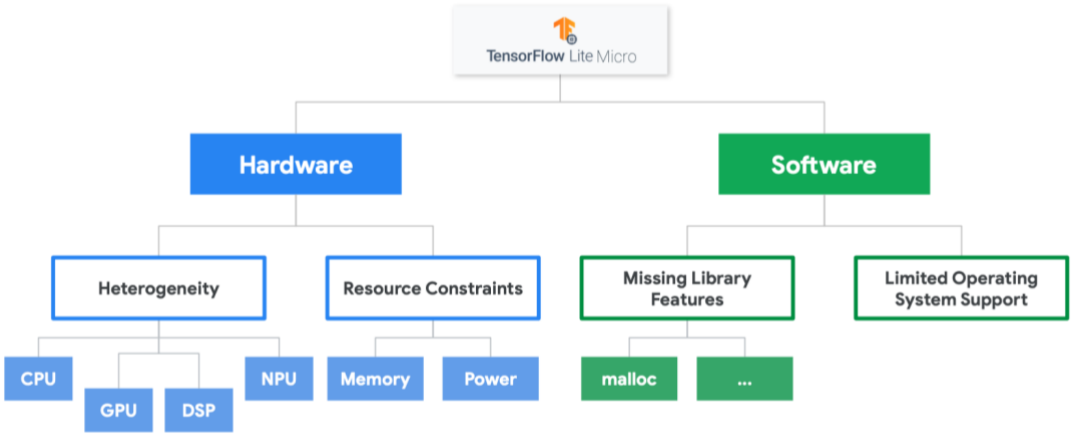
TFLite Micro: The Big Picture

An introduction to Google's Open-Source Machine Learning Framework for Embedded Systems
By Pete Warden Google

What is TensorFlow Lite Micro?

It's a software library that aims to make it easy to run machine learning models efficiently on embedded hardware. To understand what that means in more depth, I'd like to show the constraints of the embedded platforms it has to run on so hopefully, the design decisions make a bit more sense.



There are two broad categories of constraints that drive the library's design, HW and SW. Embedded systems face extremely tough requirements for power usage, price, form factor, and compute performance.

HARDWARE

These requirements mean that there are massively diverse sets of solutions available because different products require radically different trade-offs. The pressure to meet different trade-offs shows up in the hardware that's available.

There are a lot of different embedded chip architectures in active use, far more than in any other computing domain. The cloud has x86 CPUs, GPUs from a couple of vendors, and recently ARM chips have been emerging. Smartphones almost exclusively run on ARM Cortex A chips, but by contrast, embedded systems have ARM Cortex M microcontrollers, DSPs, Qualcomm DSPs, Synopsis microcontrollers, FPGAs, and with many other architectures like Texas Instruments dominating in particular markets.

These are more than just manufacturers. Most of the chips have entirely unique instruction sets that require their own compilers and tool chains.

The instruction sets are often incompatible even between different versions of a chip in the same family from a single manufacturer. This variety is because there are so many different trade-offs that make sense for different situations.

So many different solutions exist for all the various types of problems that embedded systems try to solve. A consequence of this is that the framework has to be radically portable, able to compile and run with no hitches across a wide range of computer architectures.

At the same time, compute latency is often an important requirement, too, so there has to be an easy path to optimizing the code for the particular features offered by a platform.

Another consequence of the tough requirements embedded systems face is that they often have comparatively small amounts of memory and low clock rates. These memory constraints make library design particularly hard because, in an ideal world, you want a framework that can be a closed box with an abstraction layer where you don't have to worry about the internals, but, because the implementation details of the library directly affect its binary footprint, how much memory the code takes up, users of it do have to be aware of how it works.

It also means that the library code has to be very careful about how much memory it uses for other purposes, too, with a focus on memory minimization algorithms.

SOFTWARE

The constraints of embedded requirements also extend to the system software that's available. If you use the more relaxed environments, the lack of a heap and dynamic memory allocation on embedded systems will come as shock.

Dynamic memory allocation using malloc is a productivity-enhancing feature of most modern computing environments, but it implicitly assumes that there is a large enough pool of memory available, possibly through virtual memory on disk, that running out of it is unlikely in normal operation.

