DEPLOYING TINYML

HarvardX

Table of Contents

About Course

Objectives

The goal of this course is to teach learners how to engineer end-to-end tinyML applications using TensorFlow Micro. We teach learners how to program in TF Micro, and use it to deploy real-world applications.

Prerequisites

- Applications of TinyML (Course 2)
- Basic programming in C++
- TinyML course kit
    - *March 2022 update: Due to the ongoing COVID-19 crisis and the resulting semiconductor shortage, the microcontroller used by the kit (and therefore the kit) is out of stock globally. We've been working with Arduino on this and are expecting kits to be in stock in the coming months. We will continue to update this page and email learners with any updates when we receive them.* ***In the meantime, you are still able to complete the course without the kit.*** *While you will not be able to complete the hands-on exercises that are described in the course materials, you can still complete the course assessments, which are focused on your conceptual understanding of the course materials based on the readings, videos, and other instructional content in the course. If you have any additional questions or concerns, please feel free to reach out to edX support via email at support@edx.org.*

Learning Objectives/Focus:

- Discover the rich landscape of embedded ML applications
- Understand how to design and deploy end-to-end TinyML applications
- Learn to deploy TinyML models using TFLite Micro on embedded systems

1.1: Welcome to Deploying TinyML

Who's Who in TinyML3?!

Hi, there!

Welcome to Course 3 of the TinyML EdX series! This is where the pedal meets the metal!

Course 1 ("Fundamentals of TinyML") laid out the building blocks and established the language for tiny machine learning. Course 2 ("Applications of TinyML") introduced you to several use cases of TinyML in the real world and taught you how to train tiny machine learning models. Along the way, you also learned about many core concepts including quantization and model conversion. In Course 3 ("Deploying TinyML"), we are going to close the loop and talk about deploying ML models onto microcontrollers (MCUs).

Pete Warden and I (along with the help of our staff) will be taking you through the journey of deploying end-to-end TinyML applications on a Nordic Semiconductor nRF52840 MCU using TensorFlow Lite for Microcontrollers (TFLM). The MCU has an ARM Cortex-M4 32-bit processor with FPU, 64 MHz with 1 MB flash, and 256 kB RAM. It has a range of diverse applications for the Internet of things (IoT). This MCU is seated on your Arduino Nano 33 BLE sense development board, which aggregates a wide variety of sensors onboard. Combining those on-board sensors with the OV7675 camera module, you'll be able to deploy many of the TinyML applications you've seen previously along with new ones!

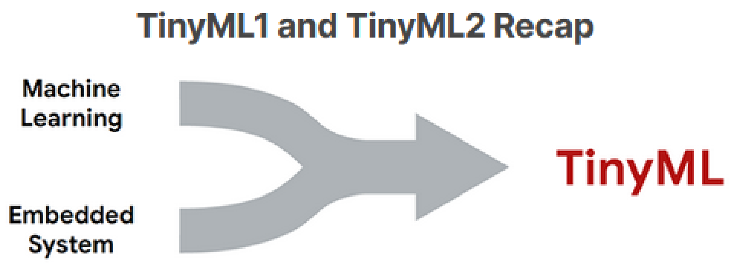
Long story short, our mission in this course is simple: we want to enable you to develop and build end-to-end TinyML applications using TensorFlow Lite for Microcontrollers.

Sincerely,

Vijay

03_TinyML_C03_04-01-03_final-en.mp4 (Command Line)

Course 1 & 2 Recap



Over the first two courses, you have been exposed to various facets of the machine learning ecosystem. You have seen how models are developed using the machine learning workflow, from the data collection phase through to the training and deployment phases. You have seen how models are produced using Tensorflow and then ported into a lighter-weight framework such as Tensorflow Lite. You have also seen how these models can interact with various forms of input data, such as time-series data from sensors and microphones, to images from an onboard camera. Throughout both courseswe also made sure to keep in mind the core principles of responsible AI. We talked you through how to design products with all users in mind. We discussed various error types that (can) occur when we are not careful in designing and developing AI models. We explored these concepts using various real-world case studies. We then considered how to ensure that models and datasets are designed with the goals of maximizing fairness and minimizing bias in mind. Finally, we explored Google's What-If Tool which allowed us to explore and analyze fairness and bias issues in real datasets.

In this course, you will go one step further still, and learn to deploy TinyML models responsibly on your own embedded system. Let's first revise some of the key concepts in more detail.

Course 1: Fundamentals of TinyML

Before we could understand TinyML, we needed to understand its underpinnings, namely, machine learning. The utility of TinyML only becomes apparent once we understand the machine learning workflow, the different supervised learning algorithms, how they are structured, and the tasks they are able to solve (and what data they act on). To this end, the first course provided a crash course in these topics, showing that models can be trained on existing data, allowing the model to learn important associations between its parameters and the data. These trained models could then be exposed to new information, and make predictions about the data based on these associations. The two main tasks that these models perform are called regression and classification. Regression algorithms try to approximate the mapping function (f) from the input variables (x) to numerical or continuous output variables (y), whereas classification algorithms try to approximate the mapping function from the input variables to different discrete or categorical output variables.

Machine Learning

We have seen that machine learning entails the development of data-driven models to make predictions or decisions for a particular task. Supervised learning takes labeled data, meaning that both the input and output variables are known, and attempts to produce a model that can manipulate the input variables in such a way that the output variable can be effectively predicted.

Depending on the curated dataset, there may be a large number of input variables or very few; there may be very many data points for us to train our model with or only a small number; we may need to preprocess the data in order to extract more relevant feature information, known as **feature engineering**, or simply leave the input data unperturbed. Thus, the first important steps are **dataset collection** and **dataset preprocessing**. The collection stage is often the most time-consuming unless an existing dataset is available. Determining how much data is necessary to perform a robust analysis, the best method to obtain the data (e.g., crowd-sourcing, external data providers, simulated data), as well as the number of features and their relevance, are challenging steps that will often require multiple iterations to get right.

Once our dataset is curated and preprocessed, we are ready to begin developing and training machine learning models. The training process is where a model tunes its parameters based on associations learned from a particular dataset. Different tasks lend themselves better to certain machine learning methods. For example, convolutional neural networks often work best for image data - a standard fully connected neural network can also be used, but will likely perform worse at the same task. Usually, models are selected based purely on their performance, but other metrics may also be important, such as interpretability and model complexity.

Different machine learning algorithms have different **hyperparameters**, pre-specified values which characterize the configuration of the algorithm. During the training of a machine learning model, often through the use of a **stochastic gradient descent** algorithm on a chosen **loss function**, these parameters are often altered to find the values that lead to the highest-performing model. This process is known as **hyperparameter optimization**. Some platforms, known as AutoML tools, have **automated** this activity, making it easier for the user to optimize their models during the training process (e.g., Google's Cloud AutoML). The optimization process performs the **model evaluation**, which assesses the performance of a model using metrics of interest, such as false-positive rate and F1 score. Cross-validation is often used to improve the robustness of the algorithm by preventing overfitting from occurring.

Recall that once our model has been suitably trained and optimized, we test our model on a hold-out dataset known as the test set. This simulates an unseen dataset, like our model would see in the production environment. If our model works on this set of data, it should work in our production environment! After a positive result here, we are then ready to deploy our model, which, depending on the application, may involve integration into existing infrastructure. Models are often **monitored** after deployment to ensure that they are functioning as expected.

Embedded Systems

The main embedded system used in this course is the Arduino Nano 33 BLE Sense, which is a relatively small microcontroller equipped with a 64 MHz processor, 1 MB of flash memory, and 256 KB of SRAM. While this system is highly resource-constrained, it is low cost and can be purchased for about $30. Clearly, this system is very different from the device you are using to view this reading. Laptops and smartphones have processors in the range of GHz, which is hundreds, if not thousands of times faster than the Arduino Nano. Furthermore, the Nano does not have any input or output peripherals that we are mostly accustomed to, such as a trackpad, keyboard, and screen. Finally, the small amounts of program memory and RAM available mean that the system has very little capacity to perform complex tasks. In fact, our particular embedded system is designed to only run one task at a time, as compared to your laptop which probably currently has multiple tabs of Chrome open!

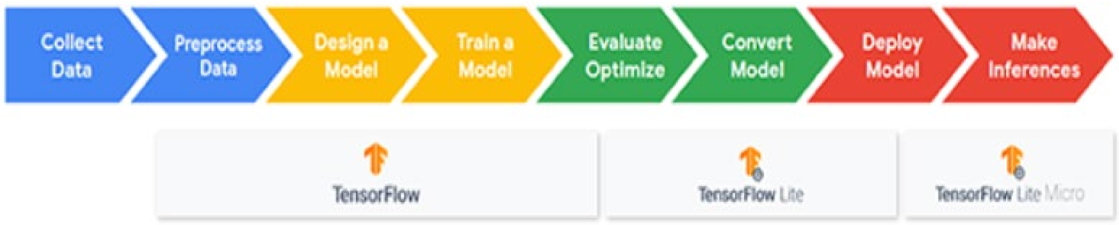
However, we have seen that there are multiple benefits despite the severe resource constraints. Since there is no (or a very limited and lightweight) operating system, we have minimal **computational overhead**, which improves inference speed (i.e., how many times our model can run per second) and reduces system **latency** (i.e., communication delay between sending and receiving data).

Privacy is another important benefit. Embedded systems that can perform machine learning do not need to communicate data, which reduces network loads and also prevents the possibility of man-in-the-middle attacks of transmitted data. Perhaps more important, only localized data is stored on the devices, meaning that there is no risk of data being stolen from a central repository.

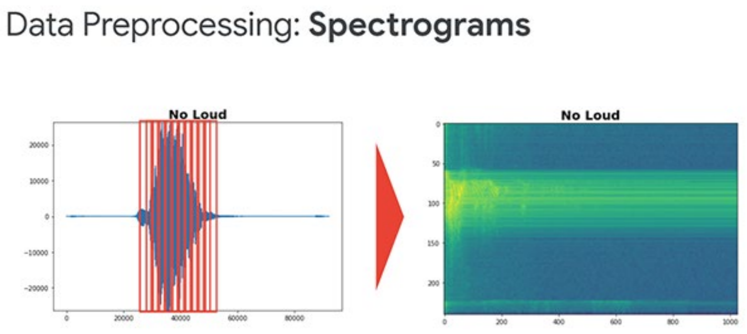
Perhaps the greatest benefit of embedded systems is their small **power consumption**. Modern devices such as laptops and smartphones consume a great deal of power, and running large machine learning algorithms often has a high computational cost. In comparison, embedded systems use very little power, typically on the order of mW. This is no small feat since such devices can run on a simple coin cell battery for an entire year without needing to be recharged - just imagine not having to charge your laptop for a year!

Course 2: Applications of TinyML

After learning the ropes of machine learning and motivating the utility of performing machine learning on embedded systems, we started to look at some archetypal examples of TinyML applications and some of the specifics of how models are ported from frameworks such as TensorFlow to embedded systems. We studied all of this in the context of the machine learning flow moving from collecting and preprocessing data to designing, training, evaluating, and converting models. We'll cover deployment in this course!



The first example we looked at was **keyword spotting**, which involved extracting the presence of specific keywords from a short voice recording. This example already exists in smartphones such as with Apple's "Hey Siri" and Google's "OK Google". We performed **feature extraction** of the voice recording by using spectrograms, which were then used to train our model.



Following feature extraction, we performed **post-training quantization** of model weights and inference calculations to allow our model to run using the 8-bit arithmetic available on most embedded systems[1]. In the penultimate stage, we took the quantized model and converted it to a more suitable file format, which, in this case, was a TFLite model. After converting this model to a binarized format, the model was then ready to be deployed in an embedded system.

We then went on to look at further examples, such as **visual wake words**. In the visual wake words application, we trained a model to determine the presence of a person in an image and saw how transfer learning could be used to retrain models for similarly related applications, such as detecting the presence of masks on a person's face.

In the final parts of Course 2, we looked at more advanced industry-oriented applications that began to move towards unsupervised learning algorithms, such as **anomaly detection**.

