Deploying AI and Machine Learning for the IoT

Edge Computing Applied to Auto and Industrial Applications

Markus Levy, NXP Semiconductors

While AI, neural networks, and classical machine learning algorithms have been around for decades, these technologies have improved 1000x in 3 years. In general, this paper begins by explaining the technical reasons that led to this improvement. This technology is being exploited by the rapid expansion of the IoT, industrial, and automotive market segments, where many industries realized that it was impractical and in violation of privacy to push all data up to the cloud. The author describes various methods for edge AI, which can be applied to vision, voice, and anomaly detection with sensors.

Introduction

AI has lived in the helm of the mathematicians for more than six decades before public interest suddenly increased. One reason for the current publicity is that, for a long time, considerations on AI applications were purely theoretical, or at least science fiction. For artificial intelligence use cases to become real in the present environment, three conditions had to be fulfilled:

1. A good representation of appearance and functionality of biological information processing
2. Very large real-world data sets
3. The capability to process these huge amounts of data in reasonable time

While many may still perceive AI as science fiction, AI as we know it today is real and represents a tremendous opportunity for significant and positive impact on our daily lives. While the cloud appears to be the major focus for AI technology, the edge is garnering a growing amount of intelligence, interfacing with the physical world – an opportunity that we 'Edge AI'. 'Under the hood' industry pundits like to associate AI with the biological representation of the human brain, when in fact the technology is still far from this realization. Nevertheless, with advanced training and learning techniques, AI even has advantages over the human brain, such as the ability to have virtually unlimited memory capacity.

Continuing with the conditions listed above, the other driving two factors that are driving this so-called Edge AI paradigm are the generation of virtually unlimited amounts of data and an increasing availability of computing resources, even down at the microcontroller level. Edge AI allows us to significantly decrease or even eliminate the latency of transferring data to the cloud and alleviate the growing privacy concerns. Advanced computing resources, with various forms of AI acceleration techniques, are an essential ingredient that shouldn't be taken for granted, but software is of the utmost importance for unlocking the real potential of AI. Therefore, given the availability of computing resources, the practical implementation of AI is made possible by optimized algorithms, software tools and frameworks that support these technologies.

Software is the Key to Machine Learning at the Edge

To achieve mainstream adoption across many application areas, Edge AI must be abstracted above the depths of its mathematical underpinnings. It must be simplified by cloud services where models can be trained, and inference engines can be developed & deployed with a easy-to-navigate web interface, rather than requiring the developer to create complex mathematical algorithms. There are many 'programming' options available or coming available, whereby programming for AI, often referred to as Software 2.0, is not about using traditional programming methods, instead AI programming is about utilizing neural networks and classical ML libraries that some entity [e.g. Google with TensorFlow]) has developed. In Software 2.0, the programming is about determining and optimizing weights and parameters (e.g. training models).

