU ハードウェアの制約との間でバランスを維持するニューラル ネットワーク アーキテクチャを作成します。通常は、最適化された層タイプを用いたより小規模なネットワークを優先します。
- *データセットの準備:*開発で予測される動作範囲全体を網羅する代表的なトレーニング データを作成します。特に、量子化と互換性のある入力正規化手法に重点を置いてください。
- *量子化対応トレーニング:*トレーニング プロセス中に量子化効果を組み込むトレーニング手順を実装し、モデルが NPU の整数のみの制約の下でうまく機能するパラメータを学習できるようにします。
- *モデルの検証:*ターゲット アプリケーションに関連する指標を使用してモデルのパフォーマンスを検証し、量子化環境で精度と計算効率の両方を評価します。

2.2 モデル コンパイル フェーズ

トレーニングが完了した後、モデルを F28P55x NPU と互換性のある形式に変換する必要があります。

- *ONNX エクスポート:*トレーニング済みモデルを Open Neural Network Exchange (ONNX) 形式に変換します。この形式は、コンパイル ツールチェーンの交換標準として機能します。
- *TI NPU コンパイラの構成:*ターゲット デバイス、最適化手法、I/O 要件を指定する構成ファイルを使用してコンパイル パラメータを定義します。
- *コンパイルの実行:*TI のニューラル ネットワーク コンパイラ (TI MCU NNC) を使用して ONNX モデルを処理し、NPU ハードウェアと互換性のある C/C++ アーティファクトを生成します。
- *コンパイル出力検証:*生成されたヘッダ ファイルとライブラリを検証して、特に回帰タスクでの逆量子化のような重要な側面で、適切な変換を確実にします。

2.3 アプリケーション統合フェーズ

最後の段階では、コンパイル済みのモデルを完全な組込みアプリケーションに統合します。

- *CCS プロジェクトの設定:*生成されたモデルのアーティファクトと、アプリケーション固有のコードを組み合わせる、Code Composer Studio プロジェクトを確立します。
- *ハードウェア ペリフェラルの構成:*ニューラル ネットワークからの入力と処理の出力を提供するために必要なペリフェラル (ADC、DAC、通信インターフェイス) を構成します。
- *アプリケーション ロジックの実装:*入力と出力の適切なスケーリングなど、ペリフェラルと NPU 間のデータ フローを調整するメイン アプリケーション ロジックを開発します。
- *ハードウェア テスト:*実際の F28P55x ハードウェアにおいて、現実的な動作条件下でエンド ツー エンド機能を検証します。
- *パフォーマンスの最適化:*アプリケーションを微調整して、推論速度、精度、消費電力のバランスを最適化します。
- *展開パッケージ:*車載または産業用環境での量産導入に対応する、最終的なファームウェア パッケージを準備します。

3 モデル作成の例 (Python)

このセクションでは、正弦関数近似モデルの作成による NPU 開発原理の実用化を示します。この例は、モデル設計から F28P55x NPU への展開までの包括的なワークフローを示すリファレンス実装として使用されます。

3.1 モデル選択の根拠

正弦関数は、以下の説得力のある理由から、デモンストレーションの例として意図的に選択されました。

- *数学的複雑性:*明らかに単純であるにもかかわらず、正弦関数は単純な線形近似を超えたニューラル ネットワーク モデリング機能を必要とする非線形の数学的関係を表しています。
- *出力範囲の制限:*出力が –1 ～ +1 に制限されている場合、正弦関数は量子化手法に適した明確に定義された範囲を示します。
- *視覚的な検証:*この例では、オシロスコープ波形の視覚化 (正弦関数に最適) を使用していますが、他のアプリケーションでは、その特定のタスクに適した異なる検証方法が必要となります。たとえば、分類には混同行列、異常検出にはヒートマップ、その他の回帰タスクには誤差分布プロットが用いられます。これらは自動車分野や産業分野で一般的な制御システム、信号処理、運動制御アプリケーションに現れます。
- *パイプラインの完全なデモ:*この正弦関数により、NPU を中央処理素子として使用した完全な A/D コンバータ信号処理チェーンをデモできます。

3.2 モデル アーキテクチャの設計

正弦関数近似モデルは、F28P55x NPU 用に意図的に制約されたアーキテクチャを備えた多層パーセプトロン (MLP) を実装しています。

```
SineApproximator(
  (regressor): Sequential (
    (0): Linear(in_features=1, out_features=64, bias=True)
    (1): ReLU(inplace=True)
    (2): Linear(in_features=64, out_features=64, bias=True)
    (3): ReLU(inplace=True)
    (4): Linear(in_features=64, out_features=1, bias=True)))
```

コード 1: Sine_64_Model の正弦関数近似モデルを基盤としたアーキテクチャ

このアーキテクチャには、特に NPU 展開向けに設計上のいくつかの重要な決定事項が組み込まれています。

- *入力層:*角度値を受け入れる単一のニューロン入力 (0 ～ 2π ラジアンにマッピング)。
- *隠れ層:*精度とリソース効率のバランスを考慮して選択された 2 つの隠れ層には、それぞれ 64 個のニューロンがありますが、1 層あたり 128 個までのニューロンを持つモデルは NPU に適合できます。
- *アクティベーション機能:*計算効率と量子化に優れた特性のために選択された ReLU 活性化。
- *出力層:*予測される正弦値 (–1 ～ +1 の範囲) を生成する単一ニューロン出力。

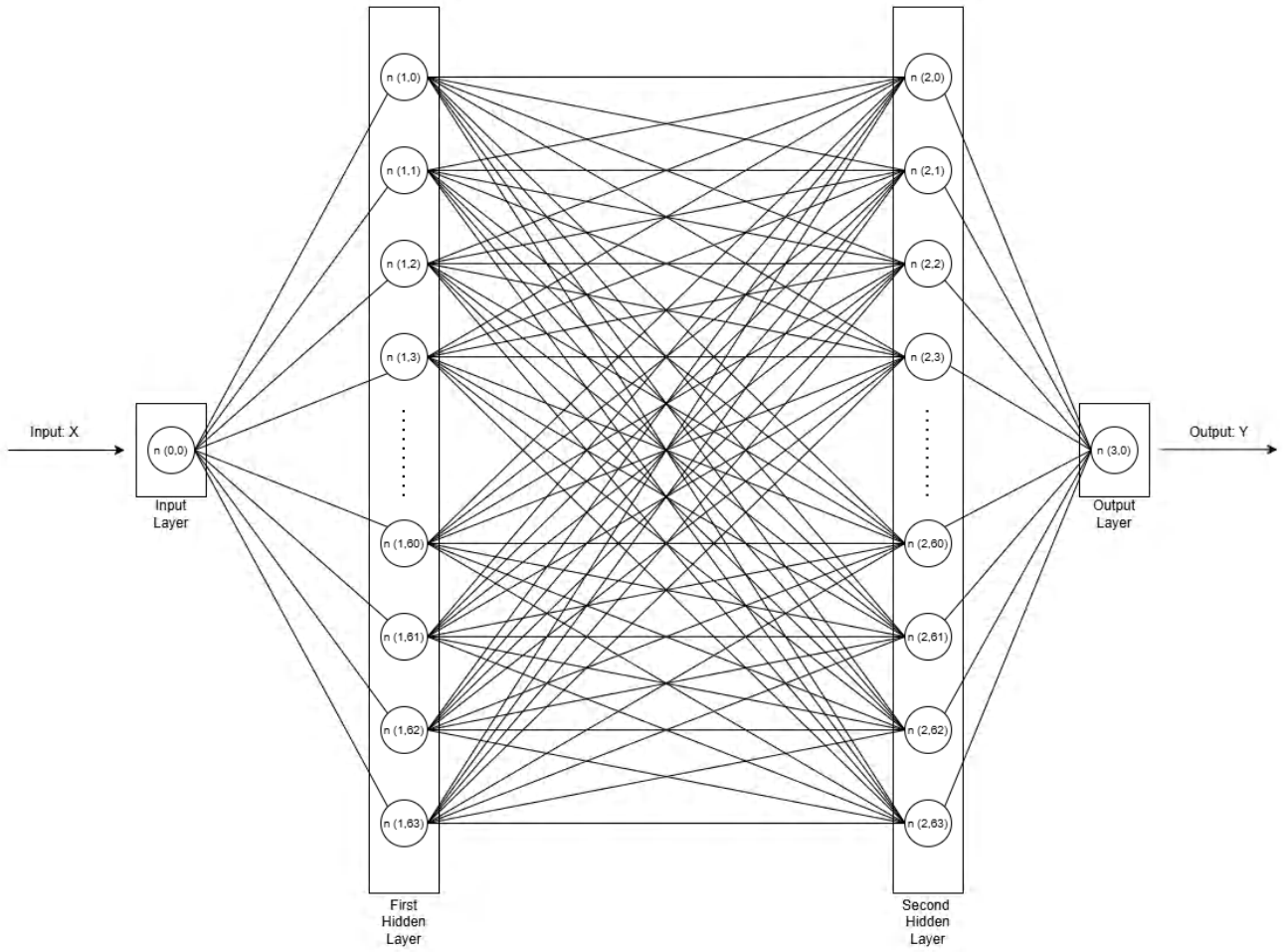


図 3-1. Sine_64_Model の正弦関数近似モデルを基盤としたアーキテクチャ

モデル アーキテクチャは、F28P55x NPU のメモリと計算能力に基づいて意図的にサイズ設定されています。重要な制約はパラメータの合計数であり、層ごとのニューロンに対する特定の制限ではありません。このアーキテクチャ (層サイズは 1×64、64×64、64×1) では、4,353 種類の合計パラメータが得られ、精度とリソース使用率のバランスがとれています。1 層あたり最大 128 個のニューロンを持つモデルはデバイスに適合できますが、精度とレイテンシを考慮して 64 個のニューロン構成を選択しました。特徴抽出機能を持たない大規模なネットワーク (層ごとに 512 個または 1024 個のニューロンを使用するネットワークなど) は、NPU の利用可能なメモリを超えます。エンジニアは、このプラットフォームのモデルを設計する際に、合計パラメータ バジェットに注目する必要があります。リソースに制約のあるこのモデル設計アプローチは、すべてのエッジ AI 実装で一般的です。メモリ、処理能力、エネルギーのバジェットが限られている場合、利用可能なハードウェアの制約内で可能な限り最良の性能を達成するために、ニューラル ネットワーク アーキテクチャを注意深く最適化する必要があります。

3.3 トレーニングの詳細

モデル トレーニングは、以下の構造化されたアプローチに従います。

3.3.1 開発環境の設定

- *モデル設定の例:*
  - ニューラル ネットワークの設計とトレーニングのための PyTorch® フレームワーク。
  - 量子化を意識したトレーニングのための TINPUTinyMLQATFxModule ラッパー (量子化の詳細についてはセクション 4 を参照)。
  - 数値計算と表示のためのサポート ライブラリ。
- *汎用ユーザー設定:*

  - お客様のターゲット ハードウェアに適した、TI のモデル最適化ツールをインストールしてください。
  - 選択したフレームワーク バージョンと TI のコンパイル ツールチェーンとの間で互換性を維持します。
  - 依存関係の競合を回避するために、仮想環境 (conda、venv) を使用して一貫性のある環境をセットアップします。
  - 開発中アプリケーション ドメインに固有の視覚化ライブラリを含めます (センサ向けの信号プロット、コンピュータ ビジョン向けの画像表示など)。

3.3.2 データセットの生成

- *モデル データセットの詳細例:*
  - 0 ～ 2π ラジアンの間で、100,000 種類のランダムに生成された角度値を作成します。
  - トレーニング ターゲットとして対応する正弦値を計算します。
  - トレーニング / テストは 80% のトレーニング データと 20% の検証データで分割されます。
- *一般的なユーザー アプローチ:*
  - 堅牢なモデル性能を維持するために、入力ドメイン全体にまたがるデータを生成または収集します。
  - センサ アプリケーションの場合、あらゆる動作条件と環境変数にわたってデータをキャプチャします。
  - データセットのバランスを取り、トレーニング済みモデル (特に分類タスク) のバイアスを回避します。
  - 展開シナリオの予想される入力範囲に基づいて適切な正規化を適用します。
  - 分類タスクには階層化サンプリングを使用して、train/test split でのクラス分布を維持します。
  - 実際の導入のためにモデルの堅牢性を向上させるために、トレーニング データに制御されたノイズの追加を検討してください。
  - 時系列データの場合は、データセットの準備中に適切なシーケンス処理を維持してください。

3.3.3 モデル トレーニングの構成

- *モデルの学習パラメータ例:*
  - *バッチ サイズ:*512 個のサンプル
  - *学習率:*1e–5 (安定収束のため意図的に小さい)
  - *オプティマイザ:* Adam
  - *損失関数:*平均二乗誤差 (MSE)
  - *トレーニング エポック数:*160
- *一般的なユーザー トレーニング フロー:*
  - モデルの複雑さと使用可能なメモリに基づいてバッチ サイズを調整します (メモリに制約のある環境では小さいサイズ)。
  - モデル サイズと収束動作に基づいて、適切な学習率を選択します。
  - *タスク要件に基づいてオプティマイザを選択します。*
    - 汎用トレーニングと高速収束に最適な Adam
    - 場合によってはより優れた一般化が可能な慣性項を付与した SGD
    - 回帰型ニューラル ネットワークに最適な RMSprop
  - *タスクに適した損失関数を選択します。*
    - 回帰問題に適した MSE
    - 分類タスク用の交差エントロピー
    - 特殊なアプリケーション向けのカスタム損失関数
  - 学習率スケジューリングを実装し、トレーニングの安定性と最終モデルの精度を向上させます。
  - 特に、トレーニング データが限られている場合は、過学習を防ぐために、早期停止を検討してください。
  - トレーニングと検証の両方の指標を監視して、適切な一般化を検証します。

3.3.4 量子化対応トレーニング プロセス

- *モデル量子化プロセスの例:*
  - 量子化効果をシミュレートするために TINPUTinyMLQATFxModule でラップされた初期モデル。
  - フォワード パスには、重みおよび活性化値量子化シミュレーションが含まれます。
  - 勾配は、誤差逆伝播中の量子化効果を示します。

  - 定期的な検証で、量子化モデルのパフォーマンスを監視します。

```
model = TINPUTinyMLQATFxModule(
      model,
      total_epochs=(int)(MAX_EPOCH/10),
      output_int=False,
      quantization_weight_bitwidth=2)
```

コード 2: 微調整用の量子化ラッパー付きラップ モデル

- *一般ユーザー向け量子化フロー:*
  - *量子化アプローチの選択:*
    - QAT による精度向上 (トレーニング時間の延長)
    - PTQ による開発期間の短縮 (潜在的な精度とのトレードオフ)
  - *ターゲット ハードウェア用の構成:*
    - ハードウェア固有のラッパーを使用します (TINPUTinyMLQATFxModule など)
    - 適切なビット幅 (2/4/8 ビット) と精密パラメータを設定します
    - 微調整のエポック数を設定します
  - *トレーニング プロセスの最適化:*
    - 最初に浮動小数点を使用してトレーニングし、その後、量子化を使用して微調整します
    - 浮動小数点ベースラインと比較したベンチマークで、精度への影響を評価します
  - *量子化の問題軽減:*
    - クリッピングを防ぐために、アクティベーション範囲を監視します
    - 回帰出力の適切な逆量子化を維持します
    - 重み / 活性化ヒストグラムを使用して、分布の問題を特定します

4 組込みプラットフォーム向けの量子化

ニューラル ネットワークの量子化は、重みと活性化の高精度浮動小数点表現を低精度の形式 (通常は整数) に変換するプロセスです。F28P55x NPU の場合、ハードウェアは整数ベースの計算専用に設計されているため、この変換は単に最適化にとどまらず、基本的な要件です。

正弦関数近似法の文脈では、量子化は連続的な数学的関係を、正弦波の本質的な特性を維持しながら NPU が効率的に処理できる形式に変換します。

4.1 量子化のアプローチ：QAT と PTQ の比較

ニューラル ネットワークの量子化には、次の 2 つの基本的なアプローチが存在します。トレーニング後の量子化 (PTQ) と量子化対応トレーニング (QAT) です。F28P55x NPU の展開に適した手法を選択するには、これらのアプローチの違いを理解することが重要です。

4.1.1 トレーニング後の量子化 (PTQ)

PTQ は、浮動小数点精度を使用して既に訓練されているモデルに量子化を適用します。このアプローチにより単純化できますが、特に小さなモデルや高精度なタスクでは、精度を犠牲にする可能性があります。

主な特性：

- *プロセス:*モデルを通常学習 → 量子化パラメータを較正 → 量子化形式に変換
- *キャリブレーション:*代表的なデータセットを使用して倍率を決定します。
- *開発期間:*開発サイクルの短縮 (再トレーニング不要)
- *精度への影響:*通常、QAT よりも精度損失が大きくなります。

利点：

- 既存のトレーニング済みモデルによるさらにシンプルなワークフロー。
- トレーニング手順の変更が不要。
- 迅速な導入パス。
- 開発の計算要件の低減。

制約：

- 精度の大幅な低下を招く可能性があります。
- 量子化効果に対する制御が限定的です。
- 正弦関数のような回帰タスクでは特に困難です。
- 量子化アーチファクトを補償する機能が制限されています。

4.1.2 量子化対応トレーニング (QAT)

QAT はトレーニング プロセス中に量子化効果を組み込んでおり、ネットワークは量子化された条件下で最適なパラメータを学習することができます。このアプローチは、一般的に精度を維持しますが、より多くの開発労力を必要とします。

主な特性：

- *プロセス:*初期トレーニング → 量子化演算を挿入する → トレーニングを続ける → 量子化された形式に変換する。
- *シミュレーション:*順方向パスと逆方向パスの両方で量子化効果をシミュレートします。
- *開発期間:*開発サイクルの延長 (追加のトレーニングが必要)。

利点：

- モデル精度の維持。
- ネットワークは量子化効果を補償することを学習します。
- 高精度を要するアプリケーションに特に有用です。
- 量子化されたハードウェアで、より予測可能なパフォーマンスを実現します。

制約：

- 複雑な開発ワークフロー。
- 追加のトレーニング計算が必要。

- 開発期間の延長。
- 慎重なハイパーパラメータ調整が必要。

TI の正弦関数近似法において、入力領域全体で滑らかな正弦波特性を維持するために不可欠な回帰タスクの精度維持のために QAT を選択しました。

4.2 量子化フレームワークおよびラッパー モジュール

TI のツールチェーンは、さまざまな導入ターゲットに対応するさまざまな量子化アプローチを封止した、特化型のラッパー モジュールを提供しています。これらのラッパーを理解することは、アプリケーションに適した量子化戦略を選択するために不可欠です。

TI の量子化フレームワークは、次の 2 つの重要な次元を組み合わせた、4 つの異なるラッパー モジュールを提供しています。

- *ターゲット ハードウェア:*
  - *汎用:*CPU 実行用に最適化 (標準整数演算)
  - *TINPU:*特に TI のニューラル プロセッシング ユニットハードウェア向けに最適化済み
- *量子化アプローチ:*
  - *QAT (量子化対応トレーニング):*トレーニング中に量子化効果を組み込みます
  - *PTQ (トレーニング後の量子化):*トレーニングの完了後に量子化を適用します

これにより、次の 4 つのラッパー オプションのマトリクスが作成されます。

表 4-1. 量子化ラッパー

ターゲット / アプローチ	QAT	PTQ
汎用 CPU	GenericTinyMLQATFxModule	GenericTinyMLPTQFxModule
TI NPU	TINPUTinyMLQATFxModule	TINPUTinyMLPTQFxModule

4.2.1 CPU 量子化のための汎用ラッパー

汎用ラッパー (GenericTinyMLQATFxModule および GenericTinyMLPTQFxModule) は、CPU ベースの実行を対象とし、PyTorch のネイティブ量子化 API を利用します。

```
from tinyml_torchmodelopt.quantization import GenericTinyMLQATFxModule
# or
from tinyml_torchmodelopt.quantization import GenericTinyMLPTQFxModule
```

コード 3: 一般的な量子化 API

これらのラッパーは、次の用途に適しています。

- NPU アクセラレーションを使用しない C2000/ARM マイコンへの導入。
- NPU 固有の最適化の前に量子化効果をテストする。
- 性能要件がそれほど厳しくないアプリケーション。
- モデルのプロトタイピングと初期検証。

汎用ラッパーは標準の PyTorch 量子化 API を使用していますが、既存のモデルへのコード変更を最小限に抑えた一貫したインターフェイスを通じてアプリケーションを簡素化します。

4.2.2 NPU ハードウェア アクセラレーション用 TINPU ラッパー

TINPU ラッパー (TINPUTinyMLQATFxModule および TINPUTinyMLPTQFxModule) は、特に、TI のニューラル プロセッシング ユニット ハードウェア アクセラレータを対象としています。

```
from tinyml_torchmodelopt.quantization import TINPUTinyMLQATFxModule
# or
from tinyml_torchmodelopt.quantization import TINPUTinyMLPTQFxModule
```

コード 4: TI NPU 固有の量子化 API

これらのラッパーは、次の場合に不可欠です。

- F28P55x や、NPU アクセラレーションを搭載した他の TI デバイスへの導入。
- TI のハードウェアの性能最大化。
- リアルタイム推論を必要とするアプリケーション。
- NPU コンパイル処理ツールとの互換性の維持。

TINPU ラッパーには、TI NPU ハードウェアに固有の制約が組み込まれているため、モデルは一般的に量子化の利点を得るだけでなく、NPU アーキテクチャでの実行向けに最適化されていることに変わりはありません。

5 モデルの検証

正弦関数近似モデルにおいて、F28P55x NPU ハードウェアで性能を最適化するように特別に設計された 2 段階検証プロセスを実施しました。このセクションでは、検証方法論、量子化対応トレーニング アプローチ、およびモデルの有効性を示す結果として得られる性能指標について詳しく説明します。

5.1 2 段階トレーニング手法

このトレーニング プロセスでは、量子化に制約のある NPU ハードウェアの性能を最大化するために、意図的な 2 段階アプローチを採用しました。

5.1.1 初期トレーニング フェーズ

- 通常の浮動小数点重みでトレーニングされたモデルです。
- 基礎となる数学的正弦関数の学習に焦点を当てます。
- 基本的なパターン認識能力を確立します。
- 量子化の制約が課される前に、強力な土台を構築します。

5.1.2 微調整フェーズ

- F28P55x 固有の量子化重み付けを使用してトレーニングを受けたモデル。
- NPU の実際の整数のみの演算をシミュレート。
- 量子化された実行に特化した重みの最適化。
- *Quant_epoch* は通常、*max(5,float_epoch/10)* に設定されます。

この戦略的アプローチは、以下の両方の分野における長所を活用します。最高の浮動小数点精度を使用した初期トレーニングにより、堅牢な特徴抽出を確立しつつ、量子化された重みを使用してターゲットを絞った微調整をすることで、特に導入ハードウェアのパフォーマンスを最適化します。

5.2 トレーニング フェーズの比較

表 5-1 では、通常の浮動小数点トレーニングの最後のエポックから、量子化を認識した微調整後の最終エポックまでの性能指標を比較します。

表 5-1. 量子化プロセス前後の指標の比較

メトリック	160 エポック後の浮動小数点数を用いたトレーニング指標	16 エポック後の QAT 微調整指標	変更 (%)
検証損失	0.00181	0.00105	42% 改善
検証用 MAE	0.02031	0.0215	6% 低下
検証用 R^2 スコア	0.9963	0.9989	0.26% 改善

標準的な浮動小数点モデルは、従来の環境では優れた性能を発揮しますが、NPU の整数のみの動作に制約されると、大幅な精度低下が発生します。QAT 手順では、学習プロセス全体の量子化効果をシミュレートし、ニューラル ネットワークがそれに応じてパラメータを適応させることを可能にします。この最適化により、F28P55x NPU に導入されたモデルは、ハードウェアに特化したニューラル ネットワーク アクセラレーション機能を活用しながら、計算の整合性を維持することを検証します。

検証損失 (-42.0%) と R^2 スコア (+0.26%) で観測された改善により、QAT は特定の性能指標を向上させると同時に、モデルを NPU 展開用に準備できることが実証されます。MAE のわずかな増加 (+5.9%) は、量子化プロセスにおける固有のトレードオフを示しています。これらの結果は、リソースに制約のある組込みハードウェアで最適なパフォーマンスを実現するための量子化対応トレーニング手法の有効性を実証しています。

5.3 検証結果と指標

検証プロセスでは、目的の性能評価を維持するためにシンプルな方法論が採用されました。

- *検証データセット:*生成されたデータの 20% がトレーニングから保持されています。
- *複数の指標:*補完的な評価指標を使用した包括的な評価。
- *量子化シミュレーション:*検証は、ターゲット ハードウェアと同じ量子化方式を使用して実行されます。

量子化対応トレーニングを受けた Sine_64 モデルは、単純な ML モデルで期待されるように、NPU 展開におけるその有効性を検証する優れた性能指標を達成しました。

- 検証損失は、平均二乗誤差 (MSE) によって測定されます。平均二乗誤差は、予測値 ($\hat{y}_i$) と実測値 (y_i) との二乗差の平均を測定します。値が小さいほど、モデル性能が向上します。

$$MSE = 1/n\sum_0^n (\mathrm{yi} - \hat{\mathrm{y}}\mathrm{i})^2 \quad (1)$$

- 平均絶対誤差は、予測値と実測値の平均絶対偏差を測定します。MSE とは異なり、これは誤差を二乗しないため、外れ値に対する感度が低くなります。

$$MAE = 1/n\sum_0^n |\mathrm{yi} - \hat{\mathrm{y}}\mathrm{i}| \quad (2)$$

- R^2 は、平均を予測因子として用いた場合と比較して、データの分散をモデルがどの程度説明しているかを示します。値の範囲は 0 ～ 1 で、1 は完全予測を表します。バリデーション R^2 スコアは、与えられたデータの分散の説明に関するモデルを示します。

$$\mathrm{R}^2 = 1 - \left(\sum_0^n (\mathrm{yi} - \hat{\mathrm{y}}\mathrm{i})^2\right) / \left(\sum_0^n (\mathrm{yi} - \bar{\mathrm{y}})^2\right) \quad (3)$$

変数:

- y_i = i 番目のサンプルにおける実際の (true) 正弦値
- $\hat{y}_i$ = i 番目のサンプルのニューラル ネットワークから予測された正弦値
- $\bar{y}$ = 検証データセット内における実際の正弦値の平均

表 5-2. トレーニング検証指標

メトリック	メトリック	重要性
検証損失	0.00038	予測精度が高いことを示す、非常に小さい平均二乗誤差
検証用 MAE	0.01305	正弦波の範囲全体で約 1.3% の平均絶対誤差 (–1 ～ +1)
検証用 R^2 スコア	0.9993	ほぼ完全な決定係数であり、このモデルが正弦値の変動の 99.93% を説明していることを示します。

これらの指標は、以下の重要な成果を示します。

- **高精度:** 量子化の制約があるにもかかわらず、このモデルは 1% 未満の誤差精度を達成しています。
- **一貫した性能:** R^2 スコアが大きい場合は、入力範囲全体で信頼性の高い予測を示しています。
- **量子化レジリエンス:** 整数のみの操作であっても、パフォーマンスの低下は最小限です。
- **導入準備:** 指標は、モデルが NPU 実装に適合していることを検証します。

6 モデルのテスト

トレーニングと検証が成功した後、実用的なアプリケーションで性能を検証するには、正弦関数近似モデルの包括的なテストが不可欠です。このセクションでは、モデルの NPU 展開への準備状況を評価するために使用されるテスト方法、推論手順、および結果評価手法について詳しく説明します。

6.1 推論のセットアップと方法論

TI の正弦関数テスト方法論では、ONNX Runtime を用いた専用の推論ノートブックを使用して、ハードウェアを導入する前にモデルを検証します。このフレームワークは、一貫したデータ処理を維持しながら、予測のためのクリーンなインターフェイスを提供するカプセル化されたテスト クラスを実装します。

テスト方法論は、以下の体系的なアプローチに従います。

- *モデルの読み込み:*ONNX モデルは ONNX ランタイムを使用してロードされ、最終的に NPU 用にコンパイルされるのと同じモデルへのアクセスが提供されます。
- *入力の準備:*テスト ポイントは、入力領域全体で 0 から 2π ラジアンまで生成されます。
- *リファレンス比較:*各予測は、実際の数学的正弦関数と比較されます。

このテスト アプローチでは、エクスポートされた ONNX モデル、特に逆量子化層を備えたバージョンが、範囲全体にわたって正弦関数を正しく近似することを検証することで、ハードウェアの実装に進む前の重要な品質ゲートとして機能します。

6.1.1 一般的なユーザー テスト アプローチ

- *モデルの検証:*適切なフレームワークを使用して、エクスポートしたモデルをテストします。
- *代表的なテスト データ:*導入条件とエッジ ケースを反映したテスト データセットを作成します。
- *ドメイン固有の指標:*アプリケーション要件に適した評価指標を選択します。
  - *分類:*精度、適合率、再現率、F1 スコア。
  - *回帰:*MAE、MSE、R^2 スコア。
  - *信号処理:*信号対雑音比、相互相関、周波数応答。
  - *ビジョン:*物体検出の IoU、mAP、画質の SSIM。
- *性能ベンチマーク測定:*推論速度、メモリ使用量、消費電力を測定します。
- *比較ベースライン:*該当する場合、参照アルゴリズムに対するベンチマークを測定します。
- *環境テスト:*重要なアプリケーションの場合、温度範囲、電圧変動、またはその他の環境要因に関する試験を実施します。

この検証は、ハードウェア実装に進む前に重要な品質ゲートとして機能し、ハードウェア導入に投資する前に、量子化モデルがアプリケーション要件を満たしていることを維持します。

6.2 テスト結果とビジュアル分析

6.2.1 視覚的なパフォーマンス評価

テスト プロセスの重要な要素は、予測された正弦値と実際の正弦値を視覚的に比較することです。このノートブックでは、入力ドメイン全体で真の正弦関数に対してモデルの予測を重ね合わせる包括的なプロットを生成します。

この可視化により、モデルの近似の品質がすぐに明らかになります。正弦関数の例では、予測された曲線は実際の正弦曲線にほぼ追従し、目に見える偏差は最小限です。

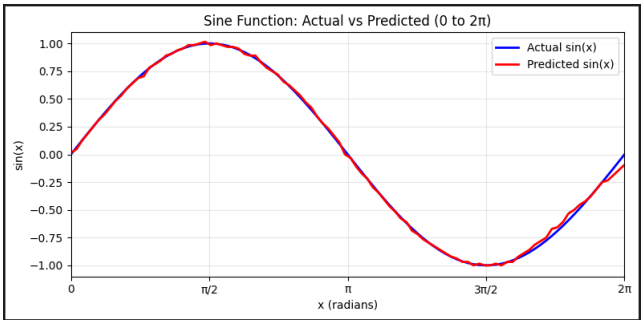


図 6-1. Python におけるニューラル ネットワーク正弦関数近似性能

図 6-1 に示すように、このグラフは、入力ドメイン全体で正弦関数に近似する際のトレーニング済みニューラル ネットワークの性能を示しています。青い線は真の数学的正弦関数を表し、赤い線は ONNX ファイルとして実行したときに量子化を意識したニューラル ネットワーク モデルによって生成される予測を示しています。このグラフは、入力ドメイン全体で正弦関数に近似する際のトレーニング済みニューラル ネットワークの性能を示しています。

6.3 定量的な性能指標

目視検査だけでなく、テスト フレームワークによって包括的な誤差指標を計算し、予測精度を正確に定量化します。モデルの性能は、テスト中に得られた次の誤差指標によって正確に定量化できます。

- *平均絶対誤差 (MAE) :*0.013827
- *最大誤差:*0.093750
- *最小誤差:*0.000126

これらの定量的結果は、グラフによる視覚的評価を補完し、定量化されたトレーニングの制約にもかかわらず、モデルの優れた近似能力を数値で確認することができます。視覚的および数値的検証の組み合わせにより、ハードウェア実装前にモデルの性能に対する包括的な信頼性が得られます。

7 TI マイコン (C2000 – F28P55x) へのモデル移動 [初心者レベル]

F28P55x への ONNX モデルの導入は、さまざまなユーザー ニーズや専門知識レベルに合わせてカスタマイズされた 2 つの異なる経路によって実現できます。

合理化された効率的な設計をお求めの場合、初心者向けのアプローチにより、最小限の構成要件で簡素化されたワークフローを提供します。この方法は、多くの技術的な複雑さを抽象化しながら、機能的な実装を可能にします。

ユーザーは、『*TI Neural Network Compiler for MCUs User's Guide - 2.1.0.LTS*』のセクション 1 とセクション 2 に従うことができます。これらのセクションでは、展開に必要なコンパイル済みライブラリとヘッダ ファイルを取得するためのステップバイステップの手順を紹介しています。

8 TI マイコン (C2000 – F28P55x) へのモデル移動 [開発者レベル]

この開発者向けアプローチは、コンパイル プロセスを包括的に制御し、既存システムとのより深い統合を必要とするエンジニアに対応しています。この方法はさらに複雑になりますが、デプロイメント パイプラインの各ステップを完全に可視化し、コンパイル パラメータを広範にカスタマイズできます。次のセクションでは、このワークフローについて詳しく説明します。開発者は、特定のアプリケーション要件に合わせてニューラル ネットワークの実装を最適化できるようになります。

このセクションでは、数学モデルを Code Composer Studio プロジェクトに統合できるハードウェア互換ファイルに変換するための完全なワークフローについて詳しく説明します。

8.1 コンパイルの前提条件

モデル コンパイル プロセスを開始する前に、いくつかの特殊なツールと環境を適切に設定する必要があります。この準備フェーズは、コンパイルと導入を成功させるために重要です。

8.1.1 必要な TI ソフトウェア コンポーネント

コンパイル ツールチェーンは、TI 固有のリポジトリとツールを複数使用しています。

- *tinyml-modelmaker*
  - *目的:*モデル開発とコンパイルのための一貫したインターフェイスを提供します。
  - *機能:*コンパイル ワークフローをオーケストレーションし、他のコンポーネントと統合します。
- *tinyml- modeloptimization*
  - *目的:*モデル量子化および最適化ツールキット。
  - *機能:*TI-NPU 展開専用モデルを準備するためのラッパーとツールを提供します。
- *C2000Ware SDK*
  - *目的:*C2000 マイコン用のプラットフォーム固有のドライバ、ライブラリ、ツール。
  - *機能:*F28P55x プラットフォームの重要なドライバとハードウェア抽象化を提供します。
- tinyml-modelzoo
  - 目的:組込みシステム向けに最適化された、厳選されたニューラル ネットワーク モデルを提供します。
  - 機能:tinyml-modelmaker メソッドによるモデル コンパイルに必要です。

8.1.2 環境設定プロセス

コンパイル環境の設定は、以下の構造化されたアプローチに従います。

- *必要なリポジトリのクローン:*
  - git clone https://github.com/TexasInstruments/tinyml-tensorlab/tree/main/tinyml-modelmaker
  - git clone https://github.com/TexasInstruments/tinyml-tensorlab/tree/main/tinyml-modeloptimization
  - git clone https://github.com/TexasInstruments/tinyml-tensorlab/tree/main/tinyml-modelzoo
  - *初期化スクリプトの実行:*
    - *cd into* ***tinyml-modelmaker*** *repo*
    - *setup_all.sh* を実行し、環境設定全体をオーケストレーションします
    - *setup_c2000ware.sh* を実行し、C2000Ware SDK を構成します
    - *setup_cg_tools.sh* を実行し、TI のコード生成ツールをセットアップします
- *Python の環境設定:*
  - 依存関係の競合を回避するために、専用の Python 仮想環境を推奨します。
  - 必要な主要 Python パッケージは次のとおりです。
    - ONNX および ONNX ランタイム
    - TVM (TI MCU NNC) フレームワーク
    - NumPy および関連する数値ライブラリ

8.2 構成ファイルのセットアップ

コンパイル プロセスは、ONNX モデルを NPU 互換コードに変換するための重要なパラメータを定義する *config.yaml* ファイルによって制御されます。この構成アプローチは、コンパイル ワークフローの一貫性を維持しながら柔軟性をもたらします。

8.2.1 構成ファイル構造

正弦関数モデルをコンパイルするための一般的な config.yaml ファイルには、次の重要なセクションがあります。

```
common:  # The common section can be plainly copied as it is
    target_module: 'timeseries'
    task_type: 'generic_timeseries_regression'
    target_device: 'F28P55'

dataset:
    dataset_name: Sine # Can be anything, used for directory name
    enable: False      # Please note the 'False'. Since model is already trained.
training:
enable: False  # Please note the 'False'. Since model is already trained.     model_name:
'SineModel_1e-05_160_64_Dequant'   # Can be anything, used for
                                                    directory name
    output_int: False # We need the model to dequantize the values after
                              running on NPU and return float value
compilation:
    enable: True
    model_path: "path//to//SineModel_1e-05_160_64_Dequant.onnx"
```

コード 5: 構成ファイルの例

各構成セクションでは、コンパイル プロセスをガイドするための特定の目的が提供されます。

- *共通設定:*基本的なタスク タイプとターゲット ハードウェアを定義します。
  - *task_type* は、機械学習タスク カテゴリを示します。
  - *target_device* は、F28P55x 固有の最適化を起動します。
- *データセット設定:*データセット処理の動作を制御します。
  - 事前トレーニング済みモデルを使用する場合、通常は無効になります。
- *トレーニング設定:*トレーニング パラメータを設定します。
  - *output_int :*これを False に設定することは、回帰タスクにおいて適切な浮動小数点出力を検証するために非常に重要です。
  - 事前トレーニング済みモデルでは通常、トレーニングは無効になります。
- *コンパイル設定:*モデル変換を制御します。
  - *イネーブル:*True 設定により、コンパイル フェーズを有効にします。
  - *model_path* は、逆量子化レイヤを使用して ONNX モデルの位置を指定します。

8.2.1.1 逆量子化フラグを必要とするモデル

*output_int :*False 設定は、次の場合に不可欠です。

- *回帰モデル:*連続値を予測するすべてのモデルで、次のようなものがあります。
  - 時系列予測 (温度、圧力、電圧予測)
  - 制御システムのモデル化 (PID 係数の推定)
  - 関数近似 (正弦関数の例と同様)
  - 信号フィルタリング アプリケーション
  - センサ キャリブレーション モデル
- *正規化多用モデル:*出力スケーリングが重要なアプリケーション:
  - 特定の範囲に正規化された出力を持つモデル
  - キャリブレーション済み出力を使用するセンサ フュージョン アルゴリズム
  - 物理量の推定 (力、トルクなど)
- *複数出力モデル:*出力によっては浮動小数点精度が必要な場合:
  - 分類 / 回帰モデルの組み合わせ

- 姿勢推定 (角度には浮動小数点が必要)
- 座標回帰 (オブジェクトのローカリゼーション)

このフラグがないと、これらのモデルは連続値範囲や正確な分数出力を必要とするアプリケーションには適さない量子化された整数出力を生成します。

8.2.2 回帰モデルの特別な構成

正弦関数近似のような回帰タスクでは、整数のみの NPU 出力を浮動小数点値へ適切に逆量子化するために、特別な処理が必要です。

8.2.2.1 出力逆量子化フラグ

- *output_int*:回帰タスクでは、false 設定は必須です。
- これにより、コンパイラは生成されたコードで逆量子化操作を保持するように指示されます。
- このフラグがないと、モデルは正弦近似には不適切な整数出力を生成します。

8.2.2.2 コンパイラ定数の変更

構成ファイルに加えて、コンパイラは浮動小数点出力をサポートするために変更を加える必要があります。

- *tinyml-modelmaker/ai_modules/timeseries/constant.py* を検索して開きます。
- *COMPILATION_C28_HARD_TINPU* 辞書が見つかるまでファイルをスクロールします。

```
COMPILATION_C28_HARD_TINPU= dict(
      target= "c, ti-npu type=hard skip_normalize=true output_int=true",
      target_c_mcpu='c28',
      cross_compiler=CL2000_CROSS_COMPILER,
)
```

コード 6: 非回帰モデルの参照定数

- この既存の定数は、分類タスク用に設定されます。ここで、
  - *type = hard* はハードウェア NPU アクセラレーションを指定します (CPU に対して)
  - *skip_normalize=true* は、正規化 / スケーリングをバイパスします
  - *output_int=true* は整数出力を強制します
- 重要な変更を加えて既存の定数を複製することにより、*COMPILATION_C28_HARD_TINPU_TEMP* という名前の新しい定数を作成します。

```
COMPILATION_C28_HARD_TINPU_TEMP = dict(
      target= "c, ti-npu type=hard skip_normalize=false output_int=false",
      target_c_mcpu='c28',
      cross_compiler=CL2000_CROSS_COMPILER,
)
```

コード 7: 回帰モデルを処理するための追加定数

- 次のパラメータは、以下の回帰モデルに合わせて変更されます。
- *type=hard*:
  - 変更されません。これにより、ニューラル ネットワークの計算に CPU を使用する *soft* ではなく、NPU ハードウェア アクセラレータをターゲットにコンパイルするように指示されます。
- *skip_normalize=false*:
  - サンプル アプリケーションは、デモとして作成された単純なモデルなので、正規化は必要ありませんでした。
- *output_int=false*:
  - コンパイラは整数ではなく浮動小数点出力を生成します。
  - 回帰タスクに不可欠な小数精度を保持します。
  - 値の全連続範囲を表すことができます。
- *target_c_mcpu='c28'*:変更せずに保持 - C28x アーキテクチャ用のコードの生成を指定します。
- *cross_compiler=CL2000_CROSS_COMPILER*:変更されません。使用するコンパイラ ツールチェーンを定義します。

8.2.2.3 コンパイル辞書の更新

カスタム定数を使用するには、デフォルトのコンパイル辞書を変更する必要があります。

- 使用している定数を *COMPILATION_C28_HARD_TINPU* から *COMPILATION_C28_HARD_TINPU_TEMP* に変更します。

```
TASK_TYPE_GENERIC_TS_REGRESSION: {
      COMPILATION_DEFAULT: dict(
-  compilation=dict(**COMPILATION_C28_HARD_TINPU,
   cross_compiler_options=CROSS_COMPILER_OPTIONS_F28P55, )
+  compilation=dict(**COMPILATION_C28_HARD_TINPU_TEMP,
   cross_compiler_options=CROSS_COMPILER_OPTIONS_F28P55, )
 ),
```

コード 8: 回帰モデルに対応するために行われた変更

8.3 コンパイル プロセス フロー

ONNX モデルから F28P55x NPU 互換コードへの変換には、TI の マイコン向けニューラル ネットワーク コンパイラ (TI MCU NNC) によって実行される多段プロセスが含まれます。このセクションでは、コンパイルのステップバイステップのワークフローと内部変換について詳しく説明します。

8.3.1 コンパイル処理の開始

構成ファイルが適切にセットアップされると、TinyML モデルメーカーのスクリプトを使用してコンパイル処理が開始されます。

```
cd tinyml-modelzoo
/<path/to/run_tinyml_modelzoo.sh> <path/to/config_file>
```

コード 9: ONNX ファイルを組込み互換ファイルにコンパイルするコマンド

正弦関数の例では、コマンドは次のとおりです。

```
./run_tinyml_modelzoo.sh    ./config.yaml
```

コード 10: コンパイル処理をトリガするコマンド例

このコマンドは、ONNX モデルを F28P55x NPU 用に最適化された C/C++ コードに変換する完全なコンパイル パイプラインをトリガします。

8.3.2 コンパイル フェーズ

コンパイラは、モデルを 3 つの主要フェーズで処理します。

- *モデルの検証*
  - コンパイラは、ONNX モデルを NPU で実行できるかどうかをチェックします。
  - すべての操作がサポートされていることを確認します (この正弦モデルの線形レイヤーや ReLU など)。
  - モデルが NPU メモリ制限内に収まることを確認します。
- *モデル変換*
  - ONNX モデルを NPU 用に最適化されたフォーマットに変換します。
  - 整数ベースの計算に量子化パラメータを適用します。
  - TI の正弦近似ツールのような回帰モデルの場合、量子化の情報を保存します。
- *コード生成*
  - F28P55x で実行できる C/C++ コードを作成します
  - 2 つのメイン ファイルを生成します。
    - ヘッダ ファイル (tvmgen_default.h) と関数宣言
    - コンパイルされたモデルを含むライブラリ ファイル (mod.a)

8.3.3 注意すべき一般的な問題

コンパイラはほとんどの複雑さを処理できますが、次の一般的なメッセージに注意してください。

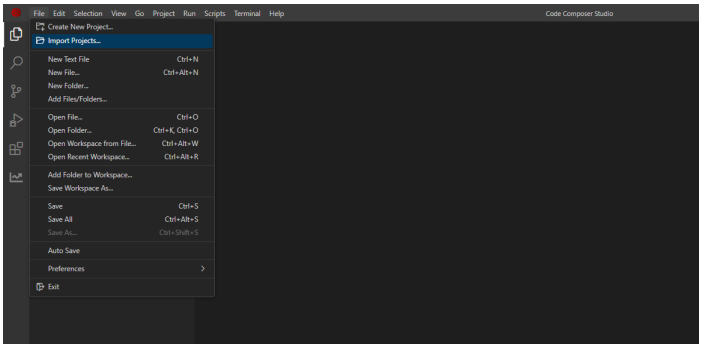
- *Unsupported operation (サポートされていない操作)*:モデルには、NPU が実行できない操作が含まれています。
- *Memory constraint exceeded (メモリ制約を超過しました)*:モデルがエッジ デバイスに対して大きすぎます。
- *Dequantization error (逆量子化エラー)*:回帰モデルでは、出力の逆量子化に注意が必要な場合があります。

9 マイコン プロジェクトのセットアップ

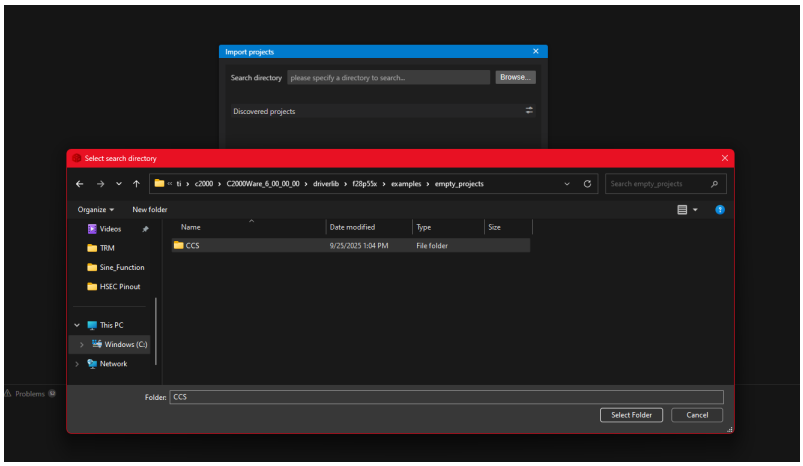
ニューラル ネットワーク モデルを NPU 互換ファイルへ正常にコンパイルした後、次の重要なフェーズは、これらのコンポーネントを Code Composer Studio (CCS) プロジェクトに統合して F28P55x マイコンで実行する必要が生じます。このセクションでは、プロジェクト構造の確立、開発環境の構成、NPU の機能を活用するために必要なアプリケーション フレームワークの実装についてエンジニアをガイドします。

9.1 NPU アプリケーション向け CCS プロジェクトの作成

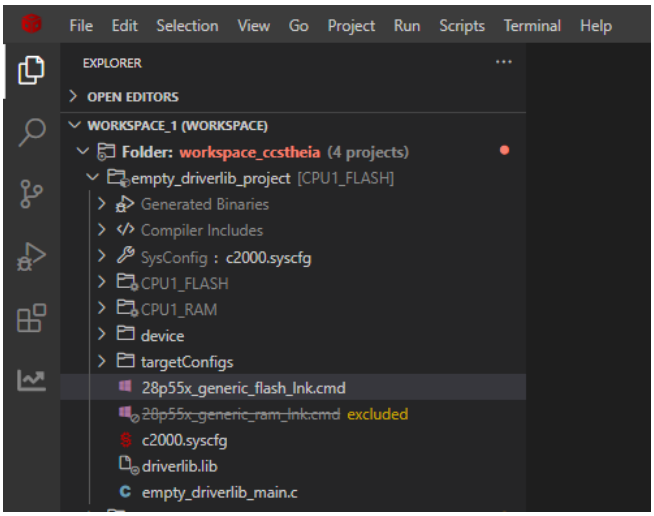
- *ステップ 1 :*CCS を開き、*File* と *Import Projects* をクリックします。



- **ステップ 2 :**C2000Ware SDK から *empty_project* をインポートします。



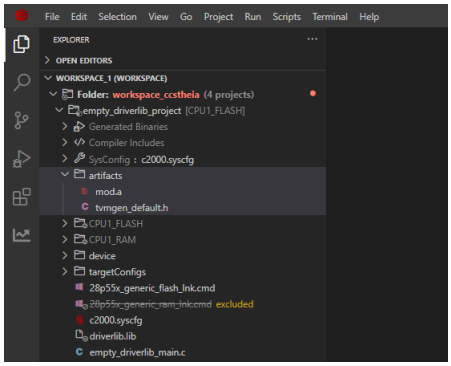
- *ステップ 3 :*インポートしたプロジェクトの *f28p55x_generic_flash_lnk.cmd* ファイルを開きます



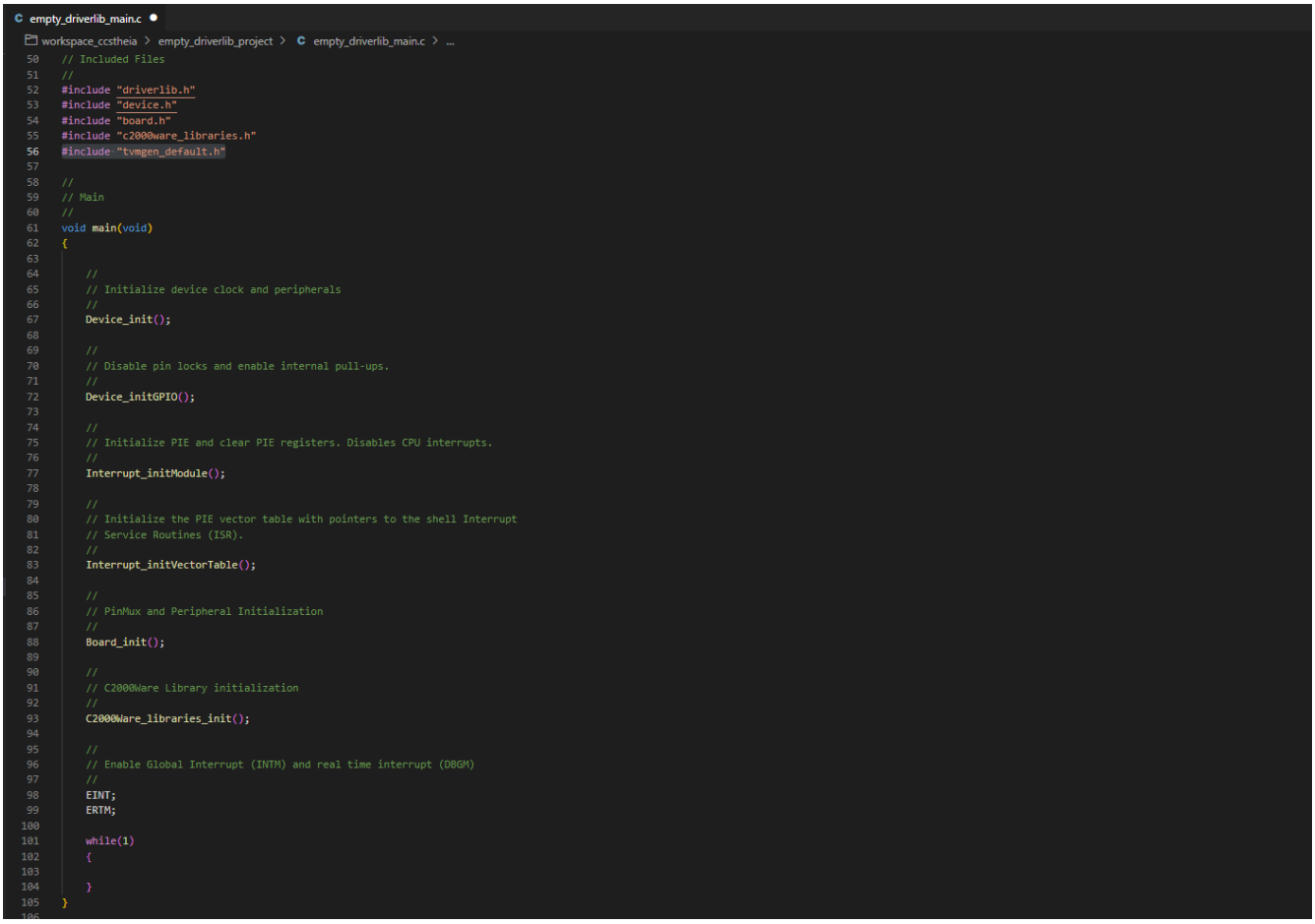
- *ステップ 4:*FLASH に .rodata.tvm セクションを追加し、ハードウェア NPU がアクセスできるようにグローバル共有 SRAM に .bss.noinit.tvm セクションを配置します。たとえば、F28P55x マイコン デバイスの RAMGS0、RAMGS1、RAMGS2、RAMGS3 などです。

```
SECTIONS
{
   codestart        : > BEGIN
   .text            : >> FLASH_BANK0 | FLASH_BANK1, ALIGN(8)
   .cinit           : > FLASH_BANK0, ALIGN(8)
   .switch          : > FLASH_BANK0, ALIGN(8)
   .reset           : > RESET, TYPE = DSECT /* not used, */

   .stack           : > RAMLS3
#if defined(__TI_EABI__)
   .bss             : >> RAMLS47 | RAMLS89 // RAMLS5
   .bss:output      : > RAMLS2 //RAMLS3
   .init_array      : > FLASH_BANK0, ALIGN(8)
   .const           : > FLASH_BANK0, ALIGN(8)
   .data            : > RAMLS89  //RAMLS5 | RAMLS6  //RAMLS2
   .sysmem          : > RAMLS2 //RAMLS4
#else
   .pinit           : > FLASH_BANK0, ALIGN(8)
   .ebss            : > RAMLS2 //RAMLS5 | RAMLS6
   .econst          : > FLASH_BANK0, ALIGN(8)
   .esysmem         : > RAMLS2 //RAMLS5
#endif

   .rodata.tvm      : > FLASH_BANK0
   .bss.noinit.tvm  : > RAMGS012
```

- *ステップ 5:*プロジェクト内に ***artifacts*** というフォルダを作成し、コンパイルされたライブラリとヘッダ ファイルをフォルダに配置します。



- *ステップ 6:*ヘッダ ファイルをアプリケーションに含め、NPU で実行されている ML モデルを使用するアプリケーションを記述します。次のセクションで説明する NPU インターフェイスを使用します。



9.2 NPU インターフェイスについて

ヘッダ ファイル (tvmgen_default.h) は、アプリケーションと NPU との間のインターフェイスとして機能する重要な構造と関数を提供します。これらのコンポーネントを理解することは、ニューラル ネットワーク統合を成功させるために不可欠です。

9.2.1 主要なインターフェイス コンポーネント

ヘッダ ファイルによって、次の基本要素を定義します。

- 入力および出力構造 *:*これらの構造は、ニューラル ネットワークとの間でデータをやり取りするメカニズムを提供します。

```
struct tvmgen_default_inputs {
    void* input;  // Points to input data (float for sine case)
};
struct tvmgen_default_outputs {
    void* output;  // Points to output buffer for results
};
```

コード 11: コンパイル プロセス中に生成される入力構造と出力構造

- *初期化関数 :*この関数は、アプリケーションの起動時に 1 回呼び出す必要があり、NPU ハードウェアを構成します。

```
void TI_NPU_init();
```

コード 12: NPU 初期化 API

- *実行関数:*この関数は、NPU でニューラル ネットワークの実行をトリガします。

```
  int32_t tvmgen_default_run(
    struct tvmgen_default_inputs* inputs,
    struct tvmgen_default_outputs* outputs
);
```

コード 13: NPU をトリガする API

- *完了フラグ:*このフラグは、ニューラル ネットワーク処理が完了したことを示します。

```
extern volatile uint8_t tvmgen_default_finished;
```

コード 14: NPU 処理の終了を示すフラグ

9.2.2 基本的な使用パターン

NPU を使用する一般的なワークフローは次のとおりです。

- *ワンタイム初期化:*システムの起動中に TI_NPU_init() を呼び出します。
- *入力の準備:*入力構造体を作成し、データへのポインタを設定します。
- *出力の割り当て:*結果を受け取る変数へのポインタを設定して、出力構造体を作成します。
- *モデルの実行:*tvmgen_default_run() を入力構造体と出力構造体で呼び出します。
- *完了監視:*tvmgen_default_finished フラグを監視して、処理が完了するタイミングを確認します。
- *結果の使用:*完了フラグが設定されると、出力データをアプリケーションで使用できるようになります。

10 組込み環境でのモデルのテスト

トレーニングとモデルの変換が成功した後、実際の展開環境で性能を検証するには、F28P55x マイコン上で正弦関数近似モデルを直接テストすることが不可欠です。このセクションでは、ターゲット ハードウェアでニューラル ネットワーク モデルを実行した場合のテスト方法論と結果について詳しく説明します。

10.1 視覚的なパフォーマンス評価

テスト プロセスの重要な要素は、予測された正弦値と実際の正弦値を視覚的に比較することです。CCS グラフ ツールを使用して、ニューラル ネットワークの予測と真の正弦関数の両方を入力ドメイン全体にわたってプロットしました。

図 10-1 は、NPU で生成される正弦近似と真の正弦関数の値を視覚的に比較したものです。

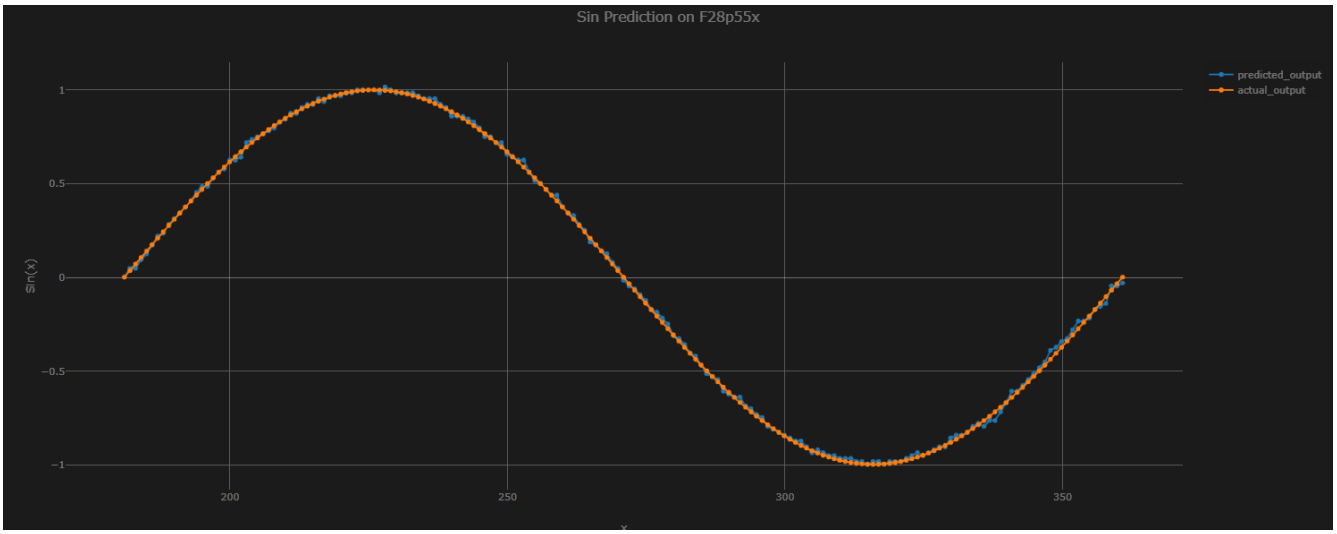


図 10-1. F28p55x のニューラル ネットワーク正弦関数近似性能

この視覚化では、実際のハードウェアでのモデル性能の重要な側面をいくつか示します。

- *一貫した性能:*このモデルは、極値 (山と谷) やゼロ交差などの臨界点を含め、波形全体にわたって高い精度を維持します。
- *スムーズな遷移:*NPU の計算の量子化された性質にもかかわらず、近似は曲線全体で滑らかな遷移を維持し、目に見える不連続性やアーチファクトはありません。
- *完全サイクル範囲:*このテストでは、正弦関数 0 ～ 360 度の全サイクルを評価し、波形の全フェーズで一貫した性能を確認できます。

視覚的な結果は、精度の低下を最小限に抑えながら、モデルがトレーニングから F28P55x ハードウェアに正常に移行できることを示す魅力的な証拠です。予測値と実測値のほぼ区別がつかないオーバーレイは、コンパイル プロセスを通じてモデルの忠実度を維持する際に、量子化対応トレーニング アプローチが有効であることを示しています。

10.2 定量的な性能指標

目視検査にとどまらず、F28P55x NPU ハードウェア上でのモデル性能の客観的な評価を可能にするために、包括的な定量指標が計算されました。定量的評価により、次の主要な性能指標が得られました。

- *平均絶対誤差 (MAE) :*0.01214
- *最大誤差:*0.0973
- *レイテンシ (ms) :*0.214
- R^2 *スコア:*0.9997

これらの指標を使用すると、精度と性能の両方を包括的に評価できます。誤差指標はモデルの近似精度を定量化しており、タイミング測定はメイン CPU でのソフトウェアベースの実行と比較して NPU ハードウェアの効率を示しています。精度と速度をこのように組み合わせることで、車載用や産業用の制御システムで一般的に厳格なタイミングと精度の要件を持つリアルタイム アプリケーションを実現できます。

11 リアルタイム シグナル チェーンでの NPU の統合

事前定義されたデータセットを使用した初期モデル検証により、ニューラル ネットワークの性能に関する貴重な知見が得られますが、NPU アプリケーションの真のテストは、リアルタイム データを処理する能力にあります。このセクションでは、F28P55x NPU 上でニューラル ネットワーク推論を利用してライブ アナログ入力を有意義な出力に変換する、包括的なリアルタイム信号処理チェーンの実装について説明します。

11.1 アプリケーション ブロック図

図 11-1 に、F28P55x マイコンへのニューラル プロセッシング ユニット (NPU) 実装を検証するために作成されたシミュレーション環境の包括的な図を示します。このアーキテクチャ図は、生成から処理、可視化までの包括的な信号フローを示しており、ニューラル ネットワーク モデルがのこぎり波をリアルタイムで正弦波に変換する方法を示しています。

- *信号生成:*左端のセクションでは、0 ～ 360 ユニットの振幅範囲ののこぎり波を生成するための CMPSS モジュールを DAC として構成したものを示します。この波形は、上のボックスのジグザグ パターンで視覚的に表され、システムの入力刺激として機能します。
- *信号収集:*のこぎり波は ADC (A/D コンバータ) ブロックに供給され、アナログ信号がデジタル化されます。出力は同じ数値範囲 (0 ～ 360) を維持しますが、現在はデジタル領域にあります。
- *ニューラル ネットワーク処理:*この図の中央部分は、NPU と正弦モデルを含む F28P55x マイコンを示しています。入力値 (ラベル「x」と) が正弦モデルに渡され、対応する正弦値が計算されます。
- *出力生成:*–1 から 1 (正弦値の自然な範囲) までのニューラル ネットワーク出力は、アナログ領域に再変換するために 2 番目の DAC に送られます。
- *可視化:*右端のセクションでは、最終的な出力がオシロスコープに表示され、のこぎり波から正弦波への変換が成功したことを視覚的に確認するなめらかな正弦波パターンが示されています。

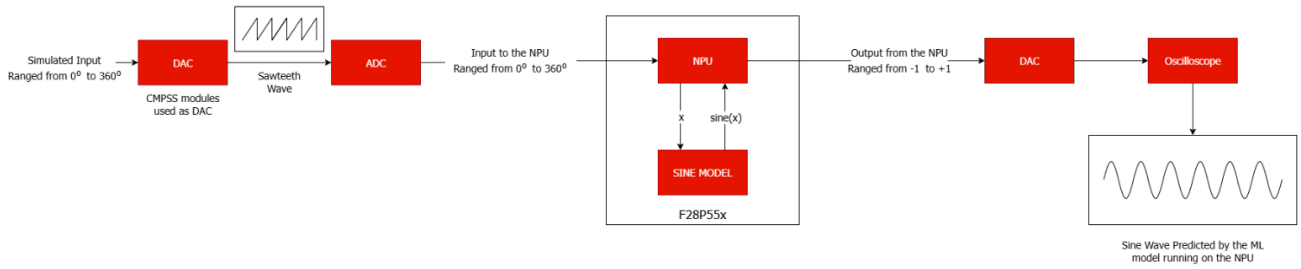


図 11-1. アプリケーション側のデータ フロー

11.2 アプリケーション コードの実装

以下のコードは、完全な信号処理パイプラインを実装し、CMPSS、ADC、NPU、DAC の各コンポーネントを統合して、ニューラル ネットワーク モデルを使用してのこぎり波を正弦波に変換する方法を示しています。

```
while(1){
   for(i = 0; i < MAX_ADC_VALUE; i++)
   {
      // Set the CMPSS low DAC value to generate sawtooth waveform
      CMPSS_setDACValueLow(CMPSS1_BASE, i);
      // Trigger ADC conversion to read back the analog signal
      ADC_forceSOC(myADC0_BASE, myADC0_SOC0);
      // Read the result from the ADC
      adcResult = ADC_readResult(myADC0_RESULT_BASE, myADC0_SOC0);
      // Scale ADC value to phase angle in range [0, 2π)
      input = (float)adcResult * (2.0f * M_PI / (MAX_ADC_VALUE + 1));
      // Prepare input/output structures for the neural network
      struct tvmgen_default_inputs inputs = { (void*)input_arr };
      struct tvmgen_default_outputs outputs = { output_arr };
      // Execute the neural network model to predict sine value
      tvmgen_default_run(&inputs, &outputs);
      // For NPU mode, wait until the neural network processing is complete
      #if defined(TVMGEN_DEFAULT_TI_NPU)
         while (!tvmgen_default_finished);
      #endif
      // Scale the sine output from [-1, 1] to DAC range [0, 4095]
```

```
        dacValue = (uint16_t)((output + 1.0f) * (MAX_DAC_VALUE / 2.0f));
        // Output the predicted sine wave to the DAC
        DAC_setShadowValue(myDAC0_BASE, dacValue);
        // Delay to control the wave frequency
        DEVICE_DELAY_US(WAVE_DELAY_US);  // Delay between samples
    }
}
```

コード 15: F28P55x におけるリアルタイム ニューラル ネットワークをベースとする正弦波生成の主なアプリケーション コード

11.3 利用されているハードウェア コンポーネント

シミュレーション環境では、F28P55x プラットフォームの主要なハードウェア コンポーネントを活用しています。

- *CMPSS DAC (コンパレータ サブシステム) :*通常はコンパレータ機能に使用されますが、このブロックは信号ジェネレータとして転用されます。CMPSS モジュールには、のこぎり波パターンを生成するためにプログラムでインクリメントされる 12 ビット DAC が内蔵されています。このアプローチは、外部ハードウェアを必要とせずに、テスト信号生成を行うための F28P55x ペリフェラルの柔軟性を示します。
- *ADC モジュール:*F28P55x の内蔵 12 ビット A/D コンバータは、生成された波形をサンプリングします。実際のアプリケーションにおいて、これは通常、外部センサまたは信号ソースに接続しますが、このシミュレーション環境では、内部で生成された波形をサンプリングし、完全な信号パスを作成します。
- *NPU ハードウェア アクセラレータ:*F28P55x 内の専用ニューラル プロセッシング ユニットは、メイン CPU のみを使用する場合よりも高い性能と効率で正弦モデル計算を実行します。この図は、NPU が処理チェーンにどのように統合され、デジタル入力を受信し、計算された出力を生成するかを示しています。「Sine Model」は、トレーニング、コンパイル、および NPU に導入されたニューラル ネットワーク モデルを表します。モデルには、入力値を対応する正弦値に変換するために必要な重みと構造が含まれています。
- *バッファ付き DAC :*独立した DAC チャネルによって、ニューラル ネットワークの浮動小数点出力を観測用にアナログ信号へ再変換します。この DAC は、標準的な使用パターンと性能特性において CMPSS DAC とは異なります。

11.4 ハードウェア検証結果

図 11-2 に示すオシロスコープ キャプチャは、NPU アプリケーション全体の決定的な検証を提供し、F28P55x プラットフォーム上で、のこぎり波から正弦波へのニューラル ネットワーク推論を通してリアルタイム変換が成功していることを示しています。

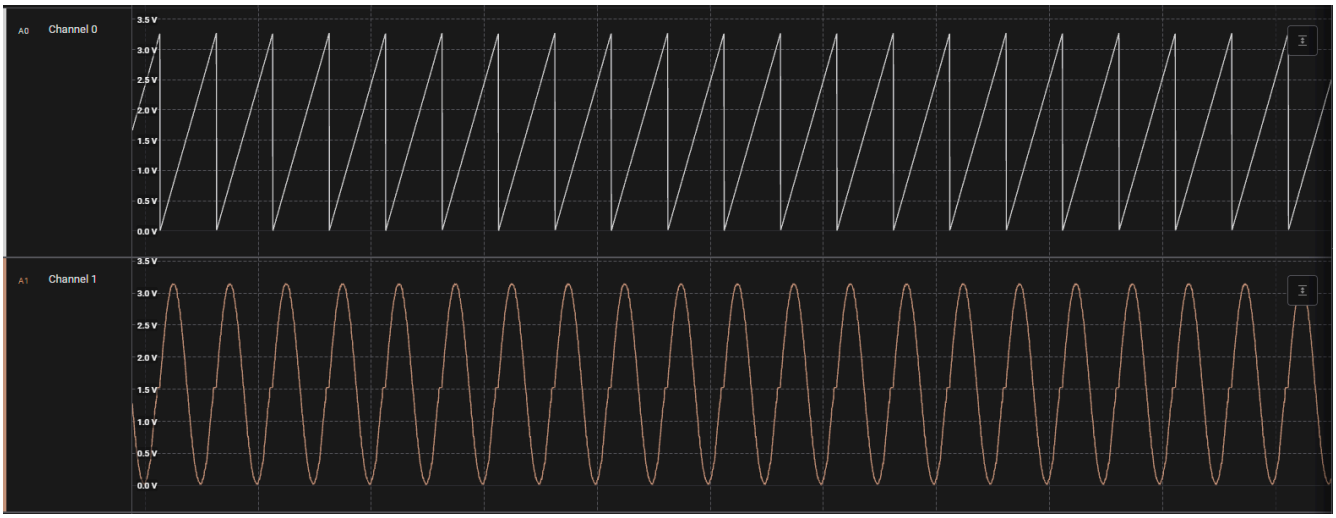


図 11-2. オシロスコープでのアプリケーション検証

11.4.1 入力信号特性

上側のパターン (チャネル 0) は、CMPSS DAC によって生成されるのこぎり波を示します。

- 急峻なリセットを伴う 0V から 3.3V までのクリーンな線形ランプを示します。

- すべてのサイクルにわたって一貫した周期と振幅、つまり安定した信号生成を実現します。
- リセット ポイントでの急激な遷移は、input 関数の不連続性を表すため、ニューラル ネットワークに対して厳しいテスト条件を提供します。

この線形ランプ入力は、各サイクルでニューラル ネットワークの入力ドメイン全体を系統的にスイープするため、理想的なテスト信号として機能し、モデルの動作範囲を包括的にカバーします。

11.4.2 ニューラル ネットワーク出力解析

下のパターン (チャネル 1) は、ニューラル ネットワークによって生成され、DAC 経由で出力される正弦波形を示しています。

- 振幅範囲が 0V ～ 3.3V の滑らかな正弦波波形。
- 可視サイクルのすべてで一貫した振幅と周波数。
- 山と谷の歪みを最小限に抑え、臨界点での優れた近似を示します。
- 入力のこぎり波との適切な位相関係で、各のこぎり波のサイクルは正確に 1 つの正弦波サイクルに対応します。
- ゼロ交差で目に見えるわずかな不完全性は、10 ビット精度のソースのみを効果的に使用した CMP1_DACL の直線性の制約によるものです。

正弦出力の品質は顕著であり、滑らかな遷移と理論的な正弦関数とほぼ一致した整形波形を示しています。複数のサイクルにわたる一貫性は、ニューラル ネットワークが連続動作において信頼性の高い再現可能な結果を生み出すことを示しています。

12 設計上の主な決定事項と影響

12.1 NPU 番号の処理

f28p55x NPU は、整数ベースの処理用に基本的に設計されており、負の値と浮動小数点データの両方を使用する際の課題を提示します。しかし、ニューラル ネットワーク アーキテクチャとコンパイル プロセスに適切な変更を加えることで、NPU はこれらのデータ型を効果的に処理することができます。本書では、これらの制限に対処するための包括的な設計を紹介します。

12.1.1 整数のみのアーキテクチャ

- NPU ハードウェアは整数演算専用に最適化されています。
- ネイティブ サポートは、符号なし整数の計算にのみ存在します。
- このコア設計の選択肢により、組込み型アプリケーションで最大の処理効率を実現できますが、他のデータ タイプでは特別な処理が必要です。

12.1.2 負の値および浮動小数点数の処理

- *量子化プロセス:*負の値と浮動小数点データの両方が、NPU の整数表現にマッピングされるためには量子化が必要です
- *自動変換層:*コンパイル プロセスでは、入力に量子化層、出力に逆量子化層が自動的に挿入されます
- *構成要件:*コンパイル パイプラインは、値の全範囲を保持するように明示的に設定する必要があります。

```
output_int: False  # Enables float output from integer NPU processing
```

コード 16: 逆量子化された浮動小数点数出力を有効にするフラグ

- *特別なコンパイラ フラグ:*浮動小数点または負出力 (正弦関数など) を必要とするアプリケーションでは、追加のコンパイル フラグが必要です。

```
skip_normalize=false output_int=false
```

コード 17: 正規化を無効にするフラグ

- *処理のオーバーヘッド:*追加の量子化 / 逆量子化層は、わずかな計算オーバーヘッドを追加します。

12.2 サポートされているニューラル ネットワーク層と制約

F28p55x NPU は、特定の制約のあるニューラル ネットワーク層タイプの設定をサポートしています。これらの機能を理解することは、このハードウェア上で正常にコンパイルおよび導入できるモデルを設計するために不可欠です。

12.2.1 サポートされている層の種類

12.2.1.1 畳み込み層

- *一次元畳み込み (FCONV) :*単一チャネル入力機能マップをサポートします (深度単位でも、点単位でもありません)。
- *汎用畳み込み (GCONV) :*マルチチャネル入力を 4 の倍数で処理します (深度単位でも、点単位でもありません)。
- *深さ方向畳み込み (DWCONV) :*各入力チャネルにフィルタを適用します。
- *点単位畳み込み (PWCONV) :*チャネル単位のミキシングに 1×1 畳み込みを実装します。
- *点単位畳み込みと残差接続 (PWCONVRES) :*残存学習のためのスキップ接続が含まれます。
- *転置畳み込み (TCONV) :*アップサンプリング操作をサポートします。

12.2.1.2 その他のコア レイヤ

- *全結合層 (FC) :*密度 / 線形動作をサポート。
- *平均プーリング層 (AVGPOOL) :*平均化を使用したダウンサンプル。
- *最大プーリング層 (MAXPOOL) :*最大値を選択したダウンサンプル。

12.2.1.3 フレキシビリティ

- 入力、出力、および残差テンソル データ (特徴マップ) は 8 ビットの整数で、符号付きまたは符号なしです。
- 重みは 2、4、または 8 ビットであり、常に符号付きです。
- 今後、4 ビット データと 4 ビット重みのさらなる組み合わせがサポートされる予定です。

- 推論では、バッチ サイズ 1 のみがサポートされます。
- 畳み込み層のグループ数は、DWCONV 層を除き、常に 1 です。グループ化された畳み込みは今後サポートされる予定です。
- FCONV 入力を除き、入出力チャネル数は 4 の倍数にする必要があります。
- TCONV 層のストライドはカーネル サイズと同じでなければなりません。
- モデル内の層数に制限はありません。
- 層の入力は、層の出力とは異なる符号属性とビット幅を持つことができます。
- 層は混合精度を持つことができます。たとえば、ある層は 8 ビットの重みを使用し、別の層は 2 ビットの重みを使用します。

12.3 モデルの複雑さとサイズの制限

実験では、f28p55x NPU をターゲットとしたモデルの複雑性に大きな制約があることが明らかになりました。これらの制限を理解することは、このプラットフォーム上で効果的なニューラル ネットワーク アプリケーションを開発するために不可欠です。

12.3.1 メモリの制約とモデル サイズ

- *PC と組込み開発の比較:*初期のモデルでは、1 層あたり 1024 ニューロンと 512 ニューロンのサイズは 2.5 ～ 4.2MB でした。これらのモデルは PC で正常に動作しましたが、モデルは F28p55x に完全に適合しませんでした。
- *適切なアーキテクチャ:*最終的には、精度とパフォーマンスの最適なバランスとして、隠れ層あたり 64 個のニューロンを持つはるかに小さなモデルに落ち着きました。
  - *入力層:*1×64 重み + 64 バイアス = 128 パラメータ
  - *隠れ層:*64×64 重み + 64 バイアス = 4,160 パラメータ
  - *出力層:*64×1 重み + 1 バイアス = 65 パラメータ
  - *合計:*4,353 パラメータ (大規模なアーキテクチャより大幅に小さい)
- *物理的制約:*すべてのエッジ コンピューティング デバイスと同様に、F28P55x はオンチップ リソースの制約が厳しいため、このプラットフォーム向けのニューラル ネットワーク設計を開発する際には、モデルサイズの最適化が非常に重要な検討事項となります。

12.3.2 最適化プロセスとパフォーマンスのトレードオフ

- *段階的なサイズ縮小:*異なるモデル サイズを体系的にテストした結果、1 層あたり最大 128 個のニューロンを持つモデルがデバイスに収まる一方で、64 個のニューロン構成が精度と遅延の最適なバランス ポイントであることを発見しました。
- *メモリと精度:*メモリの制約が設計上の主な検討事項となり、精度目標から順方向に設計を進めるのではなく、ハードウェアの制限から逆方向に設計を進めることを余儀なくされました。
- *実装上の検討事項:*最終的なモデルはデバイスに収まるだけでなく、以下が必要です。
  - NPU ツールチェーンで確実にコンパイル済み
  - 正弦波近似に適した精度を維持
  - 1ms 未満の一貫した推論時間を実現

13 ベンチマーク

f28p55x NPU にニューラル ネットワークを導入する場合の性能のトレードオフを定量化するために、さまざまなモデル構成、導入プラットフォーム、最適化アプローチにわたって包括的なベンチマーク測定を実施しました。これらのベンチマークは、組込みアプリケーション向けのモデル アーキテクチャを選択する際の実践的な検討事項に関する重要な情報を示します。

13.1 モデル性能の比較

性能評価には、次の主要指標が使用されています。

- *レイテンシ (ms) :*単一の推論を処理するために必要な時間 (ミリ秒単位)。値が小さいほど、リアルタイム アプリケーションにとって重要な応答時間が短くなります。
- *スループット (サンプル / 秒) :*1 秒あたりに処理できる推論の数。値が大きいほど、処理能力が高くなり、特にストリーミング データ アプリケーションにとって重要です。
- *平均絶対誤差 (MAE) :*予測と実際の値の絶対差の平均。値が小さいほど、予測精度が高くなります。
- *R^2 スコア:*モデルがどの程度データに適合しているかを測定する決定係数。1.0 に近い値は予測性能が優れており、1.0 は完全な予測を表します。
- *最大誤差:*任意の予測値と対応する実際の値の最大絶対差。値が小さいほど、ワーストケース性能が向上します。

各ニューロン構成について、以下の 3 つのモデル バリアントを評価しました。

- *参照 Python モデル:*PC でのみ実行できる標準実装。
- *量子化対応 ONNX モデル:*量子化対応トレーニングを受け、PC 上で ONNX として検証されます。
- *展開された f28p55x モデル:*ONNX モデルは、f28p55x ハードウェアにコンパイルおよび導入されます。

13.1.1 128 ニューロン モデル

表 13-1. Sine_128_Model のベンチマーク

メトリック	F28p55x CPU	F28p55x NPU
レイテンシ [ms]	1.012	0.7116
サンプル / 秒	987	1405
MAE	0.0015	0.0097
R2 スコア	0.9999	0.9996
最大誤差	0.01583	0.04407

13.1.2 64 ニューロン モデル

表 13-2. Sine_64_Model のベンチマーク

メトリック	F28p55x CPU	F28p55x NPU
レイテンシ [ms]	0.2706	0.2146
サンプル / 秒	3695	4659
MAE	0.017525	0.012144
R2 スコア	0.997	0.9993
最大誤差	0.19085	0.0979

13.1.3 16 ニューロン モデル

表 13-3. Sine_16_Model のベンチマーク

メトリック	F28p55x CPU	F28p55x NPU
レイテンシ [ms]	0.0223	0.0252
サンプル / 秒	44643	39557
MAE	0.1588	0.02029
R2 スコア	0.88812	0.8451
最大誤差	0.86379	0.9618

13.1.4 参照ベンチマーク

比較のため、精度のベースラインを確立するために、大規模なリファレンス モデルを PC でテストしました。

表 13-4. 1024 ニューロン モデルのベンチマーク (参照モデル)

メトリック	参照 Python モデル (1024 ニューロン)
レイテンシ [ms]	0.149
サンプル / 秒	7188
MAE	0.002171
R2 スコア	1
最大誤差	0.0054

この参照モデルはほぼ完璧な精度を達成した一方で、f28p55x のメモリ容量を大幅に上回りました (3.5 ～ 4.2MB)。

13.2 性能分析

性能評価では、さまざまな構成にわたってモデルの性能を定量化するために、いくつかの重要な指標を採用しています。

13.2.1 モデル選択のトレードオフ

128 ニューロン モデルは、このアプリケーションの最高精度構成を表しています。このモデルは NPU 上で実行した場合、R^2 スコア 0.9996、平均絶対誤差 (MAE) がわずか 0.0097 と非常に高い精度を達成しています。ただし、この精度は、処理速度の観点で、レイテンシが 0.7116ms、スループットがわずか 1,405 サンプル / 秒の大きなコストをもたらします。

64 ニューロン モデルは、精度とハードウェアの制約とのバランスが取れています。このモデルは優れた精度 ($R^2 > 0.99$) を維持しながら、処理速度を大幅に向上させます。NPU 上で実行する場合、64 ニューロン モデルは 4,659 サンプル / 秒 (128 ニューロン構成の 3 倍以上のスループット) を提供しつつ、予測品質の低下は最小限です。

小型モデルではスループットが大幅に向上しますが、精度を大幅に犠牲にします。16 ニューロン モデルは、NPU 上の 39,557 サンプル / 秒で実行されます (128 ニューロン モデルより 28 倍高速) が、R^2 スコアは 0.8451 に低下し、多くの精密に重要なアプリケーションでは受け入れられない予測品質の大幅な低下を示します。

13.2.2 CPU と NPU の性能比較

CPU と NPU の実行を比較すると、実装の決定を導く重要な知見が明らかになります。

*複雑なモデルにおける NPU の利点:*大型モデルの場合、NPU は大幅な性能向上を実現します。128 ニューロン モデルは、NPU で CPU よりも 29.7% 高速で動作します (0.7116ms 対 1.012ms のレイテンシ)。一方、64 ニューロン モデルでは、レイテンシが 20.7% 短縮されています。この利点は、NPU が並列ニューラル ネットワーク計算のために特化したアーキテクチャに由来します。

*単純なモデルにおける CPU の利点:*興味深いことに、16 ニューロン構成のような非常に小さなモデルでは、実際には CPU が NPU よりも優れています。NPU の 39,557 サンプル / 秒と比較して、CPU は 44,643 サンプル / 秒を達成しており、性能が 12.9% 向上しています。この直観に反する結果は、NPU との間でデータを転送する際のオーバーヘッドに起因しています。16 ニューロン モデルでは、CPU がネイティブ実行環境内で直接処理できる計算負荷が最小限であるため、複数のデータ転送ステップを回避できます。このような小さなモデルでは、NPU が発生するメモリ転送ペナルティは発生しないまま、CPU は単一の実行コンテキストで推論全体を完了します。すべての NPU 推論では、DMA 転送の設定、アクセラレータの設定、完了待ち、結果の取得が必要となります。これらの操作は、この軽量モデルの実際のニューラル ネットワーク計算よりも多くの時間を消費します。本質的に、このモデルが小さい場合、特殊なハードウェアを使用する「コスト」は計算上の利点を超えます。

13.3 パイプライン段のタイミング測定

信号処理パイプラインの各段のタイミングは、F28P55x のハードウェア タイマを使用して測定しました。各部品がシステム全体のレイテンシに及ぼす影響を理解できるように、高精度の計測装置を使用して測定を行いました。

表 13-5. 異なる部品のパイプライン段タイミング

パイプライン段	タイミング (ms)
CMPSS DAC モジュールのソフトウェア書き込み遅延	0.0006
ADC 変換におけるハードウェア遅延	0.00084
NPU 処理 (64 ニューロン モデル)	0.21
バッファ DAC のソフトウェア書き込み遅延	0.000593
パイプラインの合計レイテンシ	0.212

これらの高精度測定により、以下のことが明らかになります。

- ペリフェラルの動作:CMPSS モジュール、ADC 変換、DAC 出力の動作は非常に高速で、それぞれの所要時間は 1µs 未満です。
- NPU の支配的立場:ニューラル ネットワークの推論時間はパイプライン全体で支配的であり、レイテンシ全体の 99.04% を占めています。64 ニューロン モデルでは、NPU の処理時間は、他のパイプライン部品を組み合わせた場合よりも約 210 倍長くなります。
- 固定コスト:すべてのペリフェラル動作 (CMPSS、ADC、DAC) の合計時間はわずか 0.002033ms であり、パイプライン全体のレイテンシの 1% 未満に相当します。

14 まとめ

F28P55x ニューラル プロセッシング ユニット (NPU) は、車載および産業用アプリケーションで組込み機械学習機能の大幅な進歩を遂げ、これらのドメインで重要な確定的性能を犠牲にせずにオンデバイス推論を可能にします。このガイドでは、実用的な正弦関数近似例を使用して、NPU の機能、制約、実装方法を包括的に説明します。

14.1 主な機能と制約

NPU は、ハードウェア アクセラレーションによるニューラル ネットワークの実行を実現し、メイン CPU でのソフトウェア実装を大幅に上回る性能を発揮します。大規模なモデルでは、性能が 20 ～ 30% 向上しています。この整数ベースの計算エンジンにより、他の C2000 ペリフェラルとシームレスに統合しながら、確定的な実行でリアルタイム処理が可能です。

ただし、これらの機能には、モデルの複雑さを制限するメモリの制限、特定のネットワーク トポロジに対するアーキテクチャの優先度、量子化によって生じる精度のトレードオフなど、重要な制約があります。TI のベンチマーク測定により、より大きなモデル、非常に小さなニューラル ネットワークでは NPU が優れていることを明らかにしました

一方で、16 ニューロン モデルのようなより小さなモデルは、NPU 間でのデータ転送のオーバーヘッドにより、CPU 上でより効率的に実行できます。

14.2 開発ワークフロー

実装プロセスは、モデル開発、コンパイル、およびアプリケーション統合を含む構造化されたワークフローに従います。この方法論は、量子化を認識するトレーニングから始まり、NPU の整数のみ処理向けにモデルを準備します。その後、TI のニューラル ネットワーク コンパイラを使用してコンパイルし、ハードウェア互換のアーティファクトを生成します。次に、これらのコンポーネントを適切なペリフェラル構成を持つ CCS プロジェクトに統合して、包括的な信号処理パイプラインを構築します。

14.3 モデル設計の検討事項

このガイドは、系統的な実験を通じて、NPU の実装に成功するためには、精度要件とハードウェア制約のバランスを取る慎重なアーキテクチャ設計が必要であることを示しています。正弦関数の例では、モデル サイズに関する重要な知見が明らかになりました。

- 最適な 64 ニューロン アーキテクチャはメモリの制約の中で優れた精度 ($R^2 > 0.99$) を達成しました。
- 大規模なモデル (128 ニューロン) は、スループットを大幅に低下させながらわずかに高い精度を実現しつつ、CPU 実行と比較して NPU アクセラレーションのメリットを享受しています。
- より小型のモデル (8 ～ 16 ニューロン) ほど高いスループットを実現しましたが、精度が大幅に低下しました。

14.4 実装課題とソリューション

このガイドでは、NPU の実装で発生するいくつかの実用的な課題について説明しています。

- *負の値および浮動小数点数:*適切な逆量子化手法とコンパイル設定を使用して、範囲 [-1, 1]の正弦値を処理します。
- *ニューラル ネットワーク層のサポート:*サポートされていない操作を回避しながら、サポートされている層の種類を活用するモデルの設計。
- *メモリの制限:*ハードウェアの制約の範囲内に収まるようにモデル サイズを体系的に縮小します。

14.5 より広範なアプリケーション

正弦関数近似法で示していますが、このガイドの手法とアプローチは、以下のようなさまざまな車載用および産業用アプリケーションに対応しています。

- 振動または音響信号分析による予防保守。
- センサ データ ストリームの異常検出。
- ニューラル ネットワーク ベースのモデリングを使用する高度な制御システム。
- 認識と意思決定の強化に役立つセンサ フュージョン。

F28P55x NPU は、機械学習を組込み型制御システムに直接組込むことができ、エッジ側で独立して動作するインテリジェント アプリケーションの作成と、車載用と産業用の環境で必要とされる信頼性を維持します。マイコン上でニューラル ネットワークを直接実行すると、システムは他の場所へデータを送信せずに、複雑な決定をローカルで下すことができます。このアプローチにより、安全システムにとって不可欠な応答時間が予測可能になると同時に、従来のソリューションより消

費電力が小さくなります。クラウドとの接続性を信頼できない過酷な環境でも、予防保守、センサ フュージョン、異常検出などのアプリケーションが実用的になります。NPU は、高度な機能と重要な制御システムの厳格な動作要件との間でバランスを取っているため、これらのアプリケーションで必要とされる確定的な動作を犠牲にせずにインテリジェンスを追加できます。

15 参考資料

- テキサス インスツルメンツ、『TMS320F28P55x リアルタイム マイクロコントローラ データシート』、PDF
- テキサス インスツルメンツ、『EdgeAI Studio』、Web ページ
- テキサス インスツルメンツ、Tinyml-Tensorlab、リポジトリ
- 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
- テキサス インスツルメンツ、『C2000 マイコン 用 C2000Ware』、C2000 SDK、Web ページ
- テキサス インスツルメンツ、『F28P55X LaunchPad 開発キット』、製品詳細、Web ページ

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。

最終更新日：2025 年 10 月