We have already done a considerable amount in these first two courses. In Course 3, we will go one step further still, and provide you with all the tools necessary to develop, build, and troubleshoot your own TinyML systems. Let's jump in!

---

[1]We must also note that we explored how **Quantization Aware Training** can reduce the errors introduced by quantization by allowing the model to adjust the weights to the quantization scheme during training.

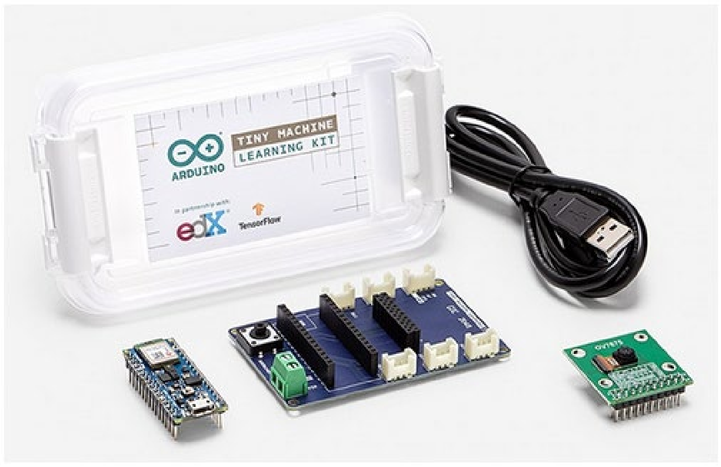
TinyML Application Deployment Preview

03_TinyML_C03_04-01-06_final-en.mp4 (Command Line)

The TinyML Kit

If you haven't already ordered your tinyML Kit, it is [available from Arduino for $49.99 at this link](). The kit is the easiest and most reliable way to obtain all of the parts necessary for this course. If you already have your kit or are planning on purchasing one then you can move onto the next reading.

That said, we understand that for some learners it may be beneficial, or even necessary, to obtain some or all of these parts via other means. As such, the remainder of this document details a bill of materials for the components you need to complete all of the planned exercises alongside potential vendors and functional alternatives. Note that availability may depend on your location and other shipping dynamics limitations.



tinyML Kit BOM

In the picture above you can see all of the components included in the purpose-designed kit that we have developed with Arduino. Fortunately, there aren't too many parts, to begin with, and most of them are generally accessible on a global scale, with the exception of Arduino's Tiny Machine Learning Shield, or PCB cradle (in the middle), for the Nano 33 BLE Sense AI-enabled microcontroller board (on the left) that serves to breakout the MCU's IO to connectors that permit easy, reliable connections with the camera module (on the right), as well as other sensing modules via connectors included preemptively for expansion to new areas of application, calling upon the Grove system for nearly plug-and-play compatibility with a range of transducers, available from SeeedStudio[1]. Fortunately, while the shield can certainly make your life easier and lends a hand in creating reliable connections with off-board sensors/actuators, we can readily replace its function with other off-the-shelf components (e.g., a breadboard and some wires as listed below).

The table below is a list of parts that are comparable with suggested vendors alongside alternatives. To be clear, the edX course staff will only support the official kit we have developed in partnership with Arduino.

Description	Part Number	Vendors	Alternatives
Nano 33 BLE Sense with headers See note (1)	ABX00035	Arduino, Amazon, DigiKey, Newark, RS Components, Arrow, Mouser, Verical, Farnell, element14, Arrow (cn)	NA
USB microB cable, often to type A or C	Varied	DigiKey, Amazon, and many others	See note (2)
Breadboard	Varied	DigiKey, Amazon, and many others	See note (3)
Jumper wires'Male to female'	Varied	DigiKey, Amazon, and many others	See note (4)
Camera sensor with breakout PCB	OV7675	RobotShop, Botland, Uctronics, and others	OV7670 See note (5)

Notes

1. Make sure that you select the variant of the Nano 33 BLE Sense *with headers*, which means that the board will arrive with 0.1" pitch pins pre-soldered, enabling you to quickly seat the microcontroller board into a breadboard without additional equipment.
2. The USB cable specification need only be *data carrying* **and** with connector type micro-B (plug) on one end — the other end will depend on the port available on your computer. Type A is the most

common connector type, whereas modern laptops are increasingly featuring Type C connectors. Note that not all USB cables support data exchange and some are intended for power delivery only. Be sure to select a cable with a description that explicitly mentions data transfer or equivalent.

- Type C cables tend to be more expensive than Type A, so it may be comparable or even cheaper to source a USB A to C adaptor if your computer only features a Type C port. These adaptors are generally available.
- For both cables and adaptors, as applicable (see 1a), think carefully about the necessary designation of plug vs. receptacle on each end. For cables, you'll likely want plugs on both ends. For a typical USB A to C adaptor, you'll find a Type A receptacle that gives way to a Type C plug, to be inserted into your PC.

3. Be sure that the breadboard you purchase is both *solderless* and equal to or greater in spatial dimensions than the standard 'half' size (5.5 x 8.5 cm), which we generally recommend for use in this context.
4. Here, 'jumper wires' refer to bundles of adjoined or individual lengths of wire that terminate in one of two possible forms: a 'male' pin (M) or a 'female' receptacle (F). To connect the camera module to our microcontroller board via the breadboard, you'll want to pick up about **25 or more** (individual wires) of the M-F variety. If you intend to go on to connect other off-board sensor modules, it may be helpful to have M-M jumpers on hand. Jumper wires can be purchased in various lengths. We suggest between 3 to 6 inches, here.
5. In searching for the OV7675 camera module, you may come across vendors selling the sensor itself, with only a ribbon cable breakout. Be sure to pick up a PCB breakout module that terminates in a 2x20 array of pins, like in the suggested link. If for some reason, you can't obtain the OV7675 module, we have previously had success in using the related OV7670 module, but note, as above, that the edX course staff will only officially support the official kit we have developed in partnership with Arduino, including the requisite code for the OV7675 modules. Any adjustments to said code to call on the OV7670 module technically fall outside of the officially supported scope. With that said, we have noted in the course the required changes (fortunately there are very few) as we recognize that many students are finding it hard to source the OV7675.

---

[1]You'll want to check module voltage requirements and IO diagrams to verify Grove compatibility.

TinyML Course Kit Overview

03_TinyML_C03_04-01-08_final-en.mp4 (Command Line)

How the Course is Structured

03_TinyML_C03_04-01-10_final-en.mp4 (Command Line)

1.2: Getting Started

Introduction to the Lab Sections

03_TinyML_C03_04-02-01-en.mp4 (Command Line)

C++ for Python Users

If you are comfortable with C/C++, please feel free to move on to the next reading. If you are new to C++, we hope this introductory material will be helpful for you. If you are comfortable with C/C++, please feel free to move on to the next reading. If you are new to C++, we hope this introductory material will be helpful for you.

Python, the language you have been using in all of your Colabs, is a dynamically-typed, “high-level” language that is interpreted at runtime. C++ (also written as Cpp), on the other hand, is a statically-typed, “low-level” language that is pre-compiled before running, allowing for very compact code.

The good news is that since we are using the Arduino platform, we won’t have to deal with much of the complications of C++ as that is taken care of for us by the many libraries and board files we will be able to leverage (more on that soon). We’ll then just have to pay attention to the changes in syntax which are more cosmetic than functional. That is, all of the main loops and conditional statements (e.g., for, if-else) remain the same functionally!

In order to make sure you feel comfortable we’ve put together a set of short appendix items and have collected some other good resources to walk you through the main changes between Python and C++ for Arudino which you can find below. As you go through the course, if you have good ideas for other introductory material, changes to the current material, or additional resources, please let us know and we’d be very excited to add them to the list for future learners!

For more information, take a look at our short appendix document that covers:

- Data Types
- Scope, Parentheses, and Semicolons
- Functions
- Libraries, Header Files, #include
- Other General Syntax Points

Additional resources:

- The C++ Language Tutor
- The Google C++ Style Guide
- The Arduino Standard Library Language Reference

Setting up your Hardware

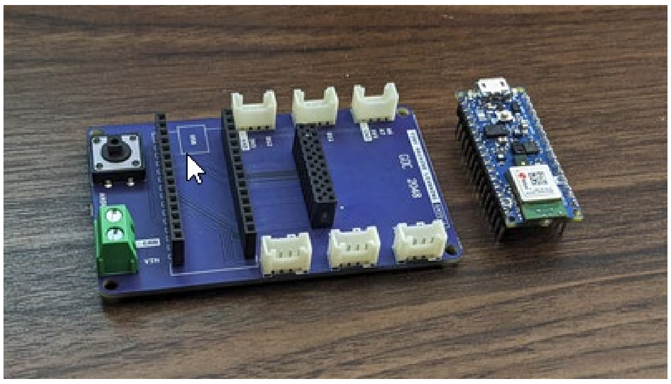
This reading will walk you through setting up your hardware whether you purchased the TinyML Kit or your own off-the-shelf parts.

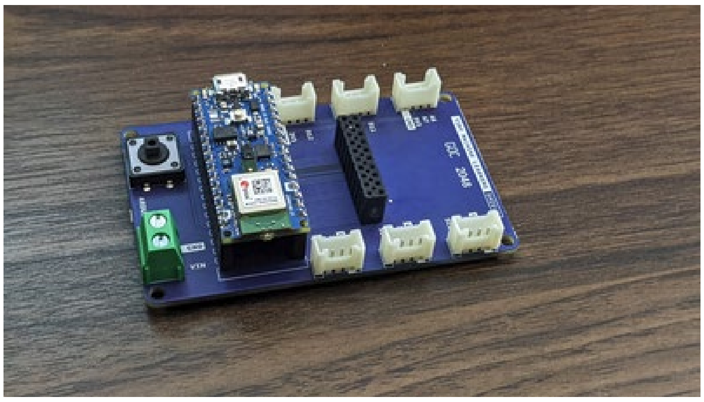
03_TinyML_C03_04-02-03_final-en.mp4 (Command Line)

Setting up the TinyML Kit

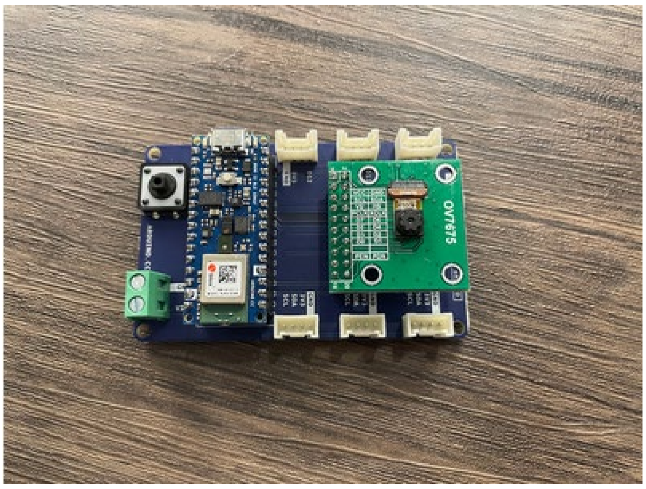
Certainly, the easiest and most reliable way to obtain all of the parts necessary for this course is to procure the purpose-designed kit that we have developed with Arduino, available at this link for $49.99. If you ordered the official kit, then getting setup will be quick and easy. We outline the required steps below:

1. Slot the Nano 33 BLE Sense board into the Tiny Machine Learning Shield.

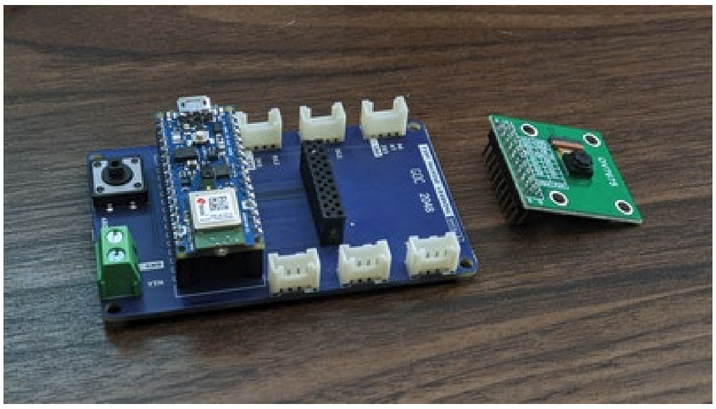




You'll want to target the pair of spatially separated 1x15 female headers. Carefully align the pins of the microcontroller board with the headers below and then gently push down until the board is seated flush against the top of each header. The downward facing pins should no longer be visible. As best you can, avoid touching the components atop the board to prevent inadvertently damaging the surface mount devices. Pay attention to the orientation of the board so that the indication of the USB port on the PCB silkscreen matches the physical port on the board itself.