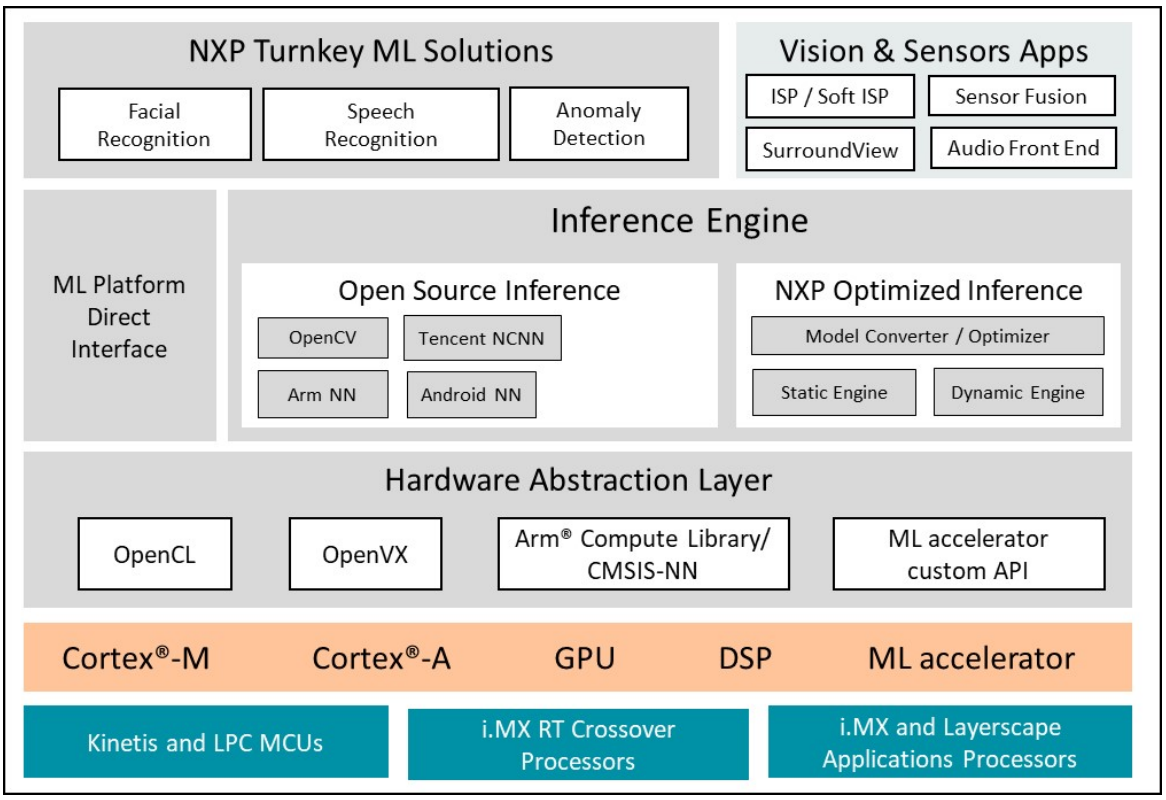

Figure 1. A sample toolkit for deploying AI in edge devices.

The growing number of options (most of which are open source) indicates that Edge AI is beginning to hit mainstream deployment. And software technologies for Edge AI are rapidly expanding and will continue to do so into the foreseeable future. We are witnessing this expansion in many forms including model frameworks, inference engines, neural network optimization techniques, convertor tools, and data augmentation methods (used for training).

Model Formats

TensorFlow leads the pack in terms of popularity and functionality (and has become a de facto standard), but it's baby brother, TensorFlow Lite, is gaining momentum, especially in mobile and edge environments. Today, there are even translation tools to convert TensorFlow models to the TensorFlow Lite format. Care must be taken when

converting from TensorFlow to TF Lite – as the latter does not support the entire list of operations, so some neural network architectures might not function properly.

There are conversion tools to support transition from most any other framework (e.g. MXNet, PyTorch, Caffe 2, Keras, and the list goes on. The convertor tools allow users to transition to their favorites, for example converting from TensorFlow to ONNX (Open Neural Net Exchange format) or NNEF (Neural Network Exchange Format), both of which are industry standard formats intended to reduce fragmentation.

Inference Engines

On the inference engine there are also a wide variety of open source options, depending on the target application. For example, for users working with Arm-based platforms (mobile or embedded), Arm NN parses and translates neural network frameworks into a high performance inference engine, utilizing Arm Compute Library (also open source) to pull in its optimized software functions (see Figure 2).

