USING MACHINE LEARNING IN EMBEDDED SYSTEMS

Abstract

The future of embedded devices becoming more intelligent, reliable, and more connected. Adding Machine Learning to embedded edge devices is now easier than ever with new tools for system developers.

Introduction

Embedded processing systems surround us and go where we go. At home they can be in the coffeemaker, wash machine, refrigerator, thermostat, television, even in your toothbrush. In a factory, you can find them in motors, automated machines, robots, even the soda machine. In a vehicle, they control various items such as the windows, air conditioning, radio/infotainment system, collision detection, and rear-view mirror. Most of those devices perform relatively simple functions and are not connected to the Internet. But the future for these embedded devices is about becoming more intelligent and more connected as shown in Figure 1.

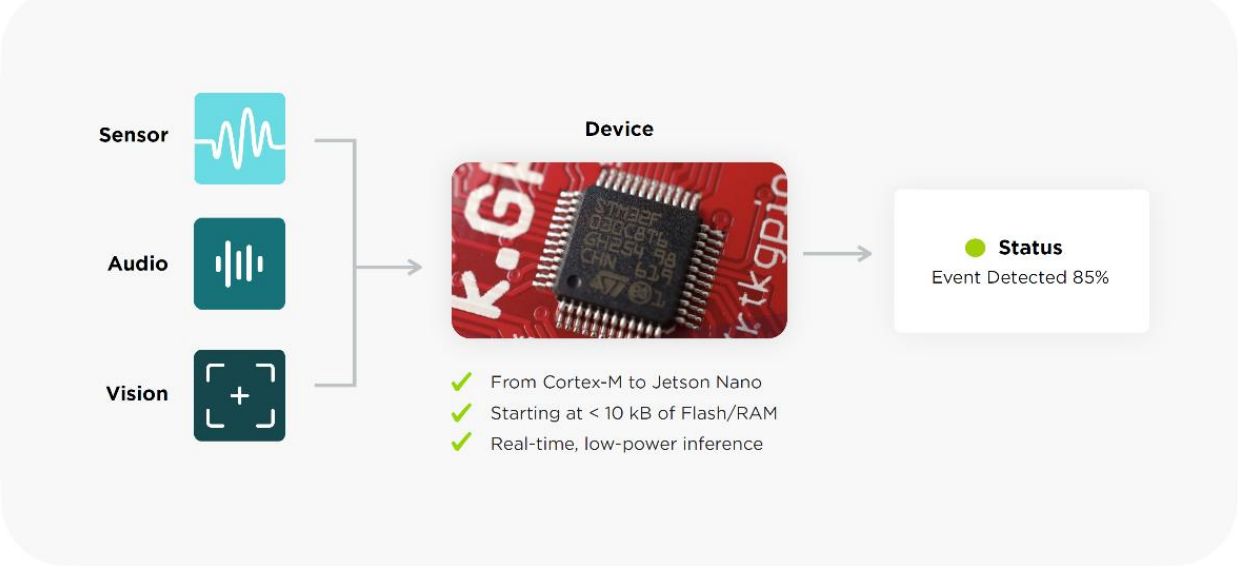

Figure 1. Adding Machine Learning to embedded edge devices make them smarter Source: Edge Impulse

Embedded System Design Constraints

By connecting embedded devices, we get the ability to orchestrate multiple devices and make them more responsive to our needs. The more intelligent embedded systems are, the better they perform their tasks and the better they can work with humans. They can also become more reliable – being able to detect problems earlier than before.

But often these embedded systems are constrained by cost, power, available memory, and/or compute performance. As such, it is always a challenge to get the most from each system within these bounds. In addition, there is much more that these systems can do, given the right resources. For example, most sensor data is discarded today because the embedded devices lack the required connectivity or intelligence to store or process the data. But with more edge intelligence, the raw sensor data can be processed locally, capturing useful information that can be communicated or acted on more economically.

