Application Note

Using MSPM0G5187 in Edge Artificial Intelligence Applications

ABSTRACT

This application note presents the implementation of Edge Artificial Intelligence (AI) applications on the MSPM0G5187 microcontroller, which integrates the TinyEngine™ Neural Processing Unit (NPU) - a dedicated hardware accelerator designed for efficient neural network inference in resource-constrained embedded systems. The document begins with an overview of the MSPM0G5187 hardware architecture and the key features of the NPU, followed by a comprehensive introduction to TI's Edge AI software ecosystem, including the TI Edge AI Studio, Tiny ML Tensorlab, and Neural Network Compiler (NNC). Two representative application examples are then presented to demonstrate the Edge AI capabilities of the MSPM0G5187: a handwritten digit recognition design based on a LeNet-5 CNN model, and a waveform classifier design based on a time-series CNN model. Quantitative performance benchmarks comparing NPU/CPU-based inference implementations are provided, demonstrating the substantial advantages of NPU hardware acceleration in inference latency and energy efficiency. The results offer practical guidance and design reference for engineers developing edge AI designs on embedded platforms.

Table of Contents

Trademarks

1 Introduction

Edge Artificial Intelligence (AI) refers to the execution of AI inference models locally on devices, at or near the data source. Compared to cloud-based AI, Edge AI delivers significant advantages, including enhanced responsiveness through reduced latency, improved operational efficiency via lower bandwidth requirements and reduced cloud dependency, increased system reliability with continued operation during network disruptions, and strengthened data security through minimized data transmission and enhanced privacy protection.

MSPM0G5187 with TinyEngine NPU have facilitated the integration of AI capabilities directly into end devices across multiple domains, including consumer electronics, smart home devices, industrial and automotive systems. The dedicated hardware NPU operating in parallel enables MCUs to achieve up to 90x lower inference latency and 120x lower energy utilization per inference compared to equivalent MCUs without hardware acceleration.

However, Edge AI is more than just hardware - it requires a complete ecosystem to truly accelerate users' development and deployment. From generic classification, regression to forecasating task, TI's comprehensive software ecosystem empowers engineers to rapidly prototype, optimize, and deploy AI solutions with minimal complexity.

- **TI Edge AI Studio**: offers an intuitive graphical environment for model development, training, compilation, and deployment
- **TI Tiny ML Tensorlab**: delivers an end-to-end workflow covering model training, quantization, compilation, and deployment via command line tool
- **TI Neural Network Compiler**: allows seamless deployment to either the NPU or CPU through simple configuration

MSPM0 SDK further accelerates development by providing a rich collection of ready-to-use Edge AI example projects, enabling developers to rapidly deploy reference code onto MCUs, import pre-built models for retraining, or integrate custom user-developed AI models into applications.

2 MSPM0G5187 with TinyEngine NPU

MSPM0G5187 microcontrollers (MCUs) are part of the MSP highly integrated, ultra-low-power 32-bit MCU family based on the enhanced Arm® Cortex®-M0+ 32-bit core platform, operating at up to 80MHz frequency. These MCUs offer a blend of cost optimization and design flexibility for applications requiring up to 128KB of flash memory in small packages or high pin count packages (up to 64 pins). These devices include a USB2.0-FS interface, digital audio interface, TinyEngine NPU, and provide excellent low power performance across the operating temperature range.

The hardware NPU is a highly optimized core for deep convolutional neural networks (CNNs), supporting machine learning inference using pre-trained models. This works in conjunction with the on-chip CPU to provide higher performance and lower power consumption for CNNs inference. With capability for 640–2560MOPS (Mega Operations Per Second), the NPU enables 90x lower latency than software implementations and consuming less than 2µA in standby mode.

Figure 2-1 shows the top-level view of the modules within the chip.