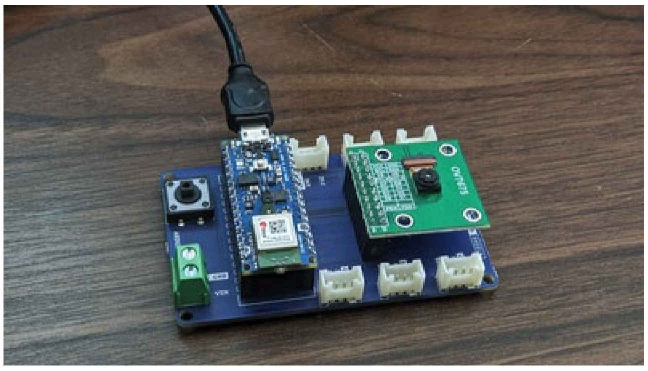


2. Slot the OV7675 camera module into the shield using the same technique.



You'll want to target the 2x10 female header. Carefully align the pins of the camera module with the headers and then gently push down until the board is seated flush against the top of each header. The downward facing pins should no longer be visible. As best you can, avoid touching the camera module atop the board to prevent inadvertent damage. Pay attention to the orientation of the camera module so that the camera sensor is to the right of the header array (as shown), further from the microcontroller board than the header array.

3. Finally, use the provided USB cable (type-A to microB) to connect the Nano 33 BLE Sense development board to your machine. If your PC only features type-C USB ports, you will need to obtain an adaptor.



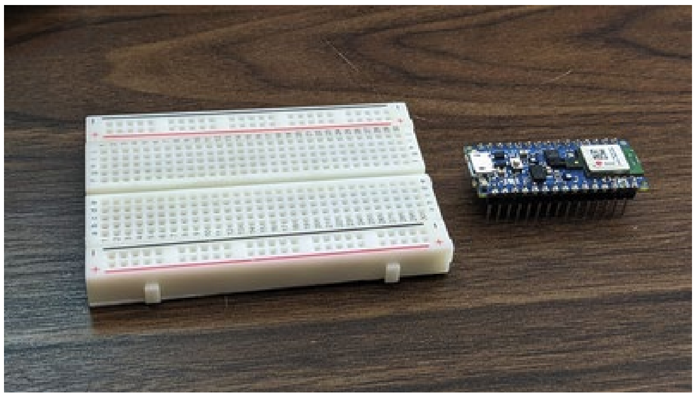
Note that if you are *only* calling upon hardware found on the Nano 33 BLE Sense development board (say the MCU and IMU), you could forgo connecting it to the Tiny Machine Learning Shield. If you need to remove either the Nano board or the camera module from the shield, grip each side of whichever board and pull back with a *gentle* rocking motion (back and forth) to work the pins out from the headers below.

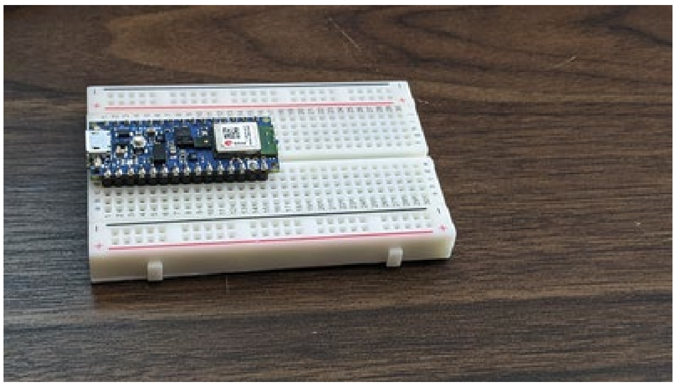
And that's it! If you purchased the official kit you can move onto the next reading!

Wiring Up Off-the-Shelf Parts

While we generally encourage all to consider Arduino's Tiny Machine Learning kit as the primary and recommended hardware option, we understand that in certain circumstances, it may be beneficial, or even necessary, to obtain some or all of these parts via other means. If you have bought off-the-shelf components using our guide, we outline the steps you are likely to take in getting setup, below.

1. Slot the Nano 33 BLE Sense board into a solderless breadboard.

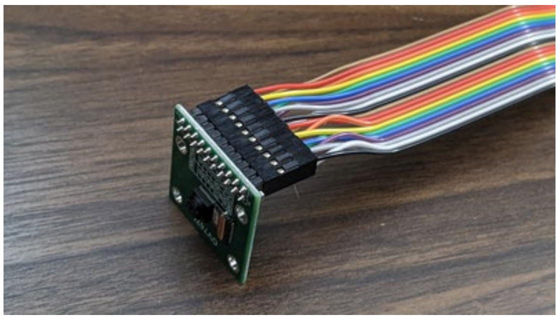




If you aren't familiar with how breadboards can be used as a substrate for connecting various electrical components, you can take a moment to review the mechanism, here. As shown in the pictures above, we've targeted the left-hand side of the breadboard and seated the microcontroller board into the solderless breadboard along its length, with each of the 1x15 arrays of male pins on opposite sides of the 'trough' that runs through the middle of the breadboard. Given the board's width, one side will have three rows of receptacles beside it along the board while the other will have two. Which side is of no consequence. Note also that we've positioned the USB port at the end of the breadboard, to facilitate connections to your PC. So, altogether, you'll want to carefully align the pins of the microcontroller board with the receptacles below and then gently push down until the board is seated flush against the breadboard. The downward facing pins should not be visible. Avoid touching the components atop the board to prevent inadvertently damaging the surface mount devices.

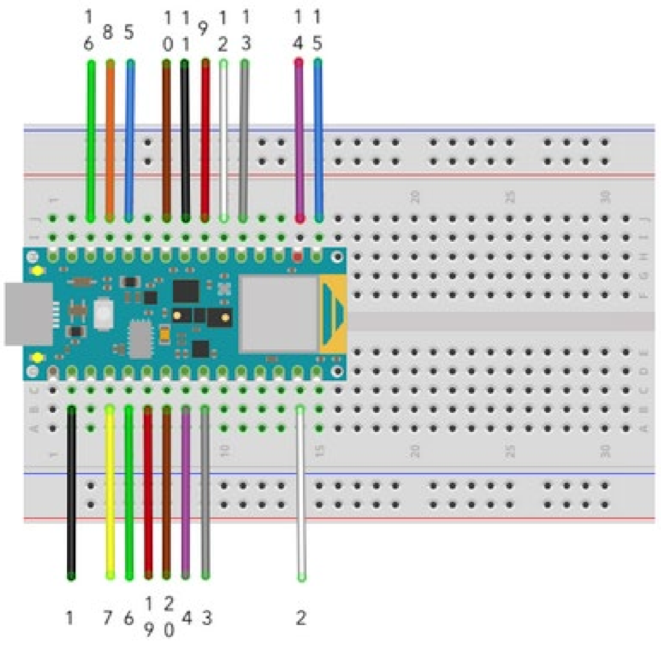
2. With female-to-male jumper wire, use the following Fritzing (wiring) diagram, pinout diagrams, and connection table to link the OV7675 camera module to the microcontroller board via the solderless breadboard.

   You can start by isolating (tearing away, as applicable) 20 adjoined female-to-male jumper wires as shown in the first photo above. For clarity, keep the desired 20 wires connected together, but tear away any number above this. The sequence of colors is of no consequence. Next connect the female side of this wire assembly to the male pins of the camera module in an alternating pattern. To keep track of what each color represents, it may be easiest to have the left or right-most color in your wire assembly connect to pin 1, where the next color in your sequence connects to 2, and so on. That way, the camera module's pinout is now encoded by the sequence of colors.

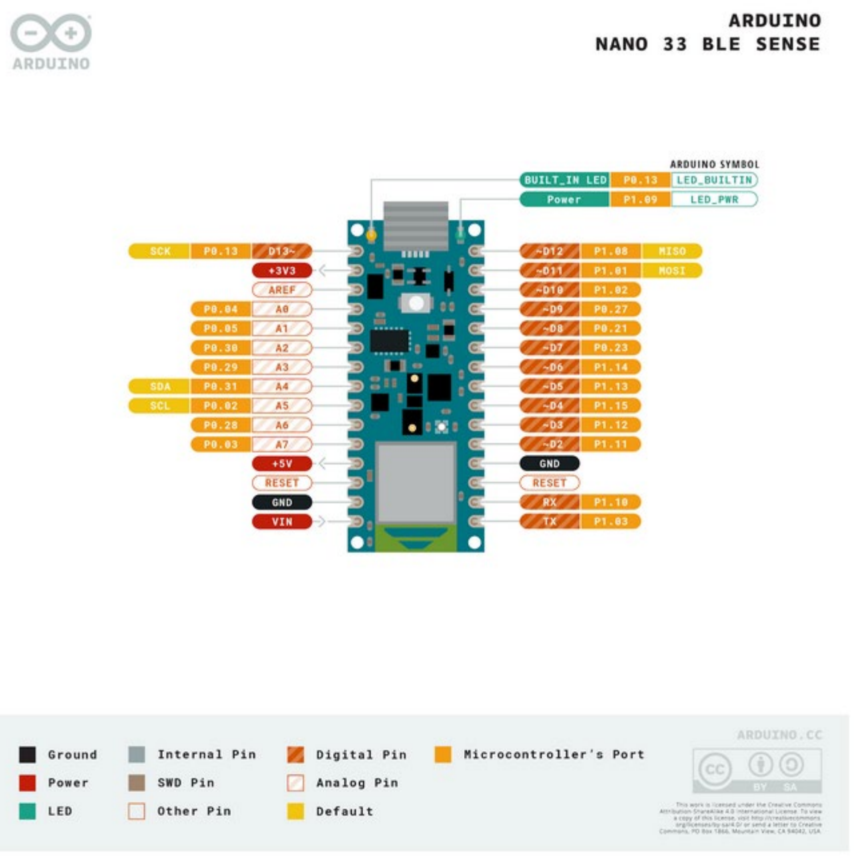


3. Our next step will be to connect the camera module pinout (1-20) to specific pins on the microcontroller board via the solderless breadboard.

   Below we’ve mapped these OV7675 module pin numbers onto a fritzing (wiring) diagram for the Arduino Nano 33 BLE sense (assuming it is placed into a breadboard):



   In case also helpful, we have also included the full pinout (designation) for the Nano 33 BLE Sense development board below as well as a table explaining the full pin connections needed from the OV7675 to the Arduino Nano 33 BLE Sense:



Description	Camera Module Pin	Microcontroller Board Pin
VCC / 3.3V	1	3.3V
GND	2	GND
SIOC / SCL	3	SCL / A5
SIOD / SDA	4	SDA / A4
VSYNC / VS	5	D8
HREF / HS	6	A1
PCLK	7	A0
XCLK	8	D9
D7	9	D4
D6	10	D6
D5	11	D5
D4	12	D3
D3	13	D2
D2	14	D0/RX
D1 (may be labeled D0)	15	D1/TX
D0 (may be labeled D1)[1]	16	D10
NC	17	--
NC	18	--
PEN / RST	19	A2
PWDN / PDN	20	A3

So, for example, the black wire shown in the top right of the image of the set of wires connected to the OV7675 above, is connected to the camera module's first pin (pin #1). We can directly see on the fritzing diagram that pin #1 from the OV7675 should be connected to the second to the top left pin on the Arduino. We can also see that according to the table above, we need to connect that to the Arduino's 3.3V pin, which we can find in the pinout diagram above as being the second pin on the top left.

