Summary of TinyMLaaS

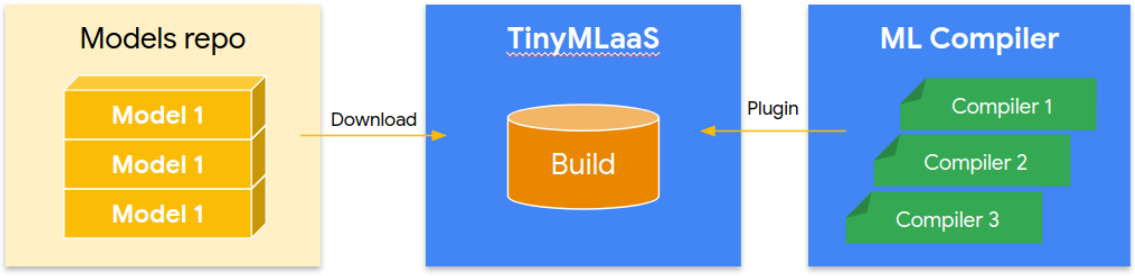
About This Reading

We have spent the last few videos discussing the TinyML-as-a-service (TinyMLaaS) cloud paradigm. This is an emerging technology that has the potential to overcome many barriers that exist today in terms of managing interoperability of ML software across heterogeneous microcontroller units (MCUs; the unit in a resource-constrained IoT device that manages program execution). While still in its early stages of development, it is an example of the coupling of TinyML within the cloud ecosystem and will likely become a key technology used by TinyML engineers as IoT devices become more functional and ubiquitous. In this reading, we will recap the architecture of TinyMLaaS and some of its associated challenges.

TinyMLaaS Components

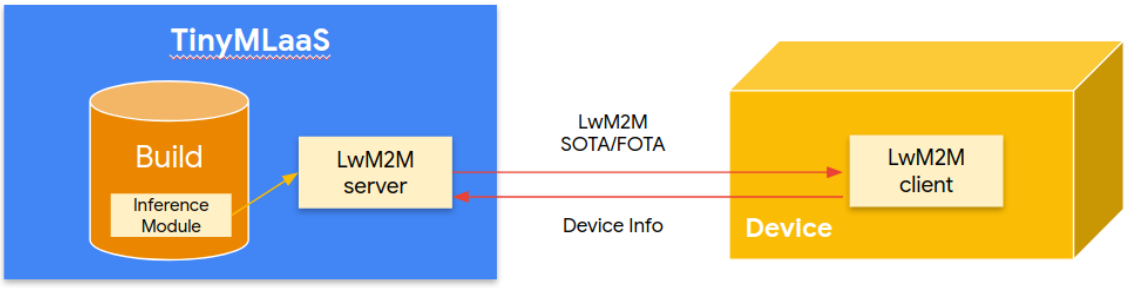
There are currently three main components to a TinyMLaaS ecosystem: (1) a compiler plugin interface, (2) an orchestration protocol, and (3) an inference module.

The TinyMLaaS runs as a piece of middleware in the cloud (or edge) and interfaces with both the cloud ecosystem (the TinyMLaaS back-end) and the embedded device (the TinyMLaaS front-end). The process is initially kickstarted when the embedded device sends a request to the TinyMLaaS client running on the cloud, communicating key information about its hardware constraints such as its available RAM, flash memory, CPU characteristics, and peripherals. The TinyMLaaS software then communicates this information to the cloud, which determines the appropriate model to extract from the model registry, and also the appropriate ML compiler and optimization that will correctly configure the model and its associated runtime to work on the embedded device. The compiler plugin interface is the component that creates the correct build for our specific embedded device by combining the appropriate model, compiler, and optimizations.

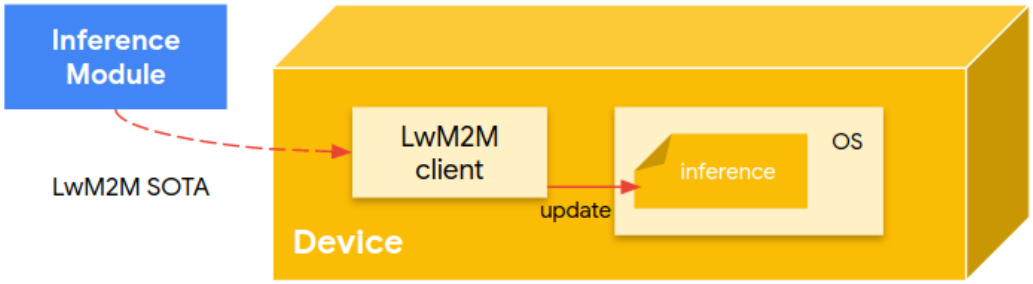


After the build is configured, we need to put it on our device somehow. We also need a way to actually interface with the embedded device, otherwise we would not be able to receive the device information in the first place! This requires some kind of orchestration protocol, with some kind of client software running on the embedded device. Typically, the OMA Lightweight machine-to-machine (LwM2M) protocol is used for this purpose. This protocol has a LwM2M

client running on the embedded device, and an LwM2M server running on the TinyMLaaS middleware. This protocol not only allows the device to send information about itself to the middleware, but allows us to send software/firmware updates over-the-air (SOTA/FOTA). This is how we are able to communicate our model build with the embedded device.



Are we done? Not yet. While we have communicated the new build to our embedded device, we have not made the ML inference application available to the device. In order to do this, we also require an inference module, which interacts with the LwM2M client to essentially update the system. We can think of it kind of like an installation manager for our device, like when we try to run a new program on our laptop or desktop computer.



Challenges

This technology is quite powerful, as it removes a lot of the friction in setting up a model on a new device. However, as we have discussed, there are some challenges associated with TinyMLaaS.

Perhaps the most notable challenge to TinyMLaaS is security and privacy. One of the benefits of TinyML is that all of the data is kept on-device, with no communication to a centralized system. This approach minimizes the potential for man-in-the-middle attacks or for a bad actor to hack into our system. However, by setting up a communication protocol between these devices and the cloud, we reintroduce those concerns and stimulate the need for new lightweight security protocols to keep these devices secure. In the cybersecurity world, IoT security is one of the most pressing concerns of the field given their rapid expansion in recent years. Some other challenges include balancing the trade-off between ML performance and device resources, and managing distributed execution.