UG103.19: Machine Learning in Embedded Applications Fundamentals

For small, embedded devices like Silicon Labs EFR32 living on the Tiny Edge, Machine Learning (ML) is a sophisticated way to detect and identify patterns. ML can be used as a feature to enhance embedded software applications for a number of use cases. This guide provides an introduction to ML in embedded applications, discusses ML model development, and explains the key challenges for using ML as a feature.

KEY POINTS

- Introduction to Machine Learning at the Tiny Edge
- Background on Machine Learning models
- Developing Machine Learning models
- Deploying Machine Learning as a feature
- Key challenges

1 Introduction

Machine Learning (ML) is a sophisticated way to detect and identify patterns. That detection can take the form of classification (such as recognizing a keyword) or anomaly detection (such as detecting vibrations in a bearing that is failing).

ML can be implemented in many ways, depending on the use case. One common distinction is the proximity of the sensor to the processor. For most embedded software use cases, the sensor and the processor are in the same device. This is also called Edge Machine Learning or, in the context of microprocessors, Tiny Edge. According to the tinyML Foundation, "Tiny Machine Learning is broadly defined as a fast-growing field of machine learning technologies and applications including hardware (dedicated integrated circuits), algorithms and software capable of performing on-device sensor (vision, audio, IMU, biomedical, etc.) data analytics at extremely low power, typically in the mW range and below, and hence enabling a variety of always-on use-cases and targeting battery-operated devices."

1.1 ML Using Neural Networks – An Analogy

ML can be thought of in two main themes – the **platform** on which you create and run an ML task, and the **model** representing the specific task designed for your application. A **neural network** is one of the most powerful ML methods that can be deployed on a small, embedded device. To understand how ML using neural networks works, it helps to look at how the human brain processes information in order to identify something. Your brain has millions of neurons working on ways to recognize things. Think of this as an ML platform.

Then, consider how you recognize a cat. You do not maintain a library of cat images and compare what you see to each of those images. Instead, you have some experience seeing cats and their distinguishing features. In effect you have a "model" in your brain of what a cat looks like. You built that "model" when you saw your first cat and someone told you, "This is a cat". You saw more cats over time, each time being told, "This is a cat". Based on that, you extracted features that defined 'cat', like four paws, long tail, and whiskers, which became your idea of what a "cat" is. This is like an ML model. At some point you may see a tiger, and someone tells you, "That's a tiger," and you train your internal 'cat' model to recognize a new pattern.

TensorFlow is one of the most commonly used ML platforms. TensorFlow (https://www.tensorflow.org/) is an end-to-end open-source platform that provides a collection of workflows to develop and train models. TensorFlow Lite converts a pre-trained model in TensorFlow to a special format that can be optimized for speed or storage. Silicon Labs provides TensorFlow Lite for Microcontrollers in the Gecko Software Development Kit Suite (GSDK). TensorFlow Lite for Microcontrollers (https://www.tensorflow.org/lite/microcontrollers) is designed to run ML models on microcontrollers and other devices with as little as 16 kB of RAM.

While Silicon Labs provides the platform to incorporate into your embedded application, we do not provide models. Silicon Labs does, however, partner with a number of model tool and solutions providers, and documents different approaches to selecting the best tool for your needs. See https://docs.silabs.com/machine-learning/latest/machine-learning-overview/ for details.

1.2 ML in Embedded Applications

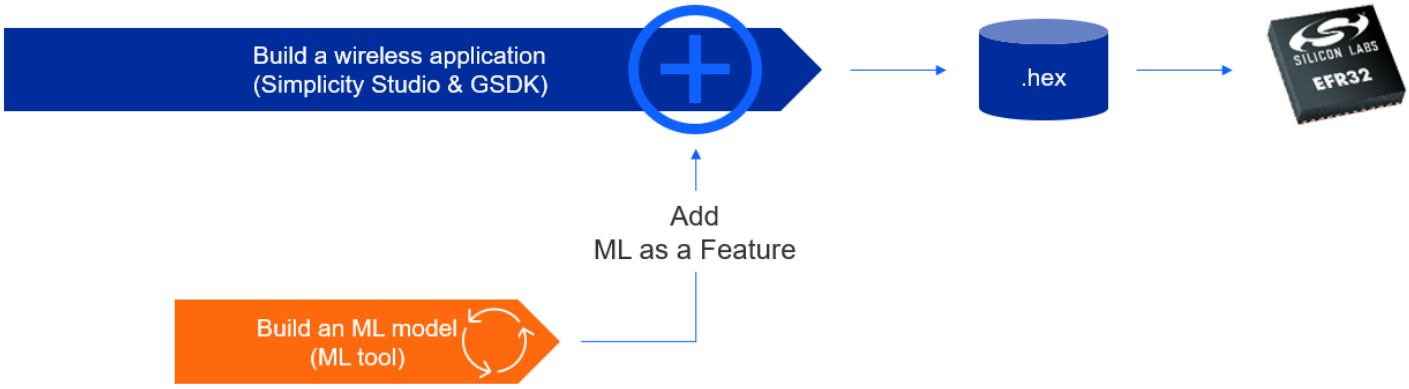
Although an application can be built solely around ML, Silicon Labs expects most of our customers will use ML to add some new differentiated functionality to their embedded wireless products. We call this "Machine Learning as a Feature". ML is a horizontal technology that can be used to enhance products in all different markets and market segments. ML at the Tiny Edge means that patterns are evaluated locally, rather than being sent to the cloud. Some examples of Tiny Edge applications are:

- **Sensor Signal Processing** uses low data-rate sensors, like accelerometers, gyroscopes, air quality sensors, temperature sensors, or pressure sensors. These use cases can be found in many markets, such as preventive maintenance, medical devices, and smoke detectors.
- **Audio Pattern Matching** uses a microphone to detect different sounds. The types of sounds detected can be a very wide range of non-speech related sounds – such as squeaky bearing, jingling keys, breaking glass, water running, animal sounds, and so on.
- **Voice Commands** is a specific sub-set of audio patterns that are the recognition of a small set of spoken words. This is also referred to as keyword spotting. It can be used in a variety of use cases, such as waking up a voice service (such as Alexa) or using voice activation for smart home devices or appliances. A keyword spotting application may also support different languages.
- **Low Resolution Vision** uses a camera with resolution around 100x100 to detect objects for presence detection, people counting, video wake-up or more.

1.3 Embedded Application Development Workflow

Developing an application that incorporates ML as a feature requires two distinct workflows:

- The embedded application development workflow used to create a wireless application (with Simplicity Studio or your favorite IDE).
- The ML workflow used to create the ML feature that will be added to the embedded application.



This document assumes you are familiar with the standard embedded application development workflow. The following sections focus on ML models, how they work in embedded applications, and the key challenges for using them in that environment.

2 About ML Models

An ML model is an application-specific encapsulation of an algorithm to detect the patterns of interest. When running a model, sensor data is fed continuously, and the model will make predictions about how closely that data matches a pattern of interest.

A **neural network** is one of the most powerful ML methods that can be deployed on a small, embedded device. Inspired by the human brain, a neural network is a set of algorithms designed to recognize patterns. Neural networks interpret sensory data through a kind of machine perception, where they label or cluster raw input and mapping that input to the correct response. The patterns they recognize are numerical, contained in vectors, into which any data (such as image, sound, text, or time series) must be translated. Neural networks are described as having layers, an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.

Deep learning neural networks are distinguished from neural networks on the basis of their depth or number of hidden layers.

A **convolutional neural network** is a type of deep neural network based on applying multiple filters to the input, which is presented as an array of data (typically a 2D image), to learn information about the data. For example, audio from a microphone is converted into a 2D image (spectrogram) and then passed into a Convolutional 2D network for processing. Silicon Labs EFR32xG24 includes a 2D matrix vector multiplier hardware accelerator. As such, it is well suited to accelerate Conv2D networks by 4-10x when compared to software only.

There are different types of neural network model architectures. Two relevant ones are:

- Classification Model
- Autoencoder Model

The **classification model** incorporates labeled data (for example, 100 instances each of data labeled as doc, cat, and goat). It then processes input and generates a vector of probabilities. Each vector entry maps to one of the dataset "classes". So, if a classification model was trained to predict if an image is a dog, cat, or goat, then the output would be a 3-element vector.

- Element 0 -> probability of being dog
- Element 1 -> probability of being cat
- Element 2 -> probability of being goat

A Confusion matrix is an N x N matrix used for evaluating the performance of a Classification Model, where N is the number of target classes. The matrix compares the actual target values with those predicted by the ML model. The confusion matrix is used to determine the accuracy of the overall model, and ultimately will be an influence on how to interpret the inference data received.

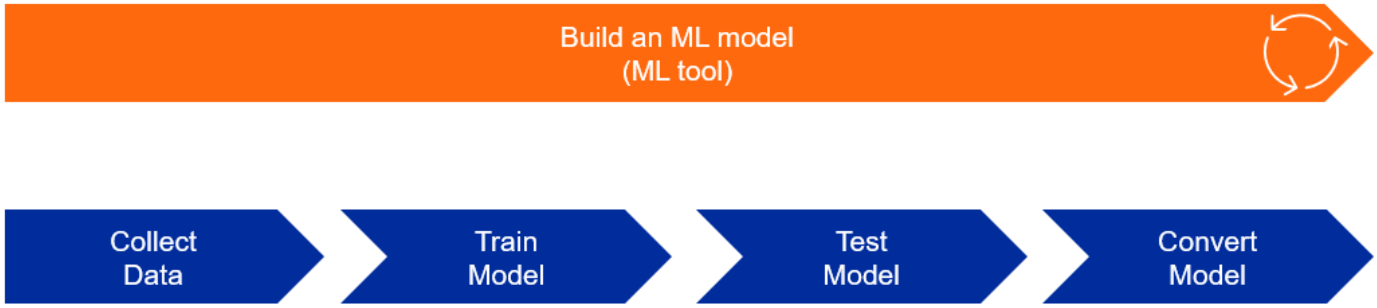
The **autoencoder model** takes unlabeled data and automatically detects the observable patterns. This is typically used for anomaly detection.

The above are just a few ML methods. ML methods are used depending on the application requirements. For example, **traditional machine learning** is a "rules-based" approach to machine-learning and is commonly used for tabulated data. Examples of traditional ML methods include:

- Random Forest
- Support Vector Machines (SVM)
- K-Nearest Neighbors (KNN)

3 Developing ML Models

As shown in the following figure, the ML model is the result of a series of steps that begin with a raw data set. These steps are discussed in the next sections.



3.1 Collect Data

The first step is to collect, clean, and classify the raw data into data sets, which are then used for training and testing models. The quality of the model will directly relate to the quality of the data set.

3.2 Train Model

Training is the step of ingesting the collected field data and developing the algorithm that will be able to predict the pattern of interest. ML developers will run through training multiple times to iteratively develop a production-quality model. Training occurs during the R&D development of a product, and typically occurs on a high-performance PC, or cloud servers. Training can take hours, or days to complete.

Two types of ML methods are relevant for embedded applications.

- Supervised learning
- Unsupervised learning

Supervised learning uses labeled data to train a model that can detect those labeled data patterns in a real-life environment, in other words a classification model. In the cat example above, the idea is that the brain was “trained” to understand a cat by feeding it cat images that were described, or “labeled”, as a cat. This method is used in most of the embedded application use cases. When determining if a use case is appropriate for ML, the first questions are 1) what labeled patterns are to be detected, and 2) is high quality data obtainable to represent those patterns.

Unsupervised learning uses unlabeled data without any explicit instructions of what to look for. Typically, large amounts of data are needed. This method will attempt to automatically find structure within the data by extracting useful features and analyzing its structure. There is no indication of what patterns to look for, the training learns how to identify the patterns on its own. The models are typically used to detect anomalies. During training, the algorithms evaluate a large amount of operational data to determine what normal operation looks like. In the field, the model is used to identify an anomaly to that operation, such as increased vibration from a wind turbine.

3.3 Test Model

Once training is complete, a separate test data set is used to evaluate the model’s performance. The test dataset contains data that has not been seen by the algorithm during the training phase. If the predictive power of a model is strong on the training dataset, but poor on the test dataset, then the model is too specific to the patterns from the training data and is considered to be overfit to the training dataset. That is, it has memorized patterns rather than learned a generalizable model. An underfit model, on the other hand, is one that performs poorly on both training and test datasets and has neither learned nor memorized the training dataset and still is not generalizable. An ideally fitted model is one that performs strongly on both datasets, suggesting it is generalizable (that is, it will perform well on other similar datasets).

3.4 Convert

Finally, the model needs to be converted or quantized into a format that is optimized for use on the embedded device. This reduces model size and improves CPU and hardware accelerator latency, with little degradation in model accuracy.

4 Deploying ML

When a device that incorporates ML as a feature is deployed in the field, the application takes input from the relevant sensor (for example, a microphone), extracts features (for example, a spectrogram), and analyzes it in a process called **inference**. Inference is the process of running live data points to an ML model to predict a classification. Additional post-processing (for example, thresholding and averaging) of the model output is usually required.

Inference cycle time is the time to run a single inference and is model-dependent. For embedded devices, inference can require 10s of millions of mathematical operations. Hardware accelerators, such as the one available on the EFR32xG24, can greatly improve the inference time as well as reduce the CPU overhead.

Depending on the resulting score, the application then generates a **trigger event**. For instance, using ML to detect the word “Quiet” would create a trigger event that would turn off a smoke alarm. A trigger event usually requires an algorithm that consumes ML inferences and generates the trigger. ML inference produces a single number and can occur as fast as every millisecond, up to a second. The algorithm consumes a series of inference results and delivers a single trigger.

One of the values of ML is to reduce the number of False Triggers, when compared with traditional approaches. For example, traditional glass break detectors may trigger on loud noises that are not from glass breaking. However, if the model is trained with a high-quality set of glass break data sets, a sensor using ML will produce fewer false triggers from loud noises that are not glass breaks.

Accuracy is the measure of how often the ML model will infer correctly. For instance, if a model is 90% accurate, on average the model will detect the pattern 9 out of 10 times. If the model is 100% accurate, it will detect the pattern 10 out of 10 times. Typically, the size of a model will increase with its accuracy. However, it is not linear. The size of a model increases geometrically, or exponentially with the accuracy of a model. There is always a need to balance accuracy and size. The goal is to get the best accuracy that fits within the size constraints of the embedded device. See the next section on Key Challenges for more information.

5 Key Challenges

Developers who are implementing ML as a feature in their embedded applications face four key challenges.

1. Acquiring a good data set.
2. Ensuring feature extraction is EXACTLY the same in training on a PC as it is running inference in the field.
3. Creating a model that fits within the fixed resources of the target device.
4. Developing the post-processing algorithm to interpret the results from the inference engine.

5.1 Acquiring a Good Data Set

Acquiring a good data set is essential to creating a good model. The quality of the model depends completely on the data provided for training the model. Some companies have spent millions of dollars and years acquiring the data used for a production-level ML model. Depending on the application, and the tools chosen, the challenge for developers can be different.

While the data set can be created from scratch, tools exist to help developers acquire an accurate dataset. These include a data capture, data ingestion feature that allows developers to gather data straight from a device such as the Thunderboard Sense 2 or EFR32xG24 Development Kit.

5.2 Ensuring the Feature Extraction is EXACTLY the Same

Feature extraction refers to the process of transforming raw data into input for the ML platform to analyze. Feature extraction is used both on the PC for training and on the embedded device for inference. It is very important that the feature extraction is the same on both the PC and the embedded device.

5.3 Creating a Model that Fits

RAM is a critical path resource in an embedded device. The amount of RAM available impacts the size and accuracy of the ML model. A Silicon Labs developer will typically create a wireless application and include ML as a feature. In this case the RAM is shared between the wireless application and the ML model. The embedded application RAM requirements depend on the communication protocol used, security requirements, application features, and so on. Understanding how much RAM is used by the embedded application allows the developer to determine the limits for the ML model.

ML development tools typically show how much RAM and flash is required by the model being developed. Therefore, the developer can design a model that fits within the constraints of device and the wireless application before building the final application. This allows the developer to iterate on a model design and know it will fit in the target embedded device.

A knowledge base article about estimating embedded application RAM usage may be found on the Silicon Labs Community site. Note that the RAM required to run the model is statically allocated. Therefore, once the model is built and compiled into the application, the static RAM allocated includes the model requirements. The remaining heap shows how much dynamic RAM space is available for the application during runtime.

5.4 Creating the Post-Processing Algorithm

This subject is related to interpreting the inference results from running the ML model. The ML developer needs to understand the model architecture being used, and how that architecture produces its inference results. The classification and autoencoder model architectures are further described in Chapter 2 About ML Models.

For the classification model, a single inference will produce an array of probabilities. The array is a list of all the classification types (or data labels), and the probability of the inference detected for each of those types. An inference cycle can run in 10s or 100s of milliseconds. The percentages will be recalculated each time and could be different for each inference. The developer needs to determine the algorithm of how to consume a steady stream of this data and translate that into the appropriate application trigger.

For the autoencoder model, a single inference may return a single number. This number would be the percentage similarity of the sensor data fed into the training. In the case of preventative maintenance, the majority of the sensor data would be of a normal operating machine. Running a single inference would return the percentage likelihood the sensors detected normal operation. It would be up to the developer to define when an appropriate application trigger should be presented.

6 For More Information

The following are some helpful resources:

- Starting point for Silicon Labs' approach to ML: https://www.silabs.com/applications/artificial-intelligence-machine-learning
- Selecting the best model development tool: https://docs.silabs.com/machine-learning/latest/machine-learning-overview/
- The tinyML Foundation: https://www.tinyml.org
- The differences between AI, ML, and deep learning: https://www.ibm.com/cloud/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks
- A good discussion on "models" and "algorithms" in ML: https://machinelearningmastery.com/difference-between-algorithm-and-model-in-machine-learning/
- Neural networks: https://wiki.pathmind.com/neural-network
- Convolutional neural networks: https://towardsdatascience.com/a-beginners-guide-to-convolutional-neural-networks-cnns-14649dbddce8
- The classification model confusion matrix: https://www.analyticsvidhya.com/blog/2020/04/confusion-matrix-machine-learning/
- Feature extraction: https://www.mathworks.com/discovery/feature-extraction.html
- Inference: https://hazelcast.com/glossary/machine-learning-inference/)
- TensorFlow Lite for Microcontrollers in the GSDK: https://docs.silabs.com/gecko-platform/latest/machine-learning/tensorflow/overview

Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support & Community
www.silabs.com/community

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

www.silabs.com