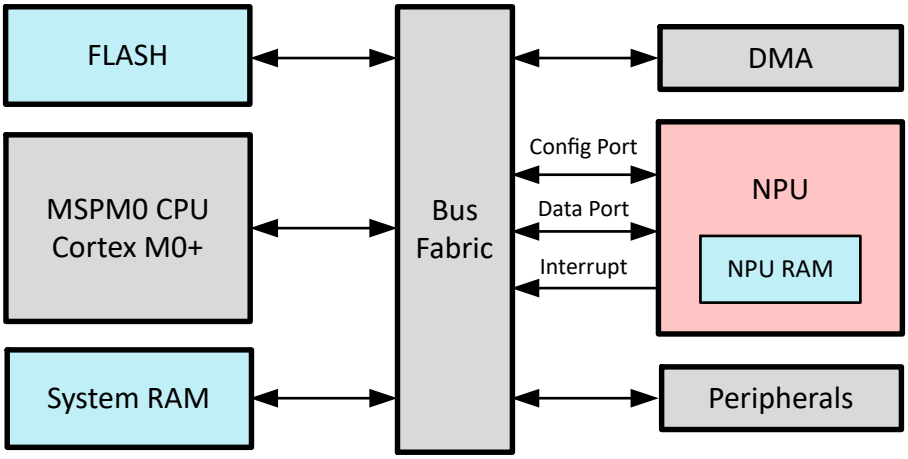

Figure 2-1. MSP CPU/NPU System Structure

MSPM0 with NPU system shared the on-chip memory including non-volatile FLASH memory and volatile RAM memory. The system RAM supports read/write with CPU or NPU. Additionally, NPU has a dedicated RAM for CNNs inference. The Edge AI model must fit within the available FLASH memory range, and the peak RAM requirements must fit within the available system RAM.

3 Edge AI Toolchains

3.1 TI Edge AI Studio

TI Edge AI Studio is a comprehensive suite of graphical and command line tools designed to simplify the development, training, compilation, and deployment of AI models onto Texas Instruments' wide range of processors, microcontrollers, and radar sensors.

Edge AI Studio is available in two versions to accommodate different development environments: the cloud-based version runs entirely online without any local environment setup, enabling developers to get started quickly with minimal configuration; the desktop version runs locally on the user's machine, verifying that all data and trained models remain on-premises and are never accessed by TI, which is preferred for projects with strict data confidentiality requirements.

The Edge AI Studio supports the following features:

- Data capture, annotation and visualization
- Training TI-provided models with user-supplied datasets (Bring Your Own Data, BYOD)
- Training model with customized training parameters
- Compiling trained model
- Live inference preview

The Edge AI Studio currently does not support the following features:

- Training user-defined models (Train Your Own Model, TYOM)
- Modification of TI model architectures
- Customization of the feature extraction pipeline or quantization flows
- Direct compilation of pre-trained or customer-provided models

3.2 TI Tiny ML Tensorlab

The Tiny ML Tensorlab is Texas Instruments' complete design for bringing AI to microcontrollers, which enables users to:

- Train machine learning models for time series and image classification tasks
- Optimize models using quantization (2-bit, 4-bit, 8-bit) for embedded deployment
- Compile models to run efficiently on TI MCUs, with optional NPU acceleration
- Deploy models using Code Composer Studio (CCS)

This supports the following Machine Learning (ML) task types:

Table 3-1. Supported Task Types

Task Type	Description
Time Series Classification	Categorize time-series data into discrete classes (for example, fault detection, activity recognition)
Time Series Regression	Predict continuous values from time-series inputs (for example, torque estimation)
Time Series Forecasting	Predict future values based on historical patterns (for example, temperature prediction)
Anomaly Detection	Identify abnormal patterns using autoencoder-based models (for example, equipment monitoring)
Image Classification	Categorize images into classes (for example, visual inspection, digit recognition)

Additional Resources

- Tiny ML Tensorlab GitHub Repository
- Tiny ML Tensorlab User's Guide

3.3 TI Neural Network Compiler

The TI Neural Network Compiler (NNC) is TI's embedded AI compilation toolchain for MCU devices, built on the TVM compiler framework. TI NNC enables flexible deployment of the AI model to either the hardware NPU accelerator or the host CPU through a simple compilation configuration, with no changes required to the application code or model side. The inference is always invoked through the unified interface, enabling developers to integrate AI inference into embedded applications without requiring deep knowledge of the underlying hardware architecture.