But sometimes the sensor data (vibration, sound, images, video) is discarded because it is difficult to program a way to process and utilize the data. To utilize the sensor data, in the past, required developing algorithms (rule-based techniques) that could be applied to data to extract important information or patterns. More recently, the data is sent to the cloud for processing. But

sending too much data can swamp costly networks and can create privacy and security issues. Adding intelligence at the sensor or the edge can process sensor data locally. Rather than trying to program edge data processing algorithmically, a new technique using Machine Learning (ML) has proved to be simpler and cheaper to implement.

Even with local processing, connecting the sensor to the cloud provides important benefits. Connectivity allows important sensor data to be monitored for actions or drift. In addition, new models can be sent to the node for new parameters or compensate for drift. Connectivity can come in many forms from local mesh protocols to cellular to satellite.

In many cases, adding machine learning can also improve functionality, enhance system reliability, and reduce power consumption with minimal power and software impact. In addition, ML inference (applying a pre-trained model to real-time data) can be more effective than rules-based programming and with new tools, making it easier to implement. With ML at the edge, the embedded processor can better process sensor data and make decisions locally. Anomalies can be identified with ML and reported.

The use of ML can apply to a wide range of embedded systems running on extending from the smallest microcontrollers (also called MCUs and largely powered by Arm Cortex-M CPUs) to complex applications processors (often running on Arm Cortex-A CPUs with additional co-processors) as shown in Figure 2. There is a new class of powerful MCUs based on new Arm and RISC-V cores that can perform image processing as well. The more capable Cortex-A processors can also be combined with accelerators such as GPUs to handle object detection and video classification.

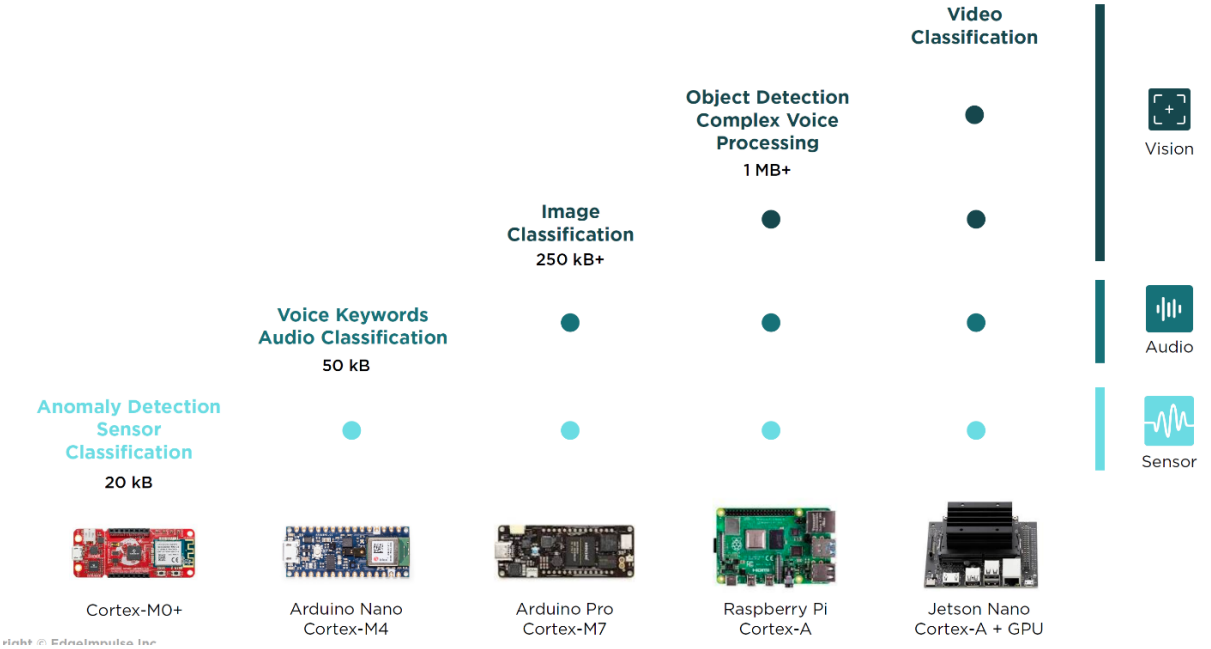

Figure 2. Typical development platforms and supported sensor types

Source: Edge Impulse

There is a large range of performance requirements and capabilities for ML. That said, ML is not magic dust that can be sprinkled on any system to make it run better, the neural network model

requirements must be matched to the processing performance. And an ideal platform would be able to scale from MCUs to application processor system-on-chips (SoCs). In fact, the entire platform should scale.

To use ML effectively, it requires quality data with expert labeling, suitable application and sensor types, and the right software stack and tool chain. The full control flow may still include filtering raw sensor data to eliminate noise. For video, feature extraction algorithms are also helpful to make ML more accurate.

Building and Maintaining Data Sets

With the right tools, the programmer can capture real life data to create custom datasets to be used to train the ML Neural Network (NN) so it learns the features that successfully match that pattern or deviate from the pattern. When real-time data is passed through the NN inference model, it can recognize the relevant feature (called classification) or detect data that falls outside the acceptable range, triggering an anomaly event. Then, the anomaly detection can be reported to the cloud or it can be used to trigger a local action (like shutting down a failing motor).

Acquiring the right data set can range from using a premade model to building a custom data set for a specific application. But premade open-source models may not be directly applicable to the embedded edge device, may not achieve the desired results, or the data set may have licensing requirements.

Many embedded designs have specific applications and sensor input data, such as a specific motor's vibration or a specific sound detection. New tools are available to be able to "roll your own" custom data models in a cost effect manner for edge embedded systems (Figure 3).

Figure 3. The data collection and model deployment for continual improvements. Source: Edge Impulse

In addition, by building a custom trained model, the systems designer can upgrade the model for continual improvements. Often equipment and sensors have drift and noise issues that need to be updated over time. Or new corner cases in the model are found. It is possible to collect additional data to partially upgrade the model and deploy it.

The range of data sets and processing requirements does vary as different tasks have a wide range in performance – usually associated the amount of data to be processed per second and the extent of pattern recognition tasks required. For example: a vibration sensor will record low resolution signals over long periods resulting in a low data rate while video cameras can steam many megabits of data per second.

Edge Impulse has tutorial that shows how to build a ML model with a simple microcontroller board with an accelerometer. The ML model is designed to detect specific movements (waves, up/down, etc.) by training the model with the actual board and sensor data, filtering processing the raw data to build the behavior mode and build up a process flow in the tools, and then deploying the finished software and ML model to the board. The tutorial can be seen in this video: https://docs.edgeimpulse.com/docs/continuous-motion-recognition

Machine Learning Performance

Various accelerators and coprocessors, such as DSPs (Digital Signal Processors), NPUs (Neural Processing Units), FPGAs (Field-Programmable Gate Arrays), and GPUs can make ML run faster to save power or increase capabilities by running the NN more efficiently than general purpose CPUs and MCUs. The trade off with using accelerators is that they add additional cost and complexity.

Even using ML NN processing, building a real system is often a mix of standard embedded systems programming, rules-based algorithmic signal conditioning, and neural net data filter and detection, and finally a user interface or automated data reporting. A fully integrated tool chain that can manage the entire control flow is still a way off. For now, developers will have to combine multiple workflows.

However, there are new tools that allow developers to build virtual models without production-ready hardware to test designs and develop detailed code. For complex systems, these are called digital twins, but for simple edge embedded systems, emulation either in the cloud or on a local development system is possible.

Most of the time, training is performed in servers in the cloud because training can be compute intensive and can involve large data sets to be processed. Feeding data directly from the device to the cloud allows real-time capture of signals that can be used to build ML models, that can them be deployed at scale to embedded devices. In the cloud, the system designer can visualize the sensor data and run multiple tests without having to run the actual hardware.

To get ML into the hands of a typical embedded systems designer requires a new set of tools. Several startups, including Edge Impulse, is trying to make the process of building ML into embedded systems accessible to a much wider range of developers with "low-code" tools.

The Edge Impulse tools will suggest a neural net model based on the application. It can also start with a default full-connected NN and then modify the model after training on local data. Some models work well for particular sensor types – like combinatorial neural nets (CNN) for audio.