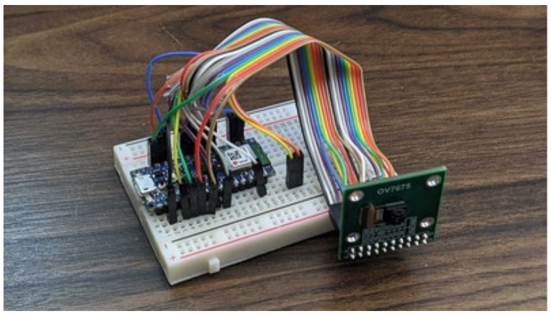
Another example: the second wire (white) should then connect to either the second pin from the bottom on the left (as we marked on the fritzing diagram), or the fourth pin from the bottom on the right hand side of the MCU board, as either spot[2] will provide a connection to GND.

Continue down the table (or around the fritzing diagram) until all pins have been connected, outside of pins 17 and 18, which are left unconnected. We choose to include wires for these pins, even though they serve no purpose, because their presence in connecting to the camera module makes the entire cable assembly physically more robust, which is favorable in maintaining a solid connection.

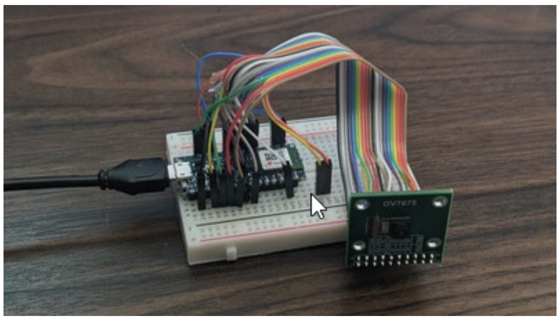
We want to underline that you should take your time in this process and verify each connection as you go, because most (nearly all) of the issues reported for this camera module stem from a wire that is connected to the wrong pin - perhaps only one location off or so. Precision here for the entire table is required for functionality.

If you're interested, the communication between the controller and the camera module is standardized as a Serial Camera Control Bus (SCCB) as well as a Camera Parallel Interface (CPI) or, equivalently, Digital Video Port (DVP).

When you are done wiring up the camera your setup should like something like the following:



4. Finally, use the provided USB cable (type-A to microB) to connect the Nano 33 BLE Sense development board to your machine. If your PC only features type-C USB ports, you will need to obtain an adaptor.



It's probably apparent why we are so excited that Arduino has developed the Tiny Machine Learning Shield for our course kits! :)

If you've sourced your jumper wires from SparkFun, you may be able to call on the same sequence of colors as we have above.

Changes for the OV7670 Camera Module

While the course staff will not officially support the OV7670 camera module, it exists as a viable alternative if the OV7675 is unavailable in your region. Fortunately, there is an Arduino blog post concerning using the OV7670 module for tinyML.

IMPORTANT NOTE: The connection table in the Arduino blog post is out of date, you should use the same connection table as for the OV7675 listed above. The one change is that the OV7670 only has 18 pins. Therefore, you need to omit the two NC pins between D0/D1 and RST/PWDN.

---

[1]Be careful, the silkscreen label on some OV7675 camera modules mistakenly swaps D0 and D1!
[2]Note that GND, or ground, is the only example where a pin designation is repeated.

Setting up your Software

In this reading, we will walk through setting up the software you'll need for this course, the Arduino integrated development environment (IDE). We will be using the Arduino IDE to program your microcontroller.

Note: you do not need your Aduino connected to your computer for this step so you can install the software even if you do not yet have a kit!

03_TinyML_C03_04-02-05-A-en.mp4 (Command Line)

What is Arduino?

Arduino describes itself as "an open-source electronics platform based on easy-to-use hardware and software." In large part, this description is fitting, as the company designs and sells a collection of microcontroller development boards that simplify deployment of embedded hardware alongside a software framework that abstracts away all, but the most relevant considerations for your application. Perhaps, what's under-represented by this description is the role that the surrounding community plays in enabling many plug-and-play experiences given the number and quality of auxiliary hardware modules, support libraries, and tutorials that have and continue to be produced within Arduino's ecosystem.

One thing we'd like to acknowledge here is that Arduino's mission of creating easy-to-use hardware and software has the necessary tradeoff of limiting the feature set of its development environment to the essentials.

What is an Integrated Development Environment (IDE)?

As is true for all forms of programming, an integrated development environment, or IDE, is an application with a feature set that facilitates software development generally or within a particular niche. The Arduino Desktop IDE is highly specific in the sense that it is intended to facilitate software development for a specific set of microcontroller boards, in C++.

Within the niche of IDEs for embedded software, there is a noticeable dichotomy between light-weight applications, (like Arduino's IDE) that minimize functionality and abstract away details in the name of simplicity and full-featured IDEs (like Keil µVision and Visual Studio Code) that exist to support industry professionals.

If you're interested in finding a happy medium to use in future embedded projects, our staff have had good experience using VS Code with the hardware specific extension PlatformIO, that together enable nice-to-have features like line completion, reference tracking, etcetera, without all of the complexity that more advanced IDEs introduce. Having said this, this particular combination of VS Code + PlatformIO will not be officially supported within this course.

What is the Arduino IDE?

As you might expect given the description above, the Arduino IDE is a light-weight development environment with features that permit you to very quickly manipulate microcontroller development boards. While there is a cloud-based offering (the Arduino Create Web Editor) as well as a so-called Arduino Pro IDE that is in development (specifically in the alpha stage at the time of writing), we are going to use the standard Arduino Desktop IDE in this course.

Downloading and Installing the Arduino IDE

Note: Our staff will only officially support the the Arduino Desktop IDE v1.8.15 or later (tested up to v1.8.19). If you have an older version of the Desktop IDE, we recommend that you update your software.

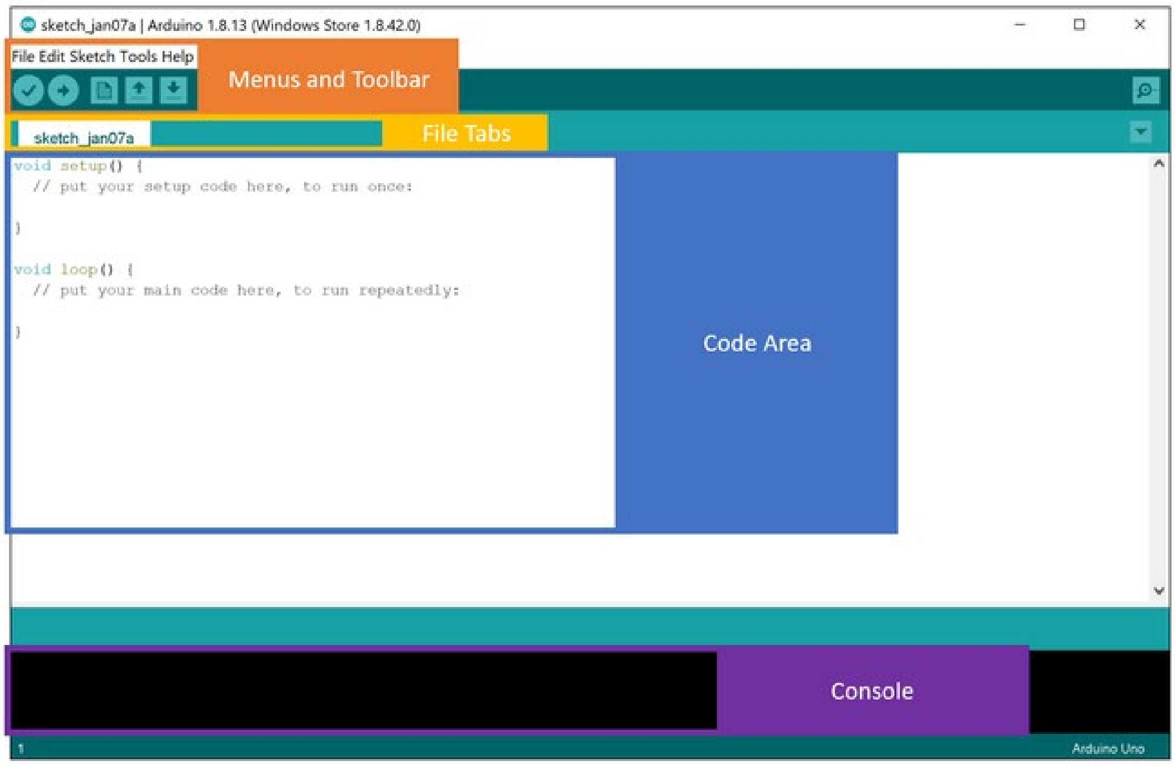
0. Navigate to arduino.cc and at the top of the page select **Software** and click **Downloads**.
1. Click on the download link appropriate for your machine.
2. There is no obligation to contribute, you can click 'Just Download' to proceed.

Operating specific installation instructions vary, so if what follows isn't self evident, you can find guidance, by OS, at the following links:

- Windows

- Mac OS X
- Linux

A Quick Tour of the IDE


sketch_jan07a | Arduino 1.8.13 (Windows Store 1.8.42.0)
File Edit Sketch Tools Help
Menus and Toolbar
sketch_jan07a
File Tabs
Code Area
Console
Arduino Uno

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

When you first open the IDE, you will see a screen that looks something like the above screenshot.

Up at the top of the IDE, you will find the menus and the toolbar, which we will explore as we work through the rest of this document and when we run the tests later in this section.

Below that, you will find the file tabs. For now, there will only be one file open that is most likely named sketch_<date>, however, later in the course when we work through more complicated examples, you will see all of the various files that make up the project across the file tab area. In "Arduino speak," a sketch is a simple project/application.

In the middle of the screen, you will see the large code area. Each sketch can consist of many files and use many libraries, but at its core, each sketch is made of two main functions, "setup" and "loop", which you can see pre-populated in the code area. We'll explore what those functions are used for through the Blink example in a future video. For now, let's continue orienting ourselves with and setting up the IDE.

Finally, at the bottom of the screen, you'll find the console. This is where you will see debug and error messages that result from compiling your C++ sketch and uploading it to your Arduino.

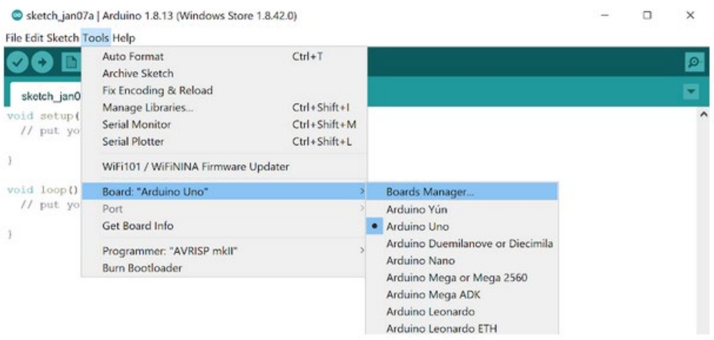
Installing the Board Files for the Nano 33 BLE Sense

03_TinyML_C03_04-02-05-B_final-en.mp4 (Command Line)

One of the primary advantages that the Arduino ecosystem affords is the portability of code you write for one or another board within their line-up or even in porting code to affiliate boards. This is made possible by the support files organized in the Boards Manager, which coordinates a download and installation of files that detail the Arduino functions (sometimes 'core') that are defined for that particular board (which is how hardware differences between boards are abstracted) as well as compiler or linker details specific to the given board.