Table 3-2. NPU/CPU Deployment Configuration Comparison

Target Deployment	Configuration Symbol	Initialization Requirement	Invoke Inference
Host CPU	TVMGEN_DEFAULT_TI_NPU_SOFT	Yes, initialize NPU before Inference	tvmgen_default_run()
Hardware NPU	TVMGEN_DEFAULT_TI_NPU	No	

For more information, see TI Neural Network Compiler for MCUs User's Guide.

4 Edge AI Application: Digit Recognition

Digit recognition technology is a fundamental capability with broad applications across embedded and personal consumer markets. However, deploying reliable recognition on embedded systems presents a significant challenge: traditional template-matching approaches lack robustness against the natural variability in human handwriting. The emergence of Edge AI offers a compelling design - enabling accurate, real-time inference directly on the device itself, without reliance on cloud connectivity. CPU-based inference pipelines demand substantial computational resources that exceed the processing capabilities typically available on resource-constrained microcontrollers.

The MSPM0G5187 microcontroller addresses this challenge by integrating a dedicated TinyEngine NPU, designed to accelerate ML workloads with minimal power and memory overhead. The design demonstrates classification of single-digit character images into one of 10 categories (0-9). The application receives a 28×28 grayscale image as pixel data over the UART back channel, performs hardware-accelerated model inference using the on-chip NPU, and sends the recognized digit class back to a host GUI for display.

Leveraging the on-chip NPU, the digit recognition system achieves approximately 99% classification accuracy with an inference latency of only 6.05ms, while consuming just 73KB of Flash and 10.9KB of RAM, as shown in Table 4-1. These results demonstrate the viability of on-device machine learning on resource-constrained microcontrollers.

Table 4-1. Digit Recognition Edge AI Design Performance

Metric	Value
Accuracy	Approximately 99%
Flash Usage	73KB
RAM Usage	10.9KB
Inference Latency (NPU)	6.05ms
Inference Power Consumption (AVG)	424.65uJ

Table 4-2 shows the mode main information.

Table 4-2. Digit Recognition AI Model Information

Property	Value
Model Architecture	CNN
Number of Parameters	60,000
Input Shape	(1, 28, 28)
Output Classes	10
Quantization	INT8

4.1 LeNet-5 Variant CNN Model

LeNet is a pioneering series of Convolutional Neural Network (CNN) architectures originally designed for recognizing handwritten digits and characters from small grayscale images. The digit recognition Edge AI model is a LeNet-5 variant CNN model compiled to NPU-native instructions via the TI Neural Network Compiler.

Figure 4-1 shows its work flow. The solution accepts a flattened 28×28 grayscale image as a 784-element float32 input and produces a 10-element float32 output representing per-class confidence scores. The network consists of two consecutive convolutional blocks, each comprising a Conv2D layer with bias, ReLU activation, and 2×2 MaxPooling. These blocks progressively extract spatial features while halving the spatial resolution at each stage. The resulting feature map is then flattened and passed through a series of fully connected layers, which map the high-dimensional features to a 10-class output. Since all layers operate on INT8 quantized weights and activations, a dequantization post-processing step is applied after the final fully connected layer to rescale the output back to float32. The predicted digit is determined by an argmax over the 10 output scores on the host side.

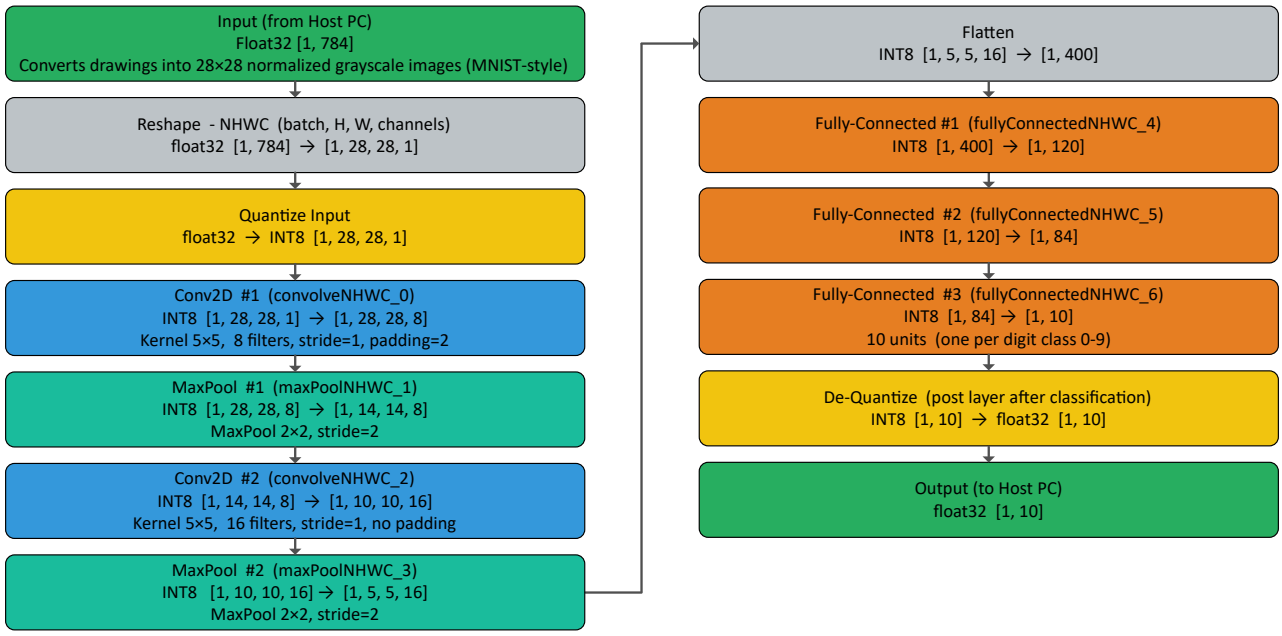

Figure 4-1. Digit Recognition AI Model Flow Chart

4.2 NPU/CPU Performance Comparison

The Edge AI model can be deployed to hardware via TI Neural Network Compiler, targeting either the dedicated hardware NPU or the host CPU. The TinyEngine NPU is a dedicated hardware accelerator specifically designed to execute neural network computations with high efficiency, delivering significantly reduced inference latency and power consumption compared to a general-purpose CPU.

To facilitate performance evaluation, the MSPM0 SDK provides both NPU and CPU implementation examples for user benchmarking, refers to:

- NPU-based Digit Recognition Edge AI Example
- CPU-based Digit Recognition Edge AI Example

Table 4-3 summarizes the performance comparison between the NPU-based and CPU-based Edge AI designs for the digit recognition application.

Table 4-3. NPU/CPU Performance Comparison

Performance Metric	NPU-Based Design	CPU-Based Design
Accuracy	~99%	~99%
Flash Usage	73 KB	68 KB
RAM Usage	10.9 KB	8.1 KB
Inference Latency	6.05 ms	89.81 ms
Inference Power Consumption (AVG)	424.65 uJ	6,265.32 uJ

The NPU-based design achieves approximately 14x lower inference latency compared to the CPU-based implementation and effectively reduces the energy consumption per inference by approximately 93%, making it a highly efficient choice for power-sensitive edge applications.

5 Edge AI Application: Waveform Classifier

MSPM0 provides an experimental waveform classifier design capable of performing real-time, continuous detection and classification of input signals, identifying common waveform types such as sine, triangle, and sawtooth waves. The design is built upon a 13K-parameter time-series classification model, offering strong scalability and extensibility.

The waveform classifier system achieves 100% (R square) classification accuracy with an inference latency of only 5.84 ms, while consuming just 21.5KB of Flash and 3.3KB of RAM, as shown in Table 5-1.

Table 5-1. Waveform Classifier Edge AI Solution Performance

Metric	Value
Accuracy	100%
Flash Usage	21.5KB
RAM Usage	3.3KB
Inference Latency (NPU)	5.84ms
Inference Power Consumption (AVG)	412.90uJ

Table 5-2 shows the mode main information.

Table 5-2. Waveform Classifier AI Model Information

Property	Value
Model Architecture	CNN
Number of Parameters	14,124
Input Shape	Tensor [(1, 1, 128, 1)]
Output Classes	3 (Sawtooth, Sine, Square)
Quantization	INT8

5.1 Feature Extraction

Feature extraction is a critical component of common EdgeAI applications, particularly for audio and time-series signal processing. The feature extraction module provides efficient signal processing functions optimized for embedded systems, leveraging ARM libraries for optimal performance. The available feature extraction functions includes:

- Real Fast Fourier Transform (RFFT): Converts time-domain signals to frequency domain representation
- Magnitude Scaling: Applies scaling for normalizing signal magnitude to appropriate range and prevent overflow/underflow for subsequent processing
- Complex Magnitude Calculation: Calculates the magnitude of complex numbers resulting from FFT
- DC Component Removal: Eliminates signal DC bias for improved feature accuracy
- Bin Calculation: Averages frequency components within defined bins for dimensionality reduction

Figure 5-1 shows the feature extraction processing pipeline.

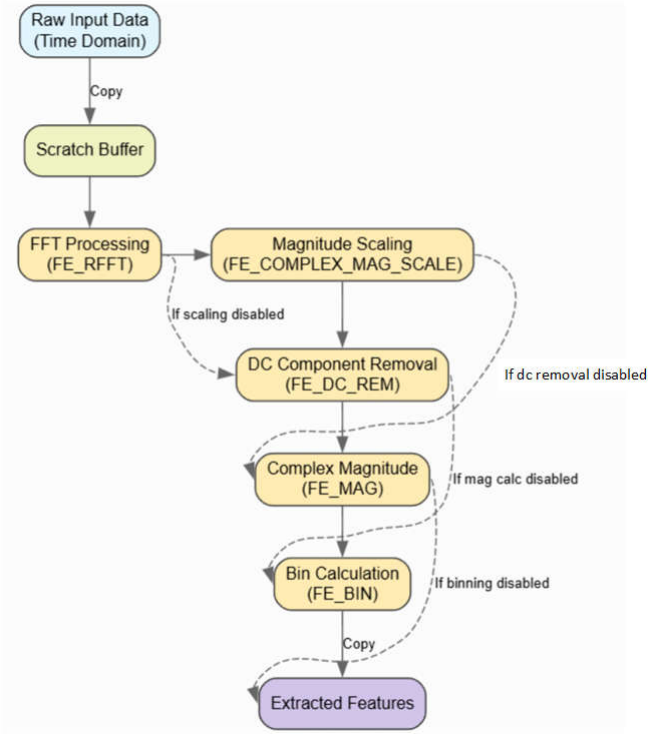

Figure 5-1. Feature Extraction Processing Pipeline

Table 5-3 shows the feature extraction processing pipeline configuration designed for waveform classifier. When the corresponding configuration symbol is defined, its associated task is executed within the processing pipeline.

Table 5-3. Pipeline Configuration for Waveform Classifier

Processing Task	Configuration Symbol	Defined or Not
RFFT	FE_RFFT	Y
Magnitude Scaling	FE_COMPLEX_MAG_SCALE	Y
DC Component Removal	FE_DC_REM	Y
Complex Magnitude Calculation	FE_MAG	Y
Bin Calculation	FE_BIN	Y

Table 5-4 shows the feature extraction configuration parameters designed for waveform classifier. For each inference task, eight frames are concatenated, where each frame processes 256 data points to generate 16 feature dimensions, resulting in a total of 128 feature dimensions fed into the waveform classifier AI model.

Table 5-4. Waveform Classifier Feature Extraction Paramters

Metric	Configuration Symbol	Value
Input Frame Size	FE_FRAME_SIZE	256
Feature Size per Frame	FE_FEATURE_SIZE_PER_FRAM	16
Frequency Bin Size	FE_BIN_SIZE	8
Normalize Bin Values	FE_BIN_NORMALIZE	1
Frame to Concatenate	FE_NUM_FRAME_CONCAT	8
Input Feature Size	FE_STACKING_FRAME_WIDTH	128

Note

The feature extraction parameters configured in Table 5-4 must match the parameters used during model training to verify consistent results.