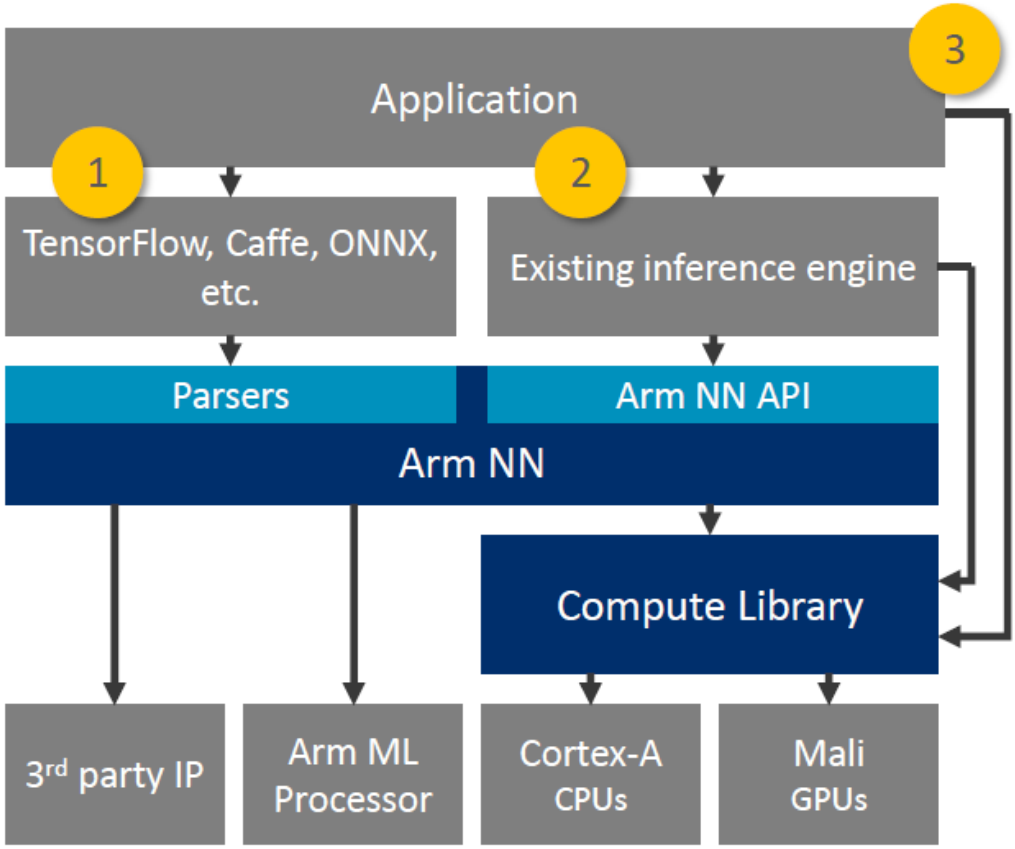

Figure 2. Arm NN offers 3 ways to deploy a trained neural network model.

The Arm NN has three ways to deploy a neural network model. One way is to is to connect to Arm NN through high level frameworks (e.g. TensorFlow); the software begins by parsing the model into a graph format, exposing the network operations which can be deployed via the Arm compute library (ACL). The user can also connect to existing inference engine, bringing the appropriate library functions from ACL as needed. Or the application can call the ACL directly; although this requires a bit more effort on behalf of the user.

Understanding Edge AI Hardware Tradeoffs

[move this section after software]

From a hardware standpoint, Edge AI hardware designs must recognize a three-axis tradeoff between cost, decision accuracy, and inference time. In any embedded design, all system designers must wrestle with the cost factor, it's inevitable. But in Edge AI design, we have the unique, interrelated properties of accuracy and inference time. Increased accuracy relates to the 'decision' that the system must make, and it implies the increased training time that is associated with the requirement for more data used in the training. Furthermore, the need for increased accuracy often means that even more complex AI models must be used – which boosts up cost because higher performing devices, more memory, and ultimately more energy are needed. Inference time (i.e. user experience) is the time required for the system to make the 'decision', and invariably the quicker the decision, the more performance is required.

Cost, accuracy, and inference time tradeoffs are tied to the AI application. For example, in an application that performs pet recognition coming through a door, an inference time in the range of 200-500 milliseconds is far within the acceptable range, easily handled by a high-performance microcontroller. A doorbell security system, an application that has become extremely popular, can suffice with inference times on the same order of magnitude as the microwave. In this application, a camera captures a person's face and in less than a second, the Edge AI system recognizes that person as friend or foe (or unknown). However, in this case accuracy is extremely critical because you wouldn't want to be locked out of your own house or worse, let a foe enter. In addition to requiring larger memory capacity, increased accuracy and more classifications (e.g. food items, faces) to 'decide' on could also yield an increased number of computations, thereby requiring a higher performance applications processor to meet the acceptable inference times. The security doorbell represents a good example where an OEM might have low- and high-end products, varying by the capability to recognize a specific number of faces within the acceptable time window.

In some life and death situations, Edge AI will have much stricter requirements (e.g. applications such as autonomous driving) that demands many accurate decisions to be simultaneously made every second. Another Edge AI application example is the monitoring of drivers' reactions through their eyes – a real-time function that requires high performance computing available from an applications processor. In short, Edge AI applications have widely varying computing performance requirements, which can be satisfied by a wide range of processing devices from MCUs to high-end application processors. But the bottom line is that software is the key that unlocks the door for machine learning at the edge, and this will continue to evolve for the foreseeable future.

Many More Options and We Just Got Started

This paper provided a very high-level perspective on what's needed to deploy AI in edge devices. It's not that it's so complicated, it's really that the options are expanding daily. We will always need the mathematical experts to manage the complex functions of neural networks - for example, craftily implementing advanced neural network optimizations or creating better and faster ways of accurately training models. Embedded system developers building Edge AI products need Software 2.0

tools to provide a comprehensive ML development environment (e.g. NXP eIQ) and such environments must be specifically targeted, not just at the computational units (e.g. processor cores, AI accelerators), but also at the SoC architecture level, to provide the most efficient implementations. We're in for exciting times, as more and more developers realize that machine learning capabilities can add significant value to their products.

Author

Markus Levy joined NXP in 2017 as the Director of Enabling Technologies. In this position, he is primarily focused on the strategy and development of AI and Machine Learning capabilities for NXP's microcontroller and i.MX product lines. Markus is also Chairman of the Board of EEMBC, which he founded and ran as the President since April 1997. Mr. Levy is also President of the Multicore Association, which he co-founded in 2005.

Contact

Email: markus.levy@nxp.com