To install the board files that you will need for your Arduino Nano 33 BLE Sense, please do the following:

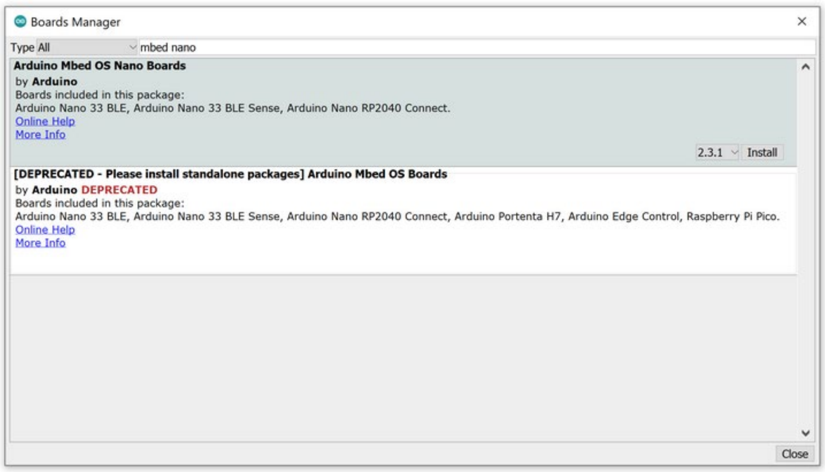
1. Open the Boards Manager, which you can find via the Tools drop-down menu. Navigate, as follows: Tools → Board → Boards Manager. Note that the Board may be set to "Arduino UNO" by default.



sketch_jan07a | Arduino 1.8.13 (Windows Store 1.8.42.0)
File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "Arduino Uno"
Port
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader
Boards Manager...
Arduino Yún
Arduino Uno
Arduino Duemilanove or Diecimila
Arduino Nano
Arduino Mega or Mega 2560
Arduino Mega ADK
Arduino Leonardo
Arduino Leonardo ETH

```
sketch_jan0
void setup(
  // put yo

}

void loop()
  // put yo

}
```

2. In the Boards Manager dialog box, use the search bar at the top right to search for "mbed nano" which should bring up a few results. We're interested in the first result (as shown), named "Arduino MBED OS Nano Boards,". Make sure you select **Version 2.3.1 or newer (tested up to Version 3.0.1)** and then click "Install." As the install process progresses, you will see a blue completion bar work its way across the bottom of the Board Manager window. Be patient, you may need to install USB drivers, which requires you to approve an administrator privileges popup which can take a couple of minutes to appear.

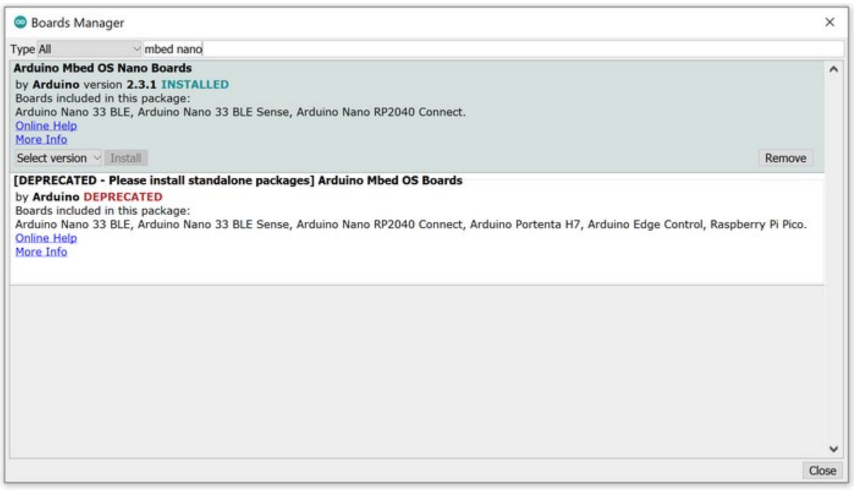


mbed nano

Arduino Mbed OS Nano Boards
by **Arduino**
Boards included in this package:
Arduino Nano 33 BLE, Arduino Nano 33 BLE Sense, Arduino Nano RP2040 Connect.

[DEPRECATED - Please install standalone packages] Arduino Mbed OS Boards
by **Arduino** DEPRECATED
Boards included in this package:
Arduino Nano 33 BLE, Arduino Nano 33 BLE Sense, Arduino Nano RP2040 Connect, Arduino Portenta H7, Arduino Edge Control, Raspberry Pi Pico.

After you have successfully installed the board, if you exit and re-open the Board Manager and search again for “mbed nano” you will now see INSTALLED next to the library and the option to “Remove” the library or install a different version.


Boards Manager

Type All | mbed nano

Arduino Mbed OS Nano Boards
by **Arduino** version **2.3.1** INSTALLED
Boards included in this package:
Arduino Nano 33 BLE, Arduino Nano 33 BLE Sense, Arduino Nano RP2040 Connect.
Online Help
More Info
Select version | Install | Remove

[DEPRECATED - Please install standalone packages] Arduino Mbed OS Boards
by **Arduino** DEPRECATED
Boards included in this package:
Arduino Nano 33 BLE, Arduino Nano 33 BLE Sense, Arduino Nano RP2040 Connect, Arduino Portenta H7, Arduino Edge Control, Raspberry Pi Pico.
Online Help
More Info

Close

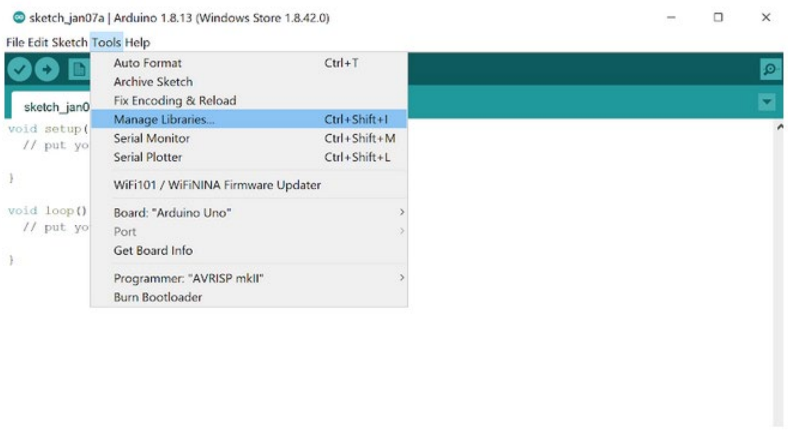
Installing the Libraries Needed for this Course

03_TinyML_C03_04-02-05-C_final-en.mp4 (Command Line)

Another advantage of the Arduino ecosystem is the availability of a wide array of libraries for performing various tasks, such as interfacing with a sensor module or manipulating data using common algorithms. There are many libraries that can be accessed from within the Library Manager in the Arduino IDE as described below. Check here for a complete list.

For this course, we are going to need four libraries. To install the libraries, please do the following and **make sure to install the version specified in the reading below or the tinyML applications will not work**:

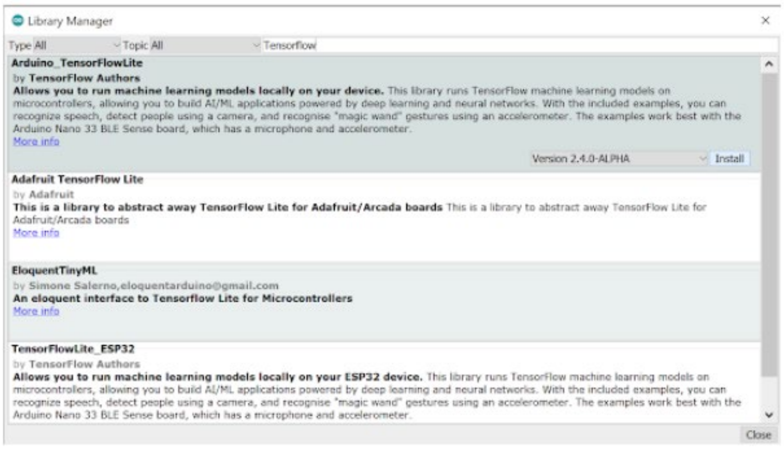
1. Open the Library Manager, which you can find via the Tools drop-down menu. Navigate, as follows: Tools → Manage Libraries.


sketch_jan07a | Arduino 1.8.13 (Windows Store 1.8.42.0)
File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "Arduino Uno"
Port
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader
sketch_jan0
void setup(
// put yo
void loop()
// put yo

2. Then, much like for the Boards Manager, in the Library Manager dialog box, use the search bar at the top right to search for the following libraries, one at a time. Note that

like with the Board manager, a blue completion bar will appear across the bottom of the Library Manager window.

- The Tensorflow Lite Micro Library:
    - Search Term: Tensorflow
    - Library Name: Arduino_TensorFlowLite
    - Version: 2.4.0-ALPHA



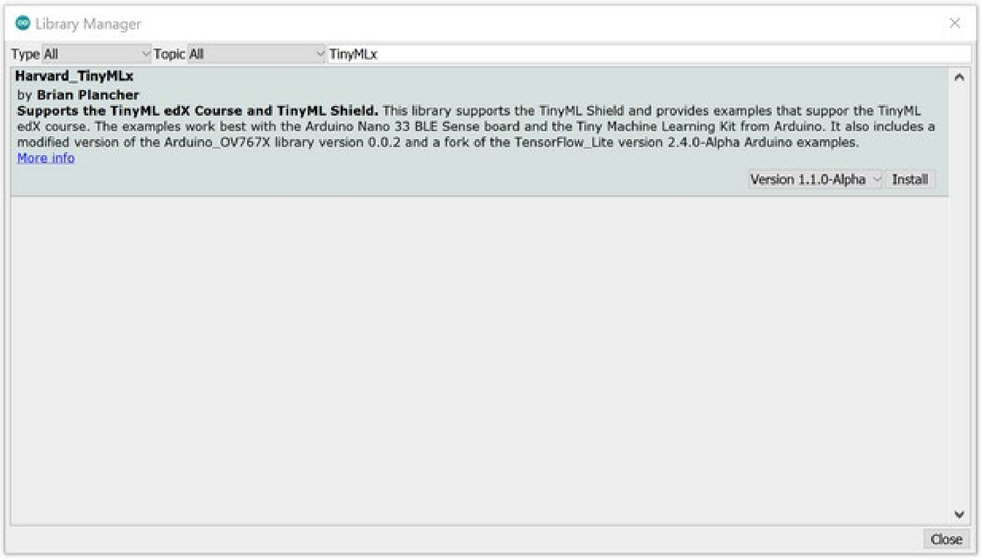
Arduino_TensorFlowLite
by **TensorFlow Authors**
Allows you to run machine learning models locally on your device. This library runs TensorFlow machine learning models on microcontrollers, allowing you to build AI/ML applications powered by deep learning and neural networks. With the included examples, you can recognize speech, detect people using a camera, and recognise "magic wand" gestures using an accelerometer. The examples work best with the Arduino Nano 33 BLE Sense board, which has a microphone and accelerometer.
More info
Version 2.4.0-ALPHA Install

Adafruit TensorFlow Lite
by **Adafruit**
This is a library to abstract away TensorFlow Lite for Adafruit/Arcada boards This is a library to abstract away TensorFlow Lite for Adafruit/Arcada boards
More info

EloquentTinyML
by **Simone Salerno,eloquentarduino@gmail.com**
An eloquent interface to Tensorflow Lite for Microcontrollers
More info

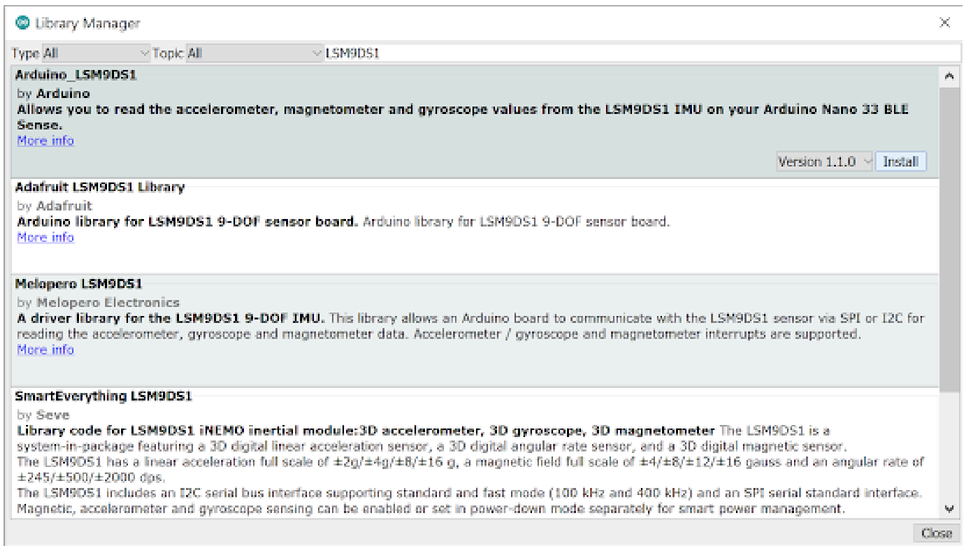
TensorFlowLite_ESP32
by **TensorFlow Authors**
Allows you to run machine learning models locally on your ESP32 device. This library runs TensorFlow machine learning models on microcontrollers, allowing you to build AI/ML applications powered by deep learning and neural networks. With the included examples, you can recognize speech, detect people using a camera, and recognise "magic wand" gestures using an accelerometer. The examples work best with the Arduino Nano 33 BLE Sense board, which has a microphone and accelerometer.

