WHITE PAPER

Machine Learning in Intelligent Power Management Systems

Paul Gorday, Director of DSP and Machine Learning
Vivek Chandrasekharan, Principal Systems Engineer

Executive Summary

Machine learning for MCU implementation (tiny ML) is a growing field that offers new and enhanced functionality for battery management and motor control. ML algorithms discover information and patterns in complex sensor data that can be used to optimize performance and improve understanding of overall system health. In addition to advances in tiny ML techniques, the availability of AutoML tools that automate the collection of data, training of ML algorithms, and generation and deployment of MCU firmware is on the rise. Such tools, combined with access to system on chip (SoC) sensor data, enable the development of ML-based solutions in today's power management systems. This paper discusses the development of machine learning (ML) applications using Qorvo's intelligent power management systems ICs. Qorvo's highly integrated power management SoCs combine Arm® Cortex® M0 and M4F MCUs with an analog front end with an array of sensors to enable smart control and monitoring functions.

Introduction

Learning from Data

Machine learning models derive their performance directly from data, which is an advantage when the data is complex, high-dimensional, or difficult for a human to determine an optimal algorithm. However, reliance on data means that a good outcome depends on access to good training data – data that is representative of actual device behavior across different environmental conditions and manufacturing tolerances. Data collection is often the most expensive and time-consuming phase of an ML project. In cases where the costs are prohibitive, synthetic data from physical models can satisfy requirements.

Qorvo's evaluation kit includes a graphical user interface (GUI) that incorporates data logging features to facilitate the data collection process. The same sensor data available to the internal MCU can be saved to files on a computer for offline model training and testing. Some AutoML (automated ML) tools also support live data collection and model testing by integrating hardware abstraction layers and data streaming functions to the evaluation firmware.

Model Selection

Another advantage of machine learning is scalability. A single model can be trained for different use cases by simply replacing the training data. A parameterized code structure can be used to implement the model, with a different set of model coefficients stored for each use case. This process can be automated, thereby saving engineering time and resources associated with manual firmware modifications.

Machine learning models vary significantly in complexity and performance. A low-complexity approach that is included in most AutoML tools is the decision tree or an ensemble of decision trees. These are well-suited to MCU implementation and may require a few hundred bytes to a few Kbytes of RAM/flash for implementation. While decision trees are typically used for classification applications, a related structure known as a regression tree can be used to estimate continuous values.

An example classification decision tree is illustrated in **Figure 1**. Two features are used to characterize a multi-cell battery pack during discharge: state-of-charge for the strongest cell, and voltage difference between the strongest and weakest cell. At each decision node in the tree, a binary decision divides the feature space in half. The successive layers of nodes subdivide the feature space into smaller regions, and the result is a coarse nonlinear decision boundary that attempts to separate the data into two classes (e.g., normal and abnormal). In the figure, training data is shown as black or white points, while the classification regions used by the decision tree are represented by the blue or gray areas.

Neural networks are also included in AutoML tools. Neural networks offer smooth decision boundaries and improved performance at the cost of increased implementation complexity. **Figure 2** shows the model diagram and classification regions for a simple 2-layer neural network trained from the same data used in the decision tree example. Each neuron in the network performs a weighted sum of its inputs followed by a nonlinear activation function. Comparing the classification regions in these two examples, the neural network is expected to perform better on data not seen during training. However, as the number of data features grows, the implementation complexity of the neural network can become prohibitive for MCU implementations.

Other common ML models include support vector machines (SVM), kernel approximation classifiers, nearest neighbor classifiers, naïve Bayes classifiers, logistic regression and isolation forests. A useful feature found in many AutoML tools is automatic model selection and automatic tuning of model hyperparameters (number of nodes, activation types, etc.) for best performance. Some tools also include target processor type and resource budgets for accuracy, latency and memory when training and comparing models [1].

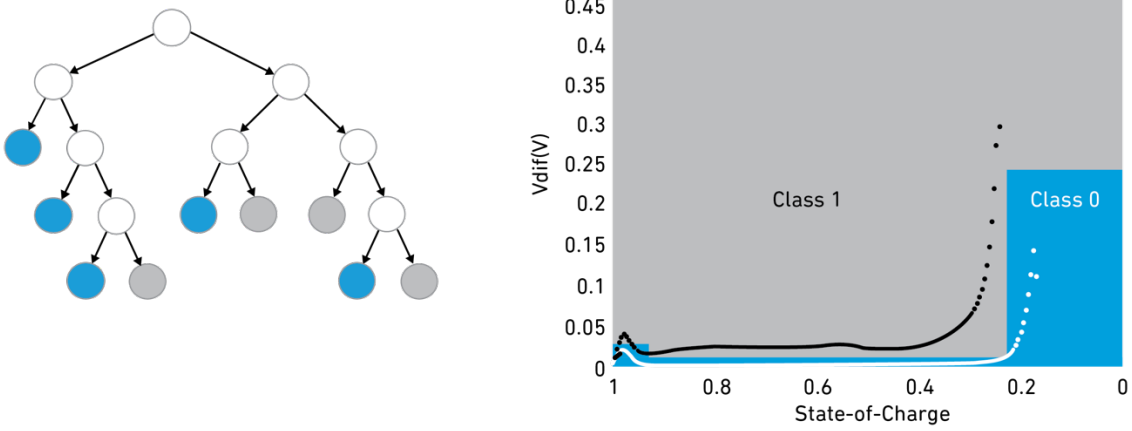
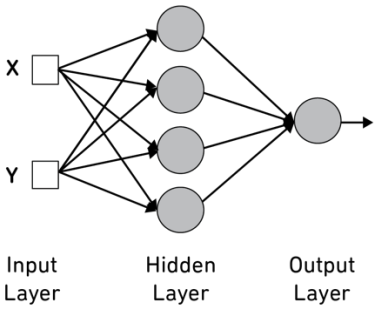


Figure 1. Simple decision tree classifier.



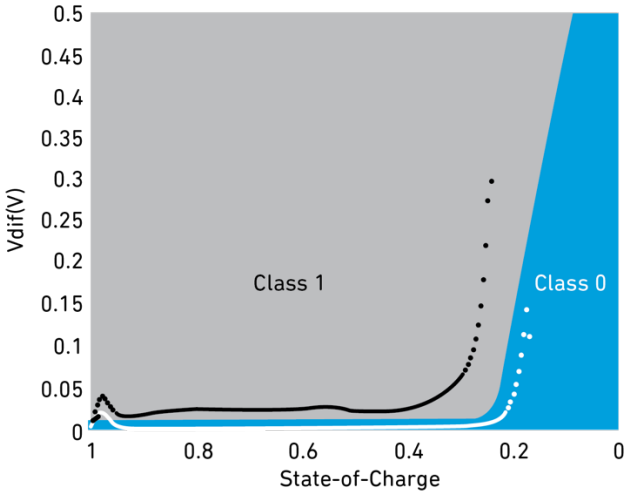


Figure 2. Simple neural network classifier.

Model Deployment

Another important feature of AutoML tools is automatic code generation for trained ML models. In most cases, code generation can be targeted at specific processors, such as the Cortex M0 and M4 processors used in Qorvo's intelligent power management SoCs. This is often referred to as a "no code" or "low code" feature of the tools, since it eliminates or minimizes the need for hand-coding. Depending on the tool, code may be provided as editable source code or as compiled binaries that can be integrated into the power management firmware.

ML Development Flow

Successful ML development is often an iterative process involving data collection and curation, model training and analysis and model deployment and monitoring (**Figure 3**). The term MLOps (ML operations) is used to describe this process [2]. Performance monitoring is a key element of MLOps that ensures actual performance matches expectations. Discrepancies can often be traced to field conditions that are not represented in the training data, and a new round of data collection and training may be required.

Figure 3 depicts an ML development flow for intelligent battery management system (BMS) applications. It leverages Qorvo evaluation kits, such as the PAC25140EVK1 [3], which include a graphical user interface (GUI) for status monitoring, HW/SW setup and sensor data logging. Data log files in .csv format can be imported into AutoML tools, where feature selection, model training, model analysis and code generation are performed. The resulting model code is integrated into the BMS firmware and deployed to the EVK for physical testing and monitoring.

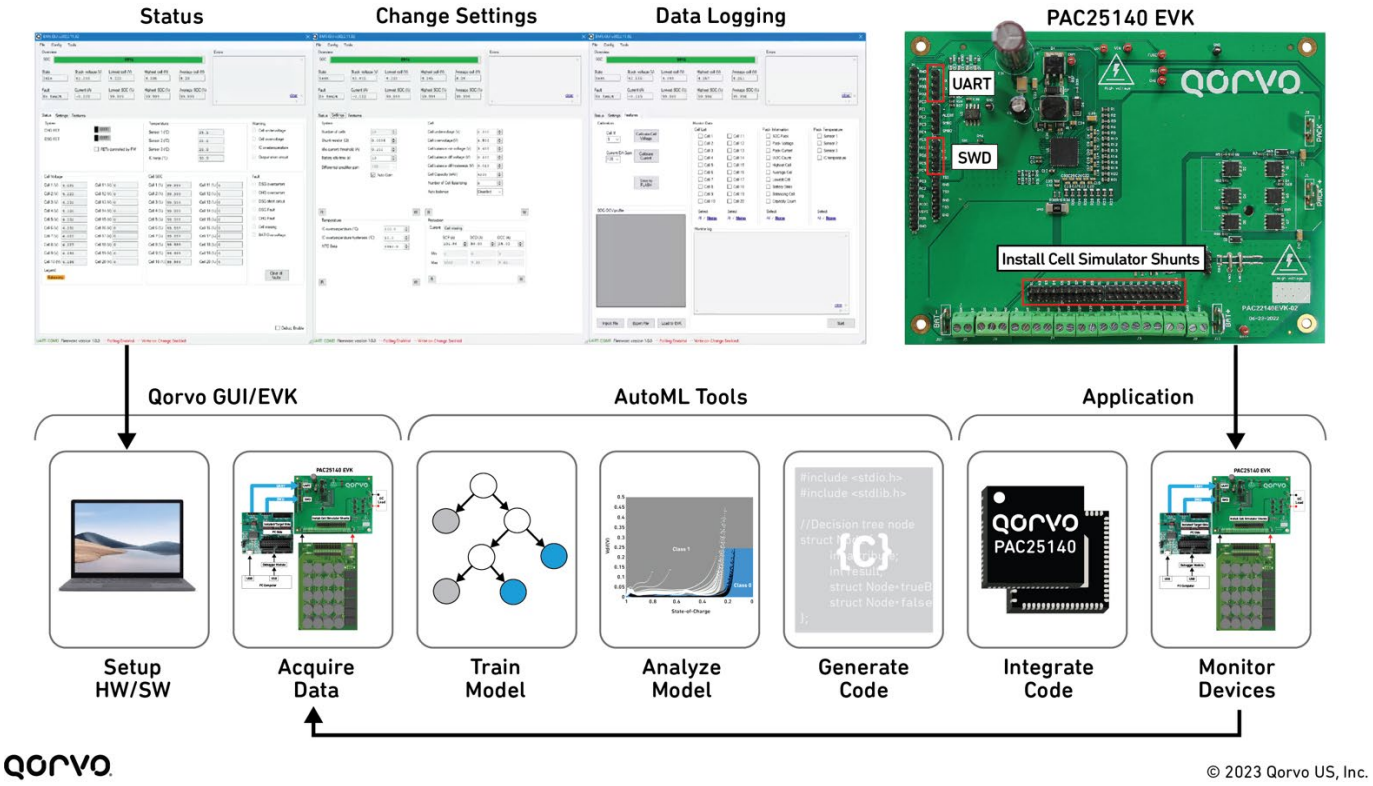


Figure 3. ML development flow.

Implementation Example: BMS Weak-Cell Detection

Detection of weak cells in a multi-cell battery pack is one of the functions of an intelligent battery management system (BMS). A straightforward approach is to compare the loaded voltage of all cells in the pack and determine if one is significantly different than the others. **Figure 4** shows discharge cycles for a strong and a weak cell at various temperatures. The figure illustrates that features like current, ambient temperature and state-of-charge all affect the voltage difference between the cells and make it difficult to distinguish the weak cell with a single threshold test until the cells are nearly depleted. Detecting the weak cell early in the discharge cycle helps minimize overheating and related safety issues. Machine learning offers a potential solution to this problem due to its ability to find patterns in complex feature sets. The characteristics mentioned above are monitored by Qorvo's intelligent BMS solutions, and those characteristics can be processed by a tiny ML algorithm in the embedded MCU leading to improved performance.

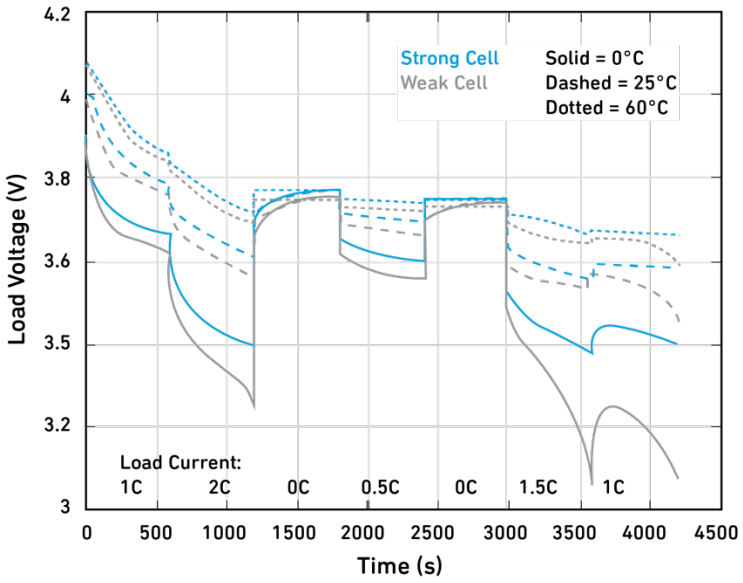


Figure 4. Example discharge cycles for strong and weak cells.

To further explore this approach, a set of training data was created using 20 discharge cycles for two 10s battery packs. The first battery pack (Class 0) included 10 strong cells, and the second pack (Class 1) included 9 strong cells and 1 weak cell. During each of the 20 discharge cycles, both packs were given a constant load current between 0.2 and 4C (where C is the capacity of the battery), and a constant ambient temperature between 0 and 60 degrees Celsius.

Four features were selected for the weak cell detection:

- *i_series* = series current through all cells in the pack
- *t_amb* = ambient temperature
- *v_diff* = voltage difference between strongest and weakest cell in a pack
- *soc* = state-of-charge for the strongest battery in the pack

The four features were sampled synchronously to create a training instance, [*i_series, t_amb, soc, v_diff*]. The 20 discharge cycles for both classes were sampled every 10 seconds, yielding more than 11,000 training instances. The training data was used to create several ML models with Matlab's "Classification Learner" app [4]. After training, classification accuracy was re-evaluated using an independent set of test data (20 additional discharge cycles for each class). Results are summarized in **Table 1** below. Note that the table results represent classification of individual sample instances. It is anticipated that multiple sample results will be combined to form more robust decisions about the health of the battery pack.

Model	Training Data			Test Data			Complexity
	Accuracy %	False Detection %	Missed Detection %	Accuracy %	False Detection %	Missed Detection %	
Decision Tree (depth=4, splits=8)	96.1	2.9	5.0	94.3	6.1	5.2	Low
Decision Tree (depth=6, splits=18)	97.3	2.1	3.3	96.7	3.0	3.6	Low
Tree Ensemble (depth=4, 30 trees)	97.3	2.4	3.0	96.4	3.8	3.4	Medium
Neural Network (1 hidden layer with 4 nodes)	97.1	3.7	1.9	97.5	3.6	1.3	Low

Table 1. Model training results for weak cell detection example.

Summary

MCU-based machine learning, known as “tiny ML,” is an emerging technology that offers promising enhancements for power management applications. Examples include fault and anomaly detection for improved safety and reliability, virtual sensing of system parameters for reduced system cost, and new predictive maintenance applications for increased system efficiency and reliability. Qorvo’s intelligent power management SoCs include ARM Cortex MCUs and an array of analog sensors from which the tiny ML algorithms can learn and improve performance. Qorvo offers evaluation kits that support both data collection and firmware testing, which, when combined with 3rd party AutoML tools, provide the necessary elements for ML algorithm development for today’s SoCs.

References

1. State of the tinyAutoML Market, June 10, 2022. Available: https://www.tinyml.org/event/auto-ml-forum/
2. tinyML Deployment Working Group White Paper #1, February 20, 2023.
   Available: https://www.tinyml.org/news/tinyml-deployment-working-group-white-paper
3. PAC25140EVK1. Available: https://www.qorvo.com/products/p/PAC25140#evaluation-tools
4. Matlab Classification Learner.
   Available: https://www.mathworks.com/help/stats/train-classification-models-in-classification-learner-app.html