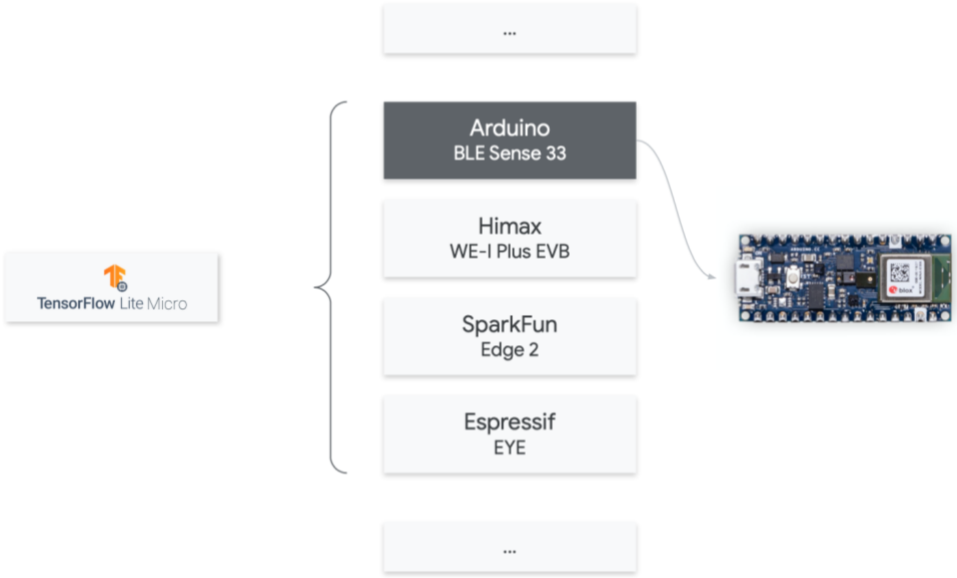
Because embedded systems may only have tens of kilobytes of memory and may need to run continuously for months or years, developers have to be a lot more careful and explicit about the allocation strategy.

TensorFlow Lite Micro avoids using any dynamic memory allocation internally and just asks for an **arena of contiguous memory** from the application to use for its variables, which is not altered after initialization time so that products can have confidence that the framework won't crash due to heat fragmentation when running for long periods.

Embedded systems also often lack other facilities we take for granted, like file system calls or even functions like printf that we might expect to always be available. To cope with this, TensorFlow Lite Micro takes pains to avoid using any C or C+ library calls that require operating system support.

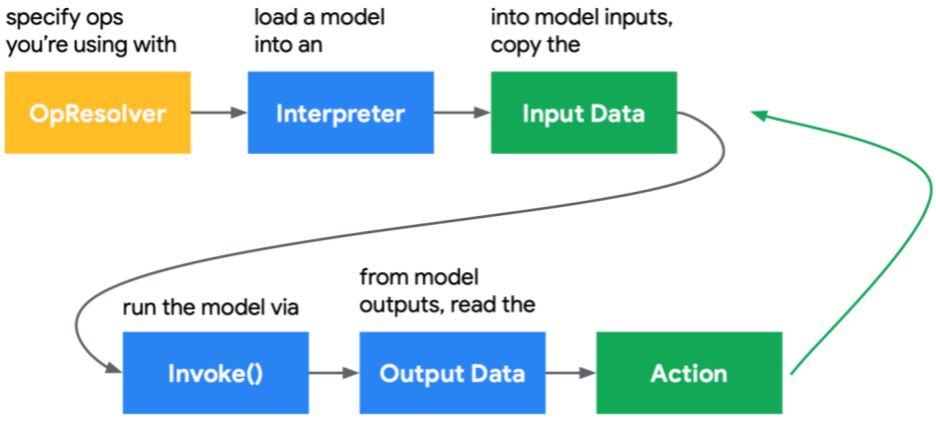
This includes features like the STL containers that rely on dynamic memory allocation as well as printf and other POSIX functions that need low level support. Even where these are available, they often require additional space in the binary so they're avoided for size reasons, too.

The end result of all this work is a software library that's able to run across a wide variety of different boards.



We're using the Arduino Nano BLE Sense 33 primarily in this course, but the same application code you're using to ML models should work portably and performantly across a wide range of other devices.

How do You use TFL Micro?



Hopefully I've given you a bit of an idea of what TensorFlow Lite Micro is, but how can you use it yourself as a developer? On the basic flow as shown here, you can see the same pattern in all of our code examples.

The first stage is specifying what ops you're using in your model
using an **OpResolver**. This is important because it ensures only the implementations of those ops will be included in the binary, keeping the space used as small as possible.

Then you load the model into an **Interpreter** object, which handles all of the initialization needed. You copy the **Input Data** you want to feed into the model into the model inputs.

You call **Invoke** on the interpreter to run the MODEL running the input data through all of the operations to calculate the final result. You then read that final result from the **model outputs**, and presumably, you're going to be taking some **Action** based on the model outputs.

You're using the ML Model to decide something, so your application should respond to what's being calculated. And then usually, you'll go right back to reading some **new input data** and running the cycle again.

This has been a very quick overview of a very complicated topic, but I hope it's at least giving you a flavor of what TensorFlow Lite Micro is all about.

Recap

To sum up, it's a library that's able to take in models from the TensorFlow training environment and run them on embedded systems with very little memory and no operating system facilities.

To work well in these environments, it focuses on memory size optimization, avoiding dynamic memory allocation, using a very minimal set of standard library functions, and keeping its code as portable as possible.