- The Harvard_TinyMLx Library we put together for this course!
    - Search Term: TinyMLx
    - Library Name: Harvard_TinyMLx
    - Version: 1.1.0-Alpha



Harvard_TinyMLx
by **Brian Plancher**
Supports the TinyML edX Course and TinyML Shield. This library supports the TinyML Shield and provides examples that suppor the TinyML edX course. The examples work best with the Arduino Nano 33 BLE Sense board and the Tiny Machine Learning Kit from Arduino. It also includes a modified version of the Arduino_OV767X library version 0.0.2 and a fork of the TensorFlow_Lite version 2.4.0-Alpha Arduino examples.
More info
Version 1.1.0-Alpha Install

- The library that supports the accelerometer, magnetometer, and gyroscope on the Nano 33 BLE sense:
    - Search Term: LSM9DS1
    - Library Name: Arduino_LSM9DS1
    - Version: 1.1.0


Library Manager

Type All Topic All LSM9DS1

Arduino_LSM9DS1
by **Arduino**
Allows you to read the accelerometer, magnetometer and gyroscope values from the LSM9DS1 IMU on your Arduino Nano 33 BLE Sense.
More info
Version 1.1.0 Install

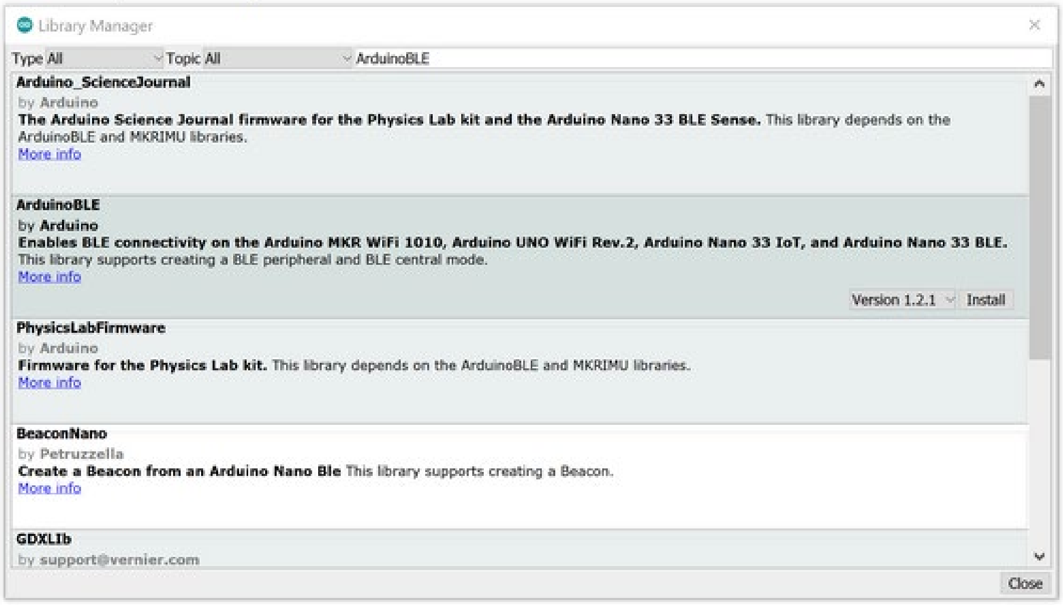
Adafruit LSM9DS1 Library
by Adafruit
Arduino library for LSM9DS1 9-DOF sensor board. Arduino library for LSM9DS1 9-DOF sensor board.
More info

Melopero LSM9DS1
by Melopero Electronics
A driver library for the LSM9DS1 9-DOF IMU. This library allows an Arduino board to communicate with the LSM9DS1 sensor via SPI or I2C for reading the accelerometer, gyroscope and magnetometer data. Accelerometer / gyroscope and magnetometer interrupts are supported.
More info

SmartEverything LSM9DS1
by Seve
Library code for LSM9DS1 iNEMO inertial module:3D accelerometer, 3D gyroscope, 3D magnetometer The LSM9DS1 is a system-in-package featuring a 3D digital linear acceleration sensor, a 3D digital angular rate sensor, and a 3D digital magnetic sensor.
The LSM9DS1 has a linear acceleration full scale of ±2g/±4g/±8/±16 g, a magnetic field full scale of ±4/±8/±12/±16 gauss and an angular rate of ±245/±500/±2000 dps.
The LSM9DS1 includes an I2C serial bus interface supporting standard and fast mode (100 kHz and 400 kHz) and an SPI serial standard interface.
Magnetic, accelerometer and gyroscope sensing can be enabled or set in power-down mode separately for smart power management.

Close

- ArduinoBLE:
  - Search Term: ArduinoBLE
  - Library Name: ArduinoBLE
  - Version: 1.2.1 or newer (tested up to 1.2.2)


Library Manager

Type All Topic All ArduinoBLE

Arduino_ScienceJournal
by Arduino
The Arduino Science Journal firmware for the Physics Lab kit and the Arduino Nano 33 BLE Sense. This library depends on the ArduinoBLE and MKRIMU libraries.
More info

ArduinoBLE
by **Arduino**
Enables BLE connectivity on the Arduino MKR WiFi 1010, Arduino UNO WiFi Rev.2, Arduino Nano 33 IoT, and Arduino Nano 33 BLE. This library supports creating a BLE peripheral and BLE central mode.
More info
Version 1.2.1 Install

PhysicsLabFirmware
by Arduino
Firmware for the Physics Lab kit. This library depends on the ArduinoBLE and MKRIMU libraries.
More info

BeaconNano
by Petruzzella
Create a Beacon from an Arduino Nano Ble This library supports creating a Beacon.
More info

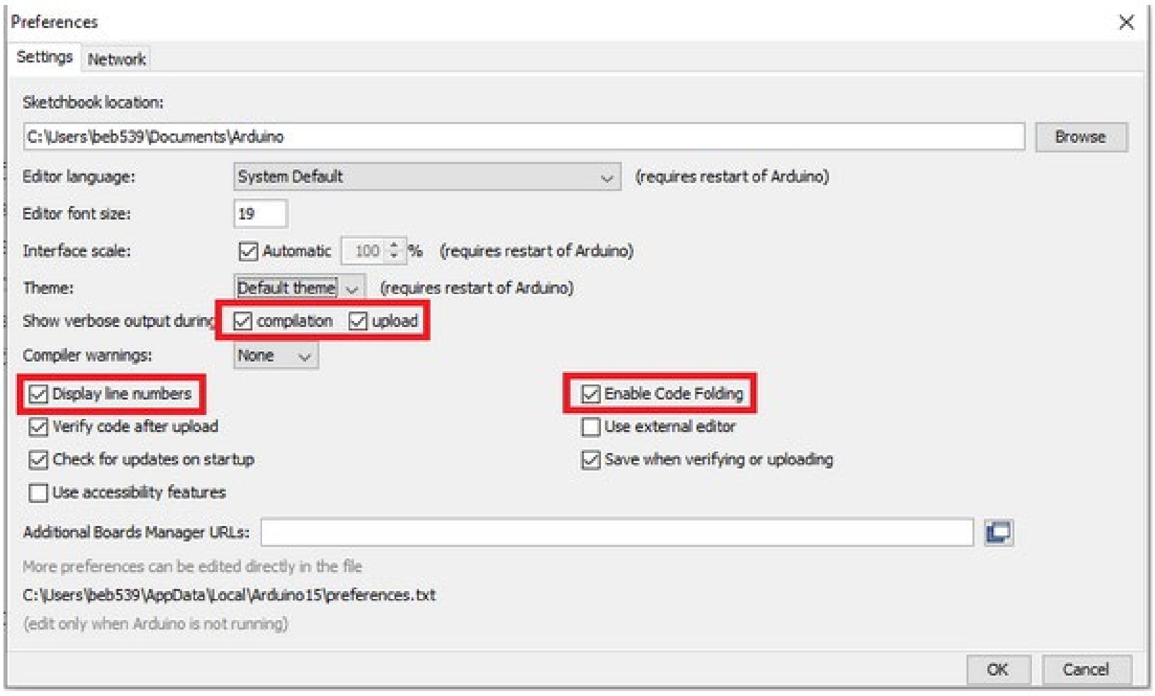
GDXLib
by support@vernier.com

Close

Setting your Preferences

You can adjust the preferences set for the Arduino Desktop IDE via the File drop-down menu, File → Preferences. There are a few preferences that we recommend enabling to make the Arduino IDE a little easier to use, namely:

1. Show verbose output during: compilation and upload.
2. Enable code folding.
3. Display line numbers.


Preferences

Settings | Network

Sketchbook location:
C:\Users\beb539\Documents\Arduino

Editor language: System Default (requires restart of Arduino)
Editor font size: 19
Interface scale: ☑ Automatic 100 % (requires restart of Arduino)
Theme: Default theme (requires restart of Arduino)
Show verbose output during: ☑ compilation ☑ upload
Compiler warnings: None

☑ Display line numbers
☑ Verify code after upload
☑ Check for updates on startup
☐ Use accessibility features
☑ Enable Code Folding
☐ Use external editor
☑ Save when verifying or uploading

Additional Boards Manager URLs:
More preferences can be edited directly in the file
C:\Users\beb539\AppData\Local\Arduino15\preferences.txt
(edit only when Arduino is not running)

On final note, if you don't like the default theme for the Arduino Desktop IDE, there is a nice tutorial for a dark theme you can find here. Also, if you would like to learn more about the IDE, check out Arduino's documentation.

And that's it! Your Arduino IDE should be all configured for this course. Now that you have all of the necessary board files and libraries installed, it's time to explore more of the features of the IDE available under the "Tools" menu and start to test out your Arduino by deploying the Blink example!

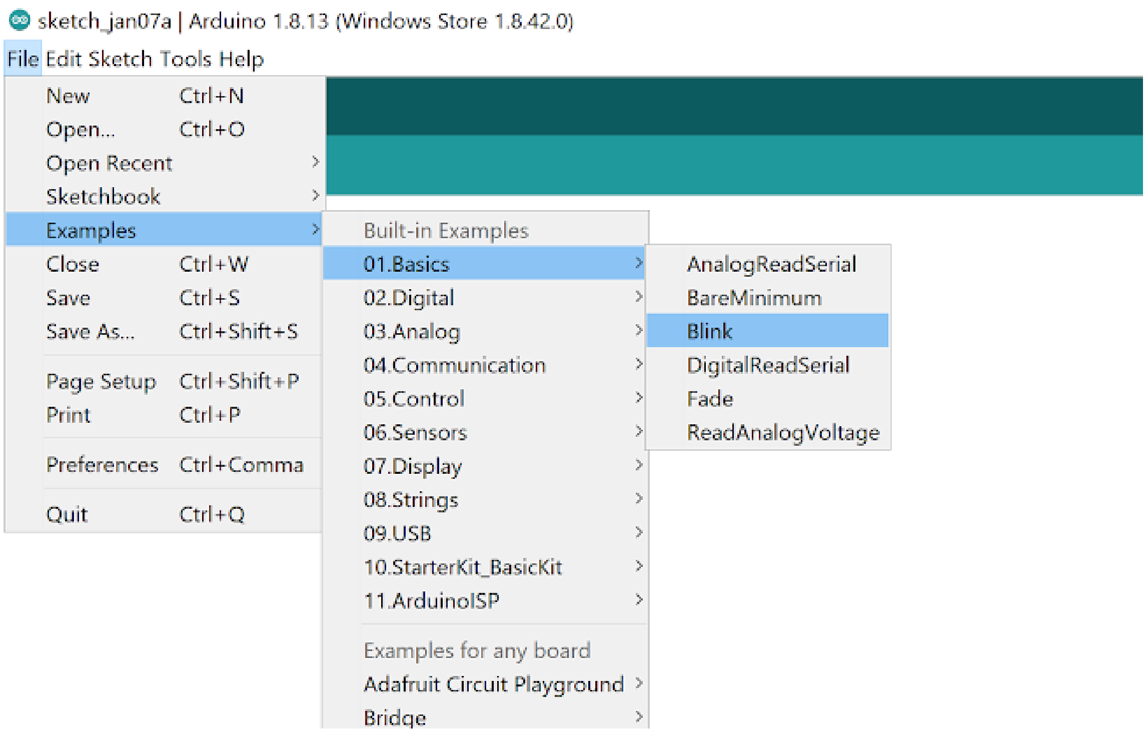
The Arduino Blink Example

In this reading, we will deploy the Arduino Blink example to make sure everything is working properly and to give you your first experience deploying code to your Arduino!