Performance Consideration

- Optimized Computation: Leverages ARM-optimized libraries for efficient processing
- Memory Efficiency: Buffer swapping minimizes memory copy overhead
- Dimensionality Reduction: Bin calculation reduces feature dimensions, lowering memory usage and computation in downstream stages
- Parameters Tuning: Configuration parameters must be tuned to match available memory and compute resources

5.2 Time-Series Classification Model

Time-series classification identifies specific categories, operational states, or predefined labels within time-series data, and is widely applied across domains including healthcare, wireless monitoring, smart security, and automated testing.

The waveform classifier solution implements a lightweight, 13K-parameter time-series classification model (CLS_13k_NPU), designed for efficient deployment on embedded neural processing units (NPUs). The design employs a 256-point Real Fast Fourier Transform (RFFT) feature extraction pipeline combined with 8-frame concatenation per AI inference task, accepting a 128-element 1D time-series feature vector as input and producing a *3*-element output vector where each element represents the confidence score of the corresponding class.

The exported model file outputs an INT8 (8-bit quantized) tensor from the final fully connected layer. Since the raw output consists of per-class confidence scores in quantized form, a post-processing step is required to derive the final classification result.

Waveform Classifier Model Architecture

The mode network is structured as a sequential feature transformation pipeline, where each stage builds directly upon the output of the previous to progressively refine and abstract the input representation toward a final classification decision. Figure 5-2 shows the workflow.

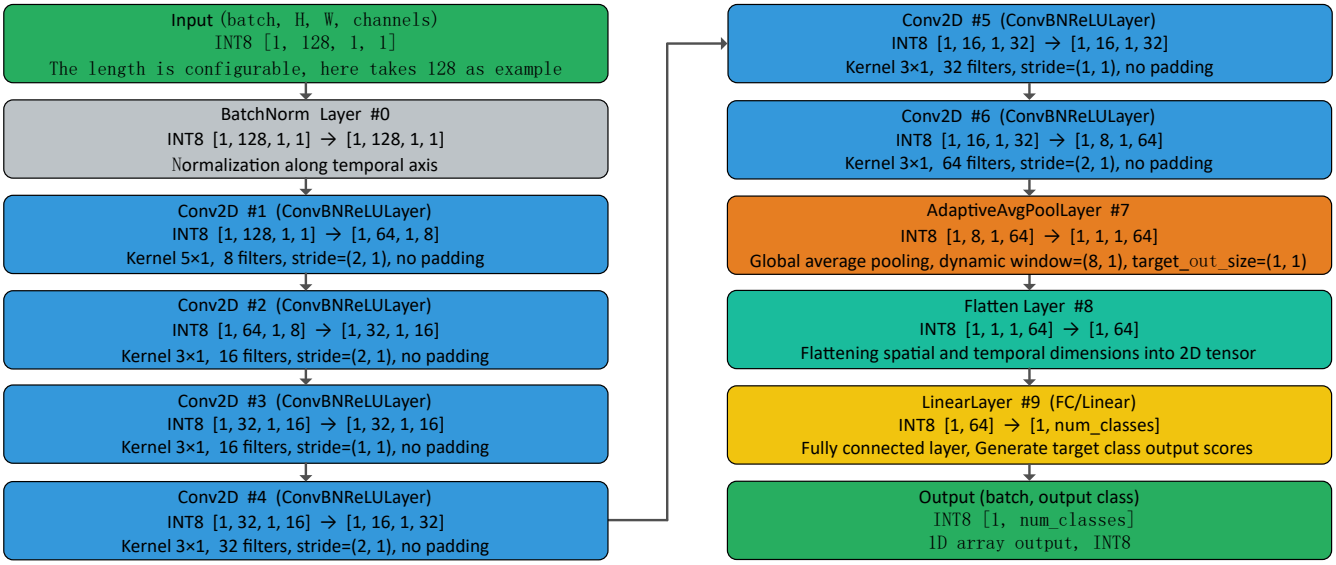

Figure 5-2. Time-series Classification Model (CLS_13k) Flow Chart

The pipeline begins with a B**atch Normalization Layer**, which standardizes the raw input sequence in real time, ensurinsg consistent feature scaling regardless of input signal amplitude variations before any learned transformation is applied. With the input stabilized, the network then progressively deepens the feature representation through six **Consecutive Convolutional Blocks**. Each block follows a consistent internal structure - a **Conv2D Layer** with bias, followed by **Batch Normalization** and **ReLU** activation, forming a repeating pattern that captures increasingly abstract temporal patterns across successive stages. The **Batch Normalization** within each block maintains training stability as depth increases, while the **ReLU** activation introduces the non-linearity necessary for the model to learn complex signal characteristics.

After the six convolutional stages, the resulting feature map passes through an **Adaptive Average Pooling Layer**, which collapses the spatial resolution to a fixed size of 1x1 regardless of the preceding temporal length. This design is particularly significant: by decoupling the model from any fixed temporal dimension, this inherently grants the model compatibility with varying time-series input lengths, providing flexibility for future deployment scenarios without requiring architectural changes.

The fixed-size pooled feature map is then flattened into a 2D tensor and passed through a final **Fully Connected (FC) Layer**, which maps the compressed, high-dimensional feature representation directly to the target class output scores.

Model Output and Post-Processing

The exported model file outputs an INT8 (8-bit quantized) tensor from the final fully connected layer. Since the raw output consists of per-class confidence scores in quantized form, a post-processing step is required to derive the final classification result. Typically, this is accomplished by applying an argmax operation over the output vector, which identifies the class index carrying the highest confidence score as the predicted classification outcome.

5.3 Model Memory Considerations

Developing edge AI solutions on resource-constrained embedded platforms requires balancing a rigid, three-way optimization triad: algorithmic performance (accuracy, latency), non-volatile storage (FLASH/ROM), and run-time memory (SRAM).

- **Algorithmic Performance**: encompasses both inference accuracy and latency, which are primarily shaped by the model architecture - including network depth, layer width, and operator complexity.
- **Static Storage Memory (Flash/ROM)**: This is primarily determined by the model's total parameter count (weights and biases). In a fully quantized INT8 pipeline, each parameter maps directly to exactly one byte of Flash storage. Consequently, deeper networks with expanded channel widths will lineally increase the static binary size. Sometimes, the variations in the input feature dimensions can also influence this static memory usage (e.g., in fully-connected layer).
- **Dynamic Runtime Memory (SRAM/RAM)**: This is governed by the input feature size and the resulting activation tensors (feature maps) across intermediate layers. As a time-series slice propagates through the network, the processing core must allocate temporary workspaces to store layer inputs and outputs. Longer input temporal windows or higher feature dimensions exponentially inflate this peak runtime RAM requirement.

Driven by these hardware realities, deploying at the edge forces a departure from traditional accuracy-first mentalities. Instead, navigating a successful deployment demands a meticulous trade-off, balancing raw model performance directly against the hard physical boundaries of static Flash and peak dynamic SRAM.

To illustrate this relationship, Table 5-5 quantifies the memory footprint and resource utilization of waveform classifier across varying model sizes and input feature configurations.

Table 5-5. Model Memory Analysis

Model Variant (Parameter)	Flash (ROM) Size	RAM Size @ Input = 64	RAM Size @ Input = 128	RAM Size @ Input = 256
CLS_1K	Approximately 5.6KB	Approximately 2.7KB	Approximately 5.3KB	Approximately 10.4KB
CLS_4K	Approximately 9.9KB	Approximately 1.2KB	Approximately 1.7KB	Approximately 2.7KB
CLS_13K	Approximately 21.4KB	Approximately 2.3KB	Approximately 3.3KB	Approximately 5.3KB

Across all input dimensions, the smallest model (CLS_1K) consistently consumes two to four times more runtime RAM than the larger models (CLS_4K and CLS_13K). This counterintuitive behavior stems from the fact that parameter count and runtime memory are driven by fundamentally different architectural decisions. TheCLS_1K relies on a shallow topology that lacks early downsampling, causing large, high-resolution feature maps to persist in memory throughout the intermediate layers. In contrast, CLS_4K and CLS_13K adopt deeper architectures with strided convolutions at the front-end of the network, which immediately reduce the temporal dimensions and significantly decrease the size of intermediate runtime tensors.

