Electronic Design

ADITHYA THONSE, Texas Instruments

Applying Edge AI to DC Arc Fault Detection (Part 2): Software Development to Deployment

Learn about the methodology and tools for AI-driven arc fault detection to create real-time classification on MCUs, improving accuracy and reducing false trips, for edge deployment.

Part 1 of this article series explored how edge AI-enabled microcontrollers can more effectively detect dangerous DC series arc faults in solar, electric-vehicle (EV), and battery storage systems compared to traditional protection methods. Part 2 explores the practical software workflow, from data collection through model training to deployment on real-time control MCUs.

In a typical arc fault detection system *(Fig. 1)*, a current transformer senses current between solar panels and an inverter, while an analog front-end circuit conditions the signal through gain and filtering stages. In a system with a real-time MCU, such as the Texas Instruments (TI) C2000 MCU, its integrated analog-to-digital converter (ADC) digitizes the signal for processing.

Traditional arc fault detection approaches involve integrating frequency energy from the current signal and comparing it against an empirically set threshold. But arc signatures change based on system conditions, requiring the fine-tuning of thresholds for each installation. Fixed thresholds also generate false alarms.

Embedded AI Enables More Accurate and Reliable Detection

Arc fault detection fundamentally is a classification problem: the system must distinguish between normal operating conditions and arc fault events. Unlike threshold-based methods that rely on a single decision boundary, classification models learn to recognize patterns across multiple features simultaneously. These features include frequency content, amplitude variations, temporal characteristics, and harmonic signatures.

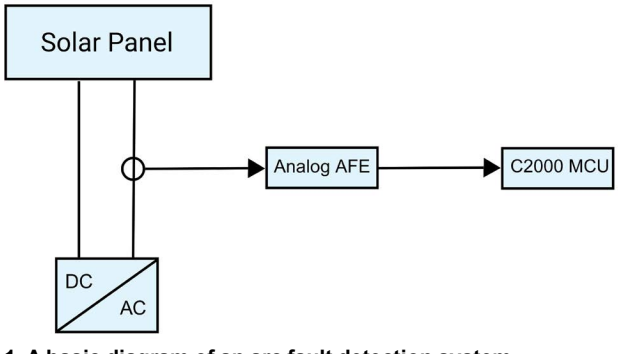
The standard development flow for deploying supervised learning models to edge devices, from data acquisition through training, compilation, and deployment, provides a framework for the rest of this article.

Development Workflow and Tools

Table 1 lists the specific TI tools needed to deploy supervised learning models to edge devices.

Training Data Acquisition

AI model performance depends heavily on training data quantity and quality. Unlike image classification with abundant open-source datasets, inverter configurations vary significantly between systems, limiting the reference value of online data. Different targets and system requirements demand the evaluation of different signals and operating



1. A basic diagram of an arc fault detection system.

conditions.

Best results can be attained by creating setups that closely mimic real-world environments. The TI CCStudio Edge AI Studio supports data collection, or datasets may be imported from external hardware setups. Edge AI Studio provides data visualization capabilities *(Fig. 2)*.

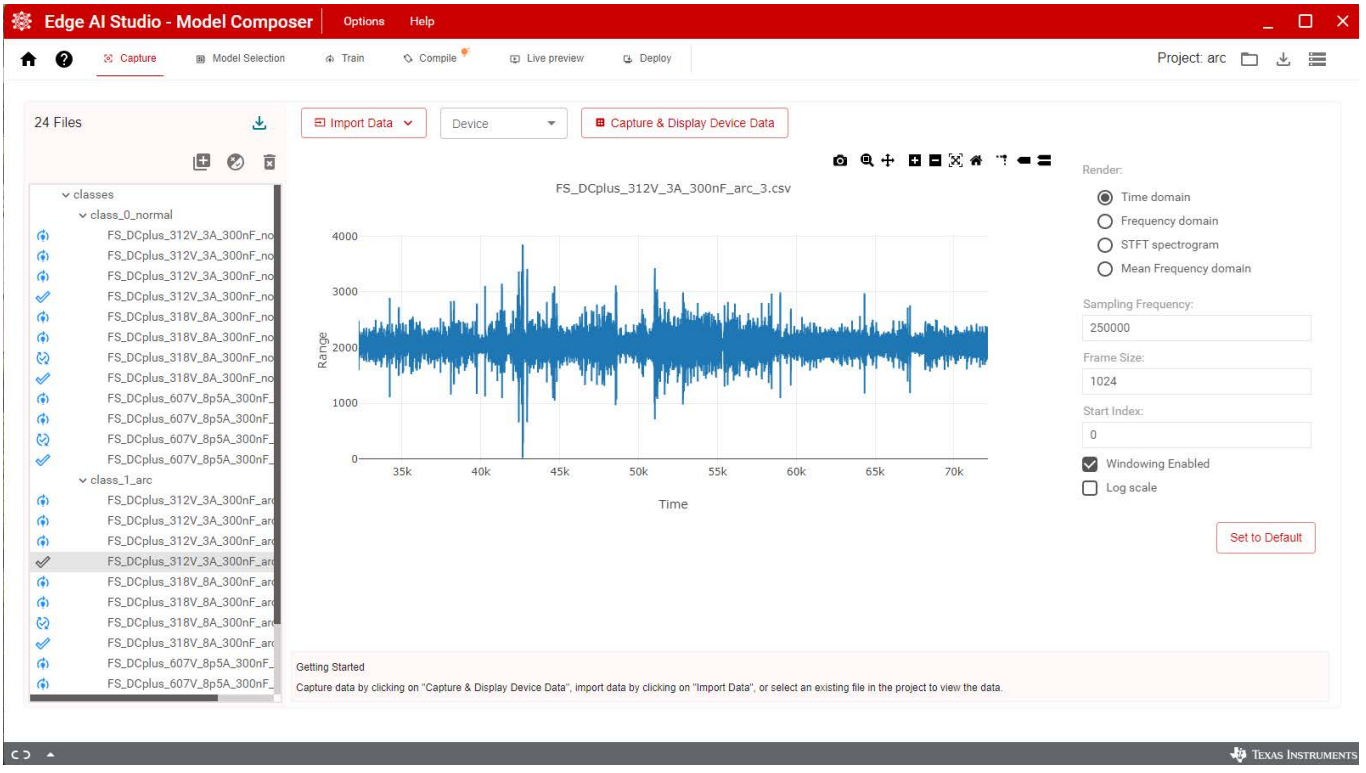
Model Training Options

Selecting the right tool for AI model development can be challenging. It requires navigating unfamiliar frameworks, understanding which model architectures suit the application, and it can require training. Specific toolchains for training AI models on C2000 MCUs can be simplified using

Step	Software tools	Hardware
Training data acquisition	CCStudio Edge AI Studio*	Analog Front End Reference Design for Machine Learning Arc Detection in Solar Applications**** and a C2000 series MCU
Develop and train an AI model	CCStudio Edge AI Studio and Tiny ML Tensorlab**	
Compile and deploy the model on MCUs	CCStudio Edge AI Studio and the DigitalPower software development kit for C2000 MCUs***	TMS320F28P550SJ (or other C2000 devices)
Real-world testing		TMS320F28P550SJ (or other C2000 devices)

Table 1: Edge AI tools for step-by-step development of AI on microcontrollers.

*https://dev.ti.com/edgeaistudio/
**https://github.com/TexasInstruments/tinyml-tensorlab/tree/main
***https://www.ti.com/tool/C2000WARE-DIGITALPOWER-SDK
****https://www.ti.com/tool/TIDA-010955



2. Data visualization on TI CCStudio Edge AI Studio.

Topic	CCStudio Edge AI Studio	Tiny ML Tensorlab
Ease of use	Very easy	Intermediate to advanced for certain features and fine-tuning capabilities (data augmentation, post-training analysis graphs, quantization modes, model addition by user)
Availability	Free on TI.com (cloud and desktop)	Free on GitHub (desktop)
Interaction	GUI-based	Command line-based
Training engine and model optimization	PyTorch-based	PyTorch-based

Table 2. An overview of which tool to use, and why they should be used.

TI CCStudio Edge AI Studio and Tiny ML Tensorlab. *Table 2* shows an overview of which tool a user should use based on their expertise.

Why Do CNN-Based AI Models Work Best for Embedded AI Apps?

AI models and open-source training frameworks, including TensorFlow and PyTorch, were originally designed for image recognition, mimicking how the visual cortex processes information in layers. But they translate surprisingly well to time-series signal data such as current waveforms.

There are a few practical reasons for a convolutional neural network (CNN):

- **Accuracy:** CNNs are good at picking out patterns even when there's noise or variation in the signal.
- **Efficiency:** The convolution operations are repetitive, which means that they can be heavily optimized for embedded targets.
- **Flexibility:** The architecture can be tuned — add or drop layers, adjust kernel sizes — depending on what's being detected.

At their core, CNNs are simply math operations (e.g., convolutions, activations, pooling) along with some logic for decision-making. The tricky part is getting them to run fast on a resource-constrained MCU.

Bridging Python Training and C Deployment

Most engineers train models in Python using mature libraries and tooling, but embedded processors such as C2000 MCUs run C or assembly code. This creates a translation challenge: converting trained models with floating-point operations into efficient MCU-executable code without consuming excessive RAM or CPU cycles.

Manual approaches require writing C implementations for each layer, optimizing memory access, and hand-tuning inner loops, which is tedious and error-prone work. TI's CCStudio integrated development environment (IDE) streamlines the process by handling both training integration and model compilation. Training can be performed inside the framework or external models may be imported, and the compiler generates optimized C code ready for C2000 hardware deployment.

Quantization-aware training is particularly valuable. During training, the framework simulates conversion of weights and activations from floating point to integer (typically 8 bit). This shrinks models significantly, reducing flash and RAM requirements while accelerating inference, with minimal accuracy loss. Models are often a quarter of the size of their floating-point equivalents, with only a marginal reduction in classification accuracy, making this approach transformative for edge deployment.

Pretrained Models and Custom Workflows

When using a tool such as the CCStudio IDE, these models can be loaded with their data, and their performance is able to be evaluated without writing code. Documentation shows training convergence on sample datasets and confusion matrices on test sets, revealing how accurately models identify arcs versus normal transients or other fault types.

This capability accelerates prototyping, allowing for validation of approaches before investing in custom model development.

Custom CNNs trained in PyTorch or TensorFlow can also be imported. The compiler ingests these models, applies optimizations (quantization, layer fusion, memory layout adjustments), and generates C code tailored for the C2000 architecture. This code can then be integrated into firmware, connect ADC sampling, and run inference on hardware.

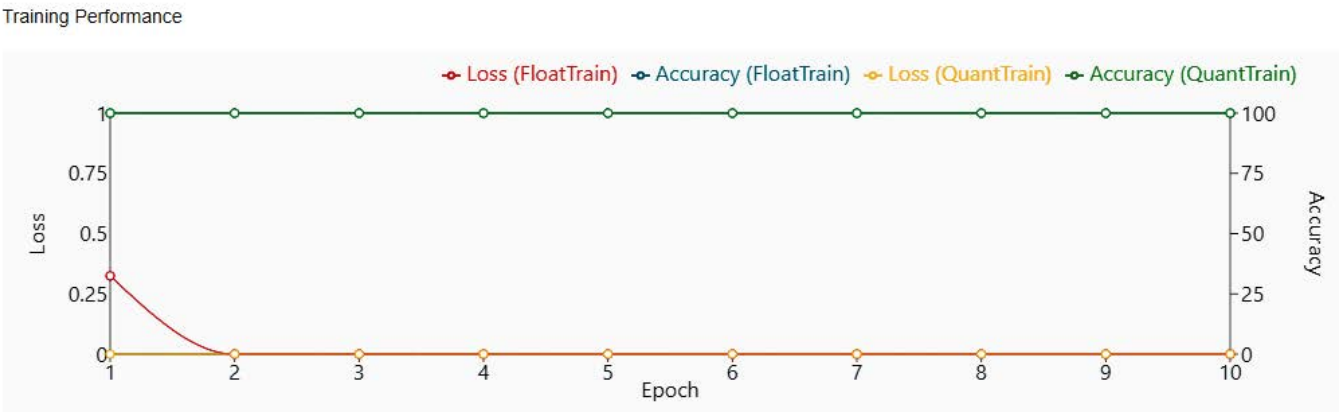
Initial iterations typically encounter challenges such as input preprocessing mismatches and RAM limits that require layer pruning. But once calibrated, MCU performance is impressive, with inference times well under 1 ms on moderately complex models.

Advanced Analysis with Tiny ML Tensorlab

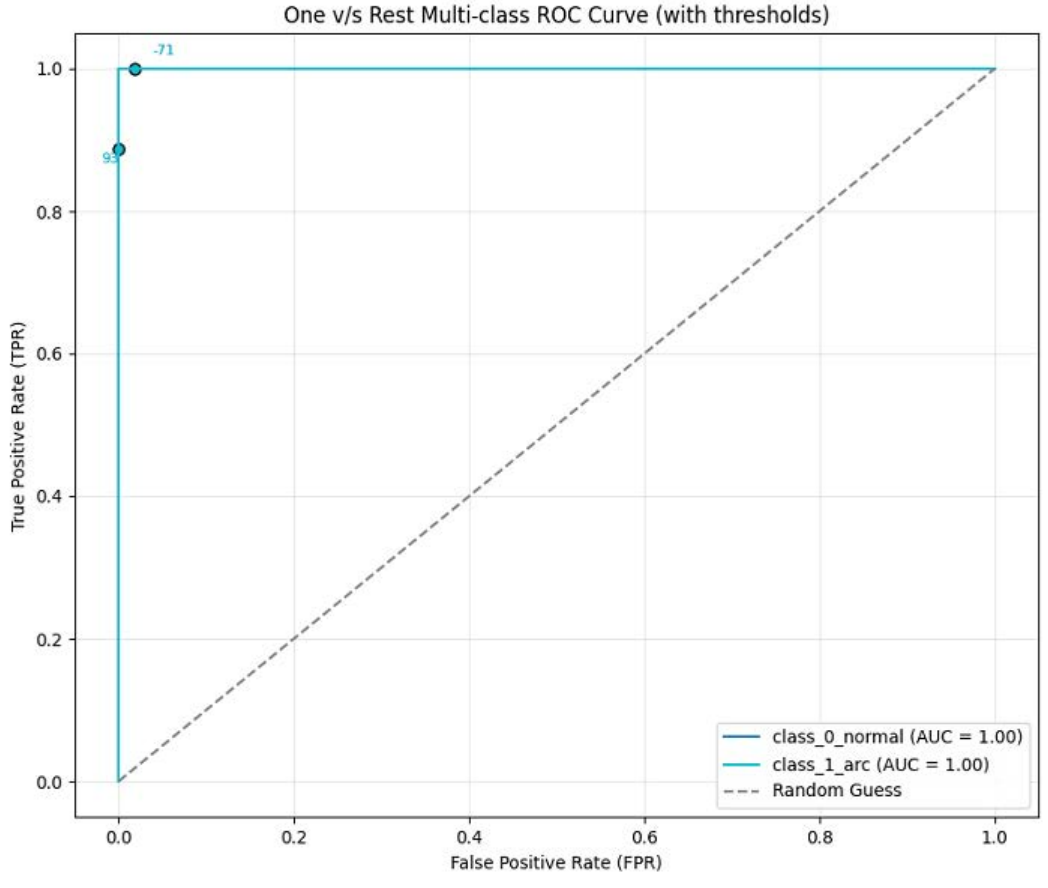
Tiny ML Tensorlab generates several informative graphs to aid model optimization.

Figure 3 shows the feature extraction visualization graph. Tiny ML Tensorlab supports multiple feature extraction techniques, such as fast Fourier transform, binning, logarithmic scaling, frame concatenation, and magnitude calculation, which are applied individually or in combination. Cluster visualization shows how effectively the feature extraction separates classes in the processed data.

Figure 4 shows the one-vs.-rest multiclass receiver operating characteristic (ROC) curve graph. This curve evaluates multiclass classification performance by plotting each class as positive against all others as negative. The area under the curve measures model performance, with higher values indicating better discrimination. Curves above the diagonal



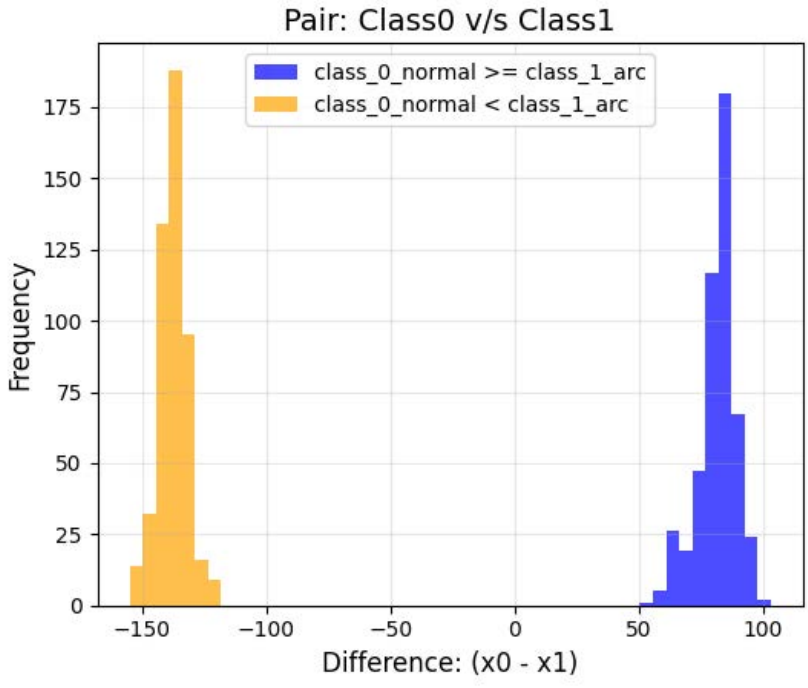
3. Training Loss and Accuracy progression through epochs.



4. The one-vs.-rest multiclass ROC curve.

line demonstrate good classification, while curves hugging the left and top borders indicate excellent performance.

Figure 5 depicts the pairwise differenced class scores graph. This histogram shows prediction score differences between classes for each test sample. For a two-class problem, if the model outputs probabilities (41, –30) for input X, the tool plots a point at 71 (that's expressed as, *41 – (–30)*). Distributions at histogram extremities indicate clear class distinctions, while points near zero reveal samples that the model has difficulty classifying.



5. Histogram of data samples between two classes (normal vs. arc).

Deployment on MCUs

After satisfactory training results and successful compilation, it's time to deploy. Both training tools generate identical artifacts that integrate directly into the DigitalPower software development kit arc fault examples, creating functional edge AI-based arc fault detection applications for C2000 boards.

For integration into existing single-MCU applications, model execution timing, processing duration, and model size must be carefully considered. The software design must align with system requirements.

In TI's fault detection and predictive maintenance examples, model execution triggers every few milliseconds with total processing times also within milliseconds, enabling proactive detection without impacting critical tasks. The TMS320F28P550SJ includes a hardware accelerator dedicated to neural-network computation and offloading processing from the main CPU.

Real-world performance may not match training accuracy given the differences between lab-collected training data and actual deployment environments. Adequate coverage of intended operating conditions and close alignment with real environments should be validated; the process will likely require iterative refinement.

Conclusion

As AI technology and edge AI capabilities advance, new applications continue to emerge, bringing greater convenience and intelligence to power systems. AI-based approaches fill gaps in current detection methods and significantly improve system reliability and the user experience.

Embedding intelligence at the measurement point makes it possible to deliver safer, more reliable, and future-proof power solutions, transforming arc faults from catastrophic surprises into detectable, manageable events.