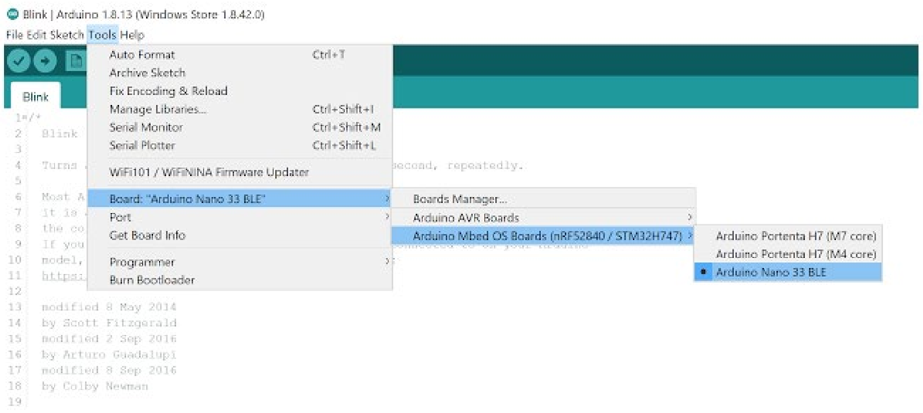
03_TinyML_C03_04-02-08-A_final-en.mp4 (Command Line)

Preparing for Deployment

1. Use a USB cable to connect the Arduino Nano 33 BLE Sense to your machine. You should see a green LED power indicator come on when the board first receives power.
2. Open the Blink.ino sketch, which you can find via the File drop-down menu. Navigate, as follows: File → Examples → 01.Basics → Blink. You'll notice that Arduino has provided a wealth of examples to choose from should you like to explore the board more on your own outside of the course material. There is great documentation about those examples on the Arduino website.



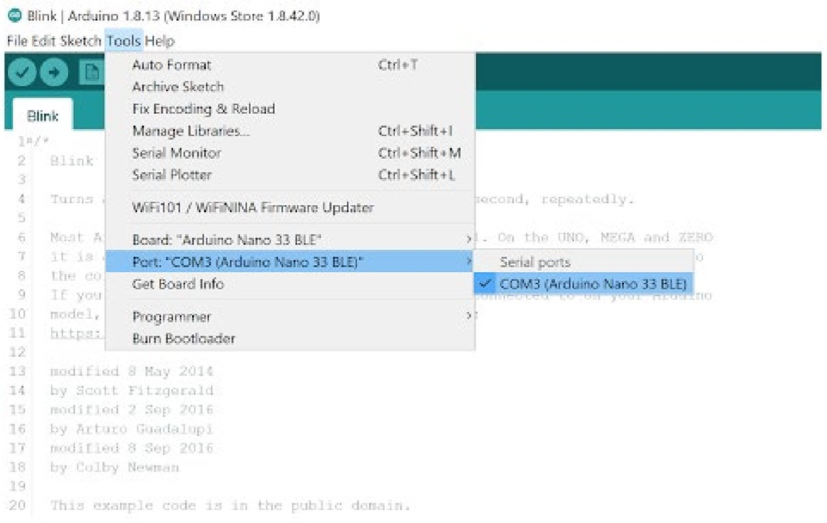
3. Use the Tools drop-down menu to select appropriate Port and Board. This is important as it is telling the IDE which board files to use and on which serial connection it should send the code. In some cases, this may happen automatically, but if not, you'll want to select:
    - Select the Arduino Nano 33 BLE as the board by going to Tools → Board: <Current Board Name> → Arduino Mbed OS Boards (nRF52840) → Arduino Nano 33 BLE. Note that on different operating systems, the exact name of the board may vary but/and, it should include the word Nano at a minimum. If you do not see that as an option, then please go back to Setting up the Software and make sure you have installed the necessary board files.


Blink | Arduino 1.8.13 (Windows Store 1.8.42.0)
File Edit Sketch Tools Help
Blink
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "Arduino Nano 33 BLE"
Port
Get Board Info
Programmer
Burn Bootloader
Boards Manager...
Arduino AVR Boards
Arduino Mbed OS Boards (nRF52840 / STM32H747)
Arduino Portenta H7 (M7 core)
Arduino Portenta H7 (M4 core)
• Arduino Nano 33 BLE

```
/*
  Blink

  Turns                                   second, repeatedly.

  Most A
  it is
  the co
  If you
  model,
  https:

  modified 8 May 2014
  by Scott Fitzgerald
  modified 2 Sep 2016
  by Arturo Guadalupi
  modified 8 Sep 2016
  by Colby Newman
```

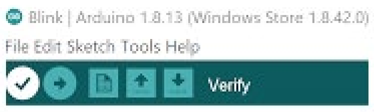
    - Then, select the USB Port associated with your board. This will appear differently on Windows, macOS, Linux, but will likely indicate 'Arduino Nano 33 BLE" in parenthesis. You can select this by going to Tools → Port: <Current Port (Board on Port)> → <TBD Based on OS> (Arduino Nano 33 BLE). Where <TBD Based on OS> is most likely to come from the list below where <#> indicates some integer number.

- Windows → COM<#>
- macOS → /dev/cu.usbmodem<#>
- Linux → ttyUSB<#> or ttyACM<#>


Blink | Arduino 1.8.13 (Windows Store 1.8.42.0)
File Edit Sketch Tools Help
Blink
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "Arduino Nano 33 BLE"
Port: "COM3 (Arduino Nano 33 BLE)"
Get Board Info
Programmer
Burn Bootloader
Serial ports
COM3 (Arduino Nano 33 BLE)

```
/*
  Blink

  Turns                                    second, repeatedly.

  Most A                                   . On the UNO, MEGA and ZERO
  it is                                    o
  the co
  If you                                   connected to on your Arduino
  model,
  https:

  modified 8 May 2014
  by Scott Fitzgerald
  modified 2 Sep 2016
  by Arturo Guadalupi
  modified 8 Sep 2016
  by Colby Newman

  This example code is in the public domain.
```

4. Finally, use the checkmark button at the top left of the UI to verify that the code within the example sketch is valid.


Blink | Arduino 1.8.13 (Windows Store 1.8.42.0)
File Edit Sketch Tools Help
Verify

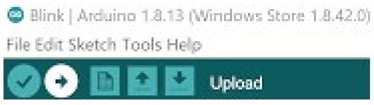
Verification will compile the code, so take note of the status and results indicated in the black console at the bottom of the IDE. The level of detail presented here will depend on whether or not you have enabled ‘verbose output during compilation’ in Preferences. You should most likely at the end see a final output indicating how much memory the sketch will take on the Arduino once it is uploaded. Something like: “Sketch uses 86568 bytes (8%) of program storage space. Maximum is 983040 bytes. Global variables use 44696 bytes (17%) of dynamic memory, leaving 217448 bytes for local variables. Maximum is 262144 bytes.” As you can see, this is a very simple example and does not take up much space. While, as you’ll see in a moment, uploading your code will also verify your code automatically, it is often helpful to verify your code first as you can iron out any compilation errors without having any hardware on hand.

Deploying (Uploading) the Sketch

Once we know that the code at hand is valid, we can ‘flash’ it to the MCU:

1. Use the rightward arrow next to the ‘compile’ checkmark to upload / flash the code.

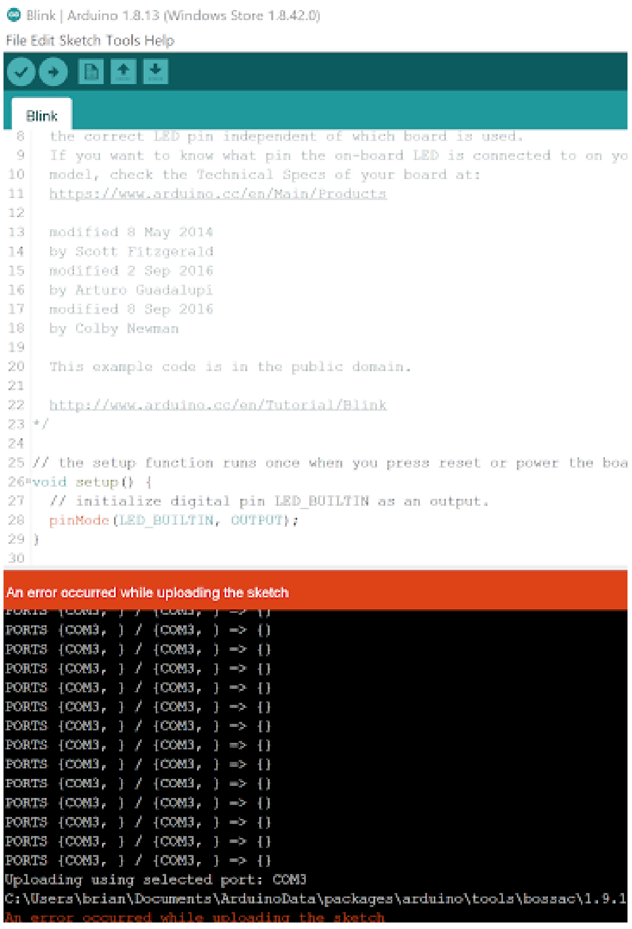
   Note that pragmatically, this step will re-compile the sketch before flashing the code, so that in the future if you intend to sequentially compile and flash a program, you need to only press the ‘upload’ arrow.


Blink | Arduino 1.8.13 (Windows Store 1.8.42.0)
File Edit Sketch Tools Help
Upload

As before, take note of the status and results indicated in the black console at the bottom of the IDE. The level of detail presented here will depend on whether or not you have enabled ‘verbose output during compilation’ in Preferences, accessible via the File drop-down menu in the IDE.

You’ll know the upload is complete when you see red text in the console at the bottom of the IDE that shows 100% upload of the code and a statement that says something like “Done in <#.#> seconds.” Again, if this is the first time you are uploading a sketch to an Arduino, the upload may hang for a little while until you get another administrator approval popup and approve it. Don’t worry, this is just a one time thing.

If you receive an error, you will see an orange error bar appear and a red error message in the console (as shown below). Don’t worry -- there are many common reasons this may have occurred. To help you debug, please check out our FAQ appendix with answers to the most common errors!