5.4 NPU/CPU Performance Comparison

The Edge AI model can be deployed to hardware via TI Neural Network Compiler, targeting either the dedicated hardware NPU or the host CPU. To facilitate performance evaluation, the MSPM0 SDK provides both NPU and CPU implementation examples for user benchmarking, refers to:

- NPU-based Waveform Classifier Edge AI Example
- CPU-based Waveform Classifier Edge AI Example

Table 5-6 summarizes the performance comparison between the NPU-based and CPU-based Edge AI designs for the waveform classifier application.

Table 5-6. NPU/CPU Performance Comparison

Performance Metric	NPU-Based Design	CPU-Based Design
Accuracy	100%	100%
Flash Usage	21.5KB	21.7KB
RAM Usage	3.3KB	3.1KB
Feature Extraction Latency	10.75ms	10.93ms
Feature Extraction Power Consumption (AVG)	752.92uJ	755.8suJ
Inference Latency	5.77ms	54.42ms
Inference Power Consumption (AVG)	412.90uJ	4559.68uJ

In a comparative evaluation of hardware computational efficiency, the NPU-accelerated architecture exhibits substantial advantages in both performance and power metrics over the baseline CPU implementation.

During the pure inference stage, the NPU-based design delivers exceptional acceleration, achieving an approximately 8x reduction in execution latency while driving down per-inference energy consumption by approximately 91%.

When factoring the front-end feature extraction into the end-to-end system profile, the overall efficiency gains exhibit a localized attenuation due to the deterministic workload imposed by signal pre-processing on the CPU; the NPU-based pipeline still maintains a robust 3x latency compression and curtails per-execution energy consumption by approximately 78%.

6 Summary

This application note provides a comprehensive introduction to the Edge AI solution based on the MSPM0G5187 microcontroller, covering the hardware characteristics, software toolchains, and representative application examples.

Two applications are validated to demonstrate the platform capability: the digit recognition solution achieves ~99% classification accuracy, with the hardware NPU delivering 15x lower inference latency and 93% better energy efficiency over the CPU-based implementation; the waveform classifier achieves 100% classification accuracy, with the NPU delivering 8x lower inference latency and 91% better energy efficiency.

These results confirm that the MSPM0G5187, powered by the integrated TinyEngine NPU, is a compelling platform for deploying AI efficiently on resource-constrained embedded devices.

7 References

1. Texas Instruments, *80MHz Arm® Cortex®-M0+ MCU with 128kB flash, 32kB SRAM, USB 2.0 FS, I2S, ADC and TinyEngine™ NPU*, product page.
2. Texas Instruments, *How edge AI-accelerated Arm® Cortex®-M0+ MCUs bring more brain power to electronics*, technical article.
3. Texas Instruments, *TI's TinyEngine™ NPU unlocks edge AI acceleration in more embedded systems*, product overview.
4. Texas Instruments, *AC Arc Fault Detection Using Edge AI on MSPM0 Low-Cost Microcontrollers*, application note.
5. Texas Instruments, *MSPM0 G-Series 80MHz Microcontrollers*, technical reference manual.
6. Texas Instruments, *Digit recognition with TI's microcontrollers (MCUs)*, product page.
7. Texas Instruments, *Tiny ML Tensorlab GitHub Repository*, webpage.
8. Texas Instruments, *Tiny ML Tensorlab User's Guide*, webpage.
9. Texas Instruments, *MSPM0 Edge AI User Guide*, resource explorer.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you fully indemnify TI and its representatives against any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale, TI’s General Quality Guidelines, or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

TI objects to and rejects any additional or different terms you may propose.

Last updated 10/2025