It is also important to understand the throughput of the inference. The Edge Impulse tools can determine the required time for inference on a target platform. Then, the designer can stream real inputs in the cloud to test and visualize the system in the cloud as a digital twin. Using such a platform allows the system designer to build a custom data model that can be updated in the field. With the Edge Impulse tools, the system designer can update the model and redeploy in the field at scale.

Edge Impulse includes MLOps that will scale up and will also offer the ability to manage the ML solution for the entire life of the product. The company's "edge" is that it can build more compressed ML models that are easier and cheaper to deploy. But the real benefit is supporting the product over the entire life-cycle with continual improvements.

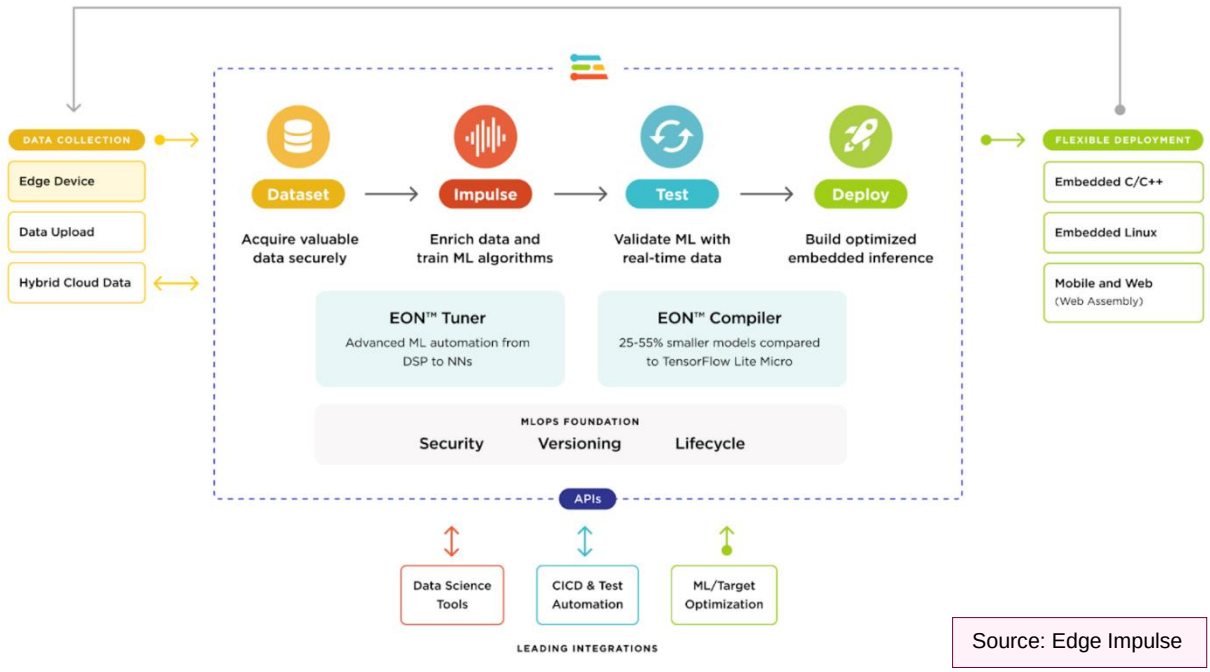

Figure 4. MLOps can make the continual model improvements and scale up to more complex platforms

Conclusion

Rather than trying to build and validate rules-based code through trial and error, ML can do the job and be built using a data-driven approach. Using a data driven NN approach, it is also possible to update the device behavior without extensive re-coding of the system just by updating the trained models in the field.

Getting ML into more embedded systems requires a more intelligent embedded software solution that can harness today's advances in sensors to match complex events on the device, resulting in new features, less bandwidth, and lower power.

The next wave of intelligent devices will power an intelligent and adaptable world. The hardware is still improving and will continue to improve with performance acceleration from vector extensions and DSP instructions. Bringing ML to embedded edge platforms requires new user friendly tools. Those tools and platforms are available now and embedded designs can begin adding ML today. It is important to get system developers exposed to what is possible today and get started.