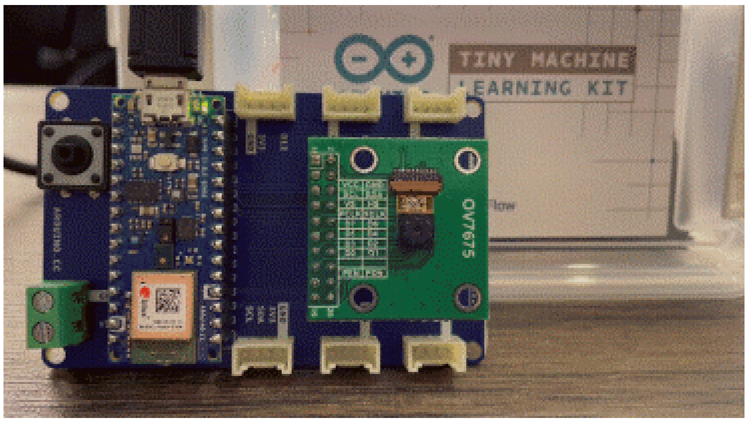

Blink | Arduino 1.8.13 (Windows Store 1.8.42.0)
File Edit Sketch Tools Help
Blink

```
  the correct LED pin independent of which board is used.
  If you want to know what pin the on-board LED is connected to on yo
  model, check the Technical Specs of your board at:
  https://www.arduino.cc/en/Main/Products

  modified 8 May 2014
  by Scott Fitzgerald
  modified 2 Sep 2016
  by Arturo Guadalupi
  modified 8 Sep 2016
  by Colby Newman

  This example code is in the public domain.

  http://www.arduino.cc/en/Tutorial/Blink
*/

// the setup function runs once when you press reset or power the boa
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

```

An error occurred while uploading the sketch

```
PORTS {COM3, } / {COM3, } => {}
PORTS {COM3, } / {COM3, } => {}
PORTS {COM3, } / {COM3, } => {}
PORTS {COM3, } / {COM3, } => {}
PORTS {COM3, } / {COM3, } => {}
PORTS {COM3, } / {COM3, } => {}
PORTS {COM3, } / {COM3, } => {}
PORTS {COM3, } / {COM3, } => {}
PORTS {COM3, } / {COM3, } => {}
PORTS {COM3, } / {COM3, } => {}
PORTS {COM3, } / {COM3, } => {}
PORTS {COM3, } / {COM3, } => {}
PORTS {COM3, } / {COM3, } => {}
Uploading using selected port: COM3
C:\Users\brian\Documents\ArduinoData\packages\arduino\tools\bossac\1.9.1
An error occurred while uploading the sketch
```

2. At this point, you’ll want to look to the board itself. The orange LED opposite the green LED power indicator about the USB port should now be blinking!



As a final note, if you’d like to learn more about how the process of flashing code to a microcontroller works, please check out this appendix document.

Understanding the Code in the Blink Example

03_TinyML_C03_04-02-08-B_final-en.mp4 (Command Line)

Now that you have gotten the blink example deployed to your microcontroller, let's explore the code as shown below.

You’ll notice that it consists of two functions: setup, and loop. As we mentioned before, this is the standard setup for an Arduino sketch. This is because when the Arduino turns on it runs the setup() function ONCE to initialize (aka setup) the sketch. Then it runs the loop() function infinitely many times (aka it runs as an infinite loop) to execute the sketch. This works well for most tinyML applications as they are designed to respond to continuous sensor input. You can imagine in the case of Keyword spotting that we need to initialize the neural network and the microphone and then in a loop we want to listen to audio and trigger (or not) depending upon the output of the neural network!

```
// the setup function runs once when you press reset or power the board
void setup(){
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop(){
  digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000);                     // wait for a second
  digitalWrite(LED_BUILTIN, LOW);  // turn the LED off by making the voltage LOW
  delay(1000);                     // wait for a second
}
```

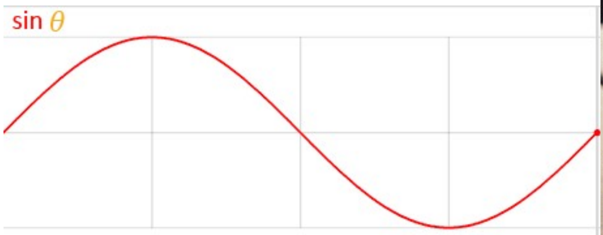
You'll also notice that both functions have the void return type as they do not ever return anything but instead have side effects.

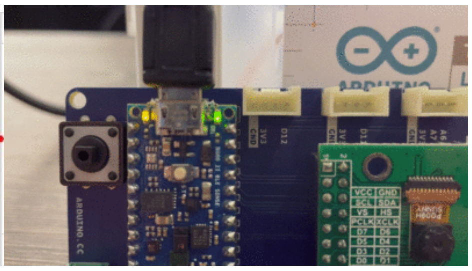
For example, in the setup function the LED_BUILTIN (which is a shortcut name for the pin that controls the voltage to the LED) is set to be an output for the duration of the loop. In general, you will need to set all of the pins you use as either inputs or outputs during the setup function. If you wired up the camera yourself, you have already explored a lot of the special names reserved for the pins as shown on the pinout diagram.

In the loop function, you'll notice that we are alternating between writing a HIGH (aka turning on the LED) and writing a LOW (aka turning off the LED). The delay of 1000 milliseconds (1 second) between each step is crucial as otherwise the light would turn on and off so fast that it would be imperceptible. In fact, if you make the delay too short the light will simply seem to be dim. This is a trick called Pulse Width Modulation that is actually used often in industry to e.g., control motors. If you'd like, feel free to experiment with modifying these delays and redeploying the code to see its effect!

Testing the Tensorflow Install

In this reading, we are going to walk through deploying the 'Hello World' example provided in the TensorFlow Lite for Microcontrollers (TFLM) library. The Hello World application runs a neural network model that can take a value, x, and predict its sine, y. Visually, the sine wave is a pleasant curve that runs smoothly from –1 to 1 and back. This makes it perfect for controlling a visually pleasing light show! We'll be using the output of our model to power a smoother blink of your Arduino LED.





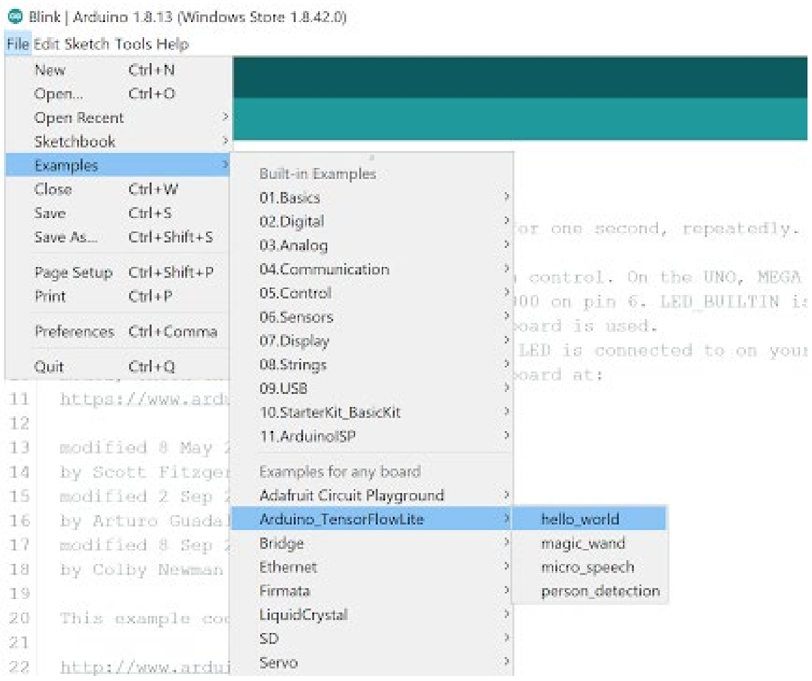
While the result won't differ greatly from what we've accomplished with standard Arduino Blink sketch we just deployed (in either case, an LED will be turning on and off), in this instance, we are going to be deploying a model. Previously, we have learned about how neural networks can learn to model patterns they see in training data to go on to make predictions. Here, the model we deploy will call on maps an input (x) to an output y = sin(x). While this is admittedly contrived, it will allow us to quickly evaluate if the TFLM stack is functional on the hardware in front of us by using this output y to control the intensity of light emitted by the on-board LED. As promised, below, you'll find instructions and screenshots and short videos walking you through the process!

TFLite Micro Hello World

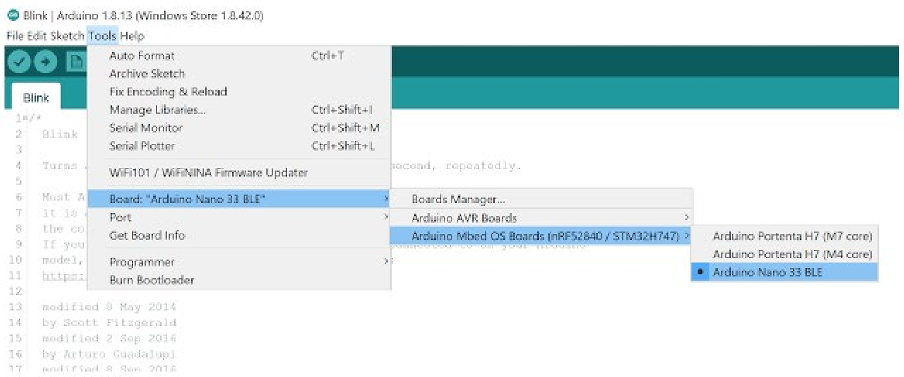
03_TinyML_C03_04-02-09-A_final-en.mp4 (Command Line)

Preparing for Deployment:

1. Use a USB cable to connect the Arduino Nano 33 BLE Sense to your machine.
2. Open the hello_world.ino sketch, which you can find via the File drop-down menu. Navigate, as follows: File → Examples → Arduino_TensorFlowLite → hello_world.

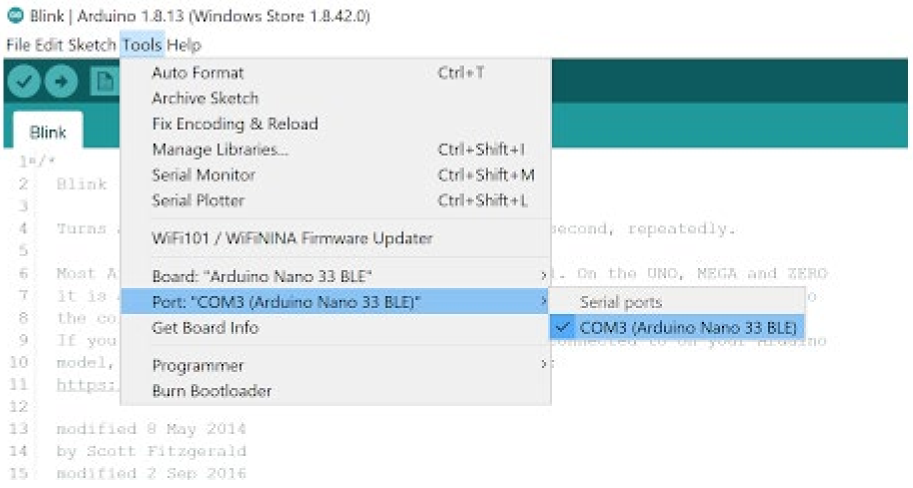


3. Just like with the Arduino Blink example (and as we will do for every deployment), use the Tools drop-down menu to select appropriate Port and Board.
   - Select the Arduino Nano 33 BLE as the board by going to Tools → Board: <Current Board Name> → Arduino Mbed OS Boards (nRF52840) → Arduino Nano 33 BLE. Note that on different operating systems the exact name of the board may vary but/and it should include the word Nano at a minimum. If you do not see that as an option, then please go back to Setting up the Software and make sure you have installed the necessary board files.

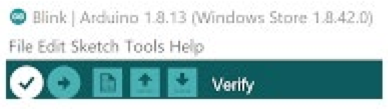


Blink | Arduino 1.8.13 (Windows Store 1.8.42.0)
File Edit Sketch Tools Help
Blink
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "Arduino Nano 33 BLE"
Port
Get Board Info
Programmer
Burn Bootloader
Boards Manager...
Arduino AVR Boards
Arduino Mbed OS Boards (nRF52840 / STM32H747)
Arduino Portenta H7 (M7 core)
Arduino Portenta H7 (M4 core)
● Arduino Nano 33 BLE

- Then select the USB Port associated with your board. This will appear differently on Windows, macOS, Linux, but will likely indicate 'Arduino Nano 33 BLE" in parenthesis. You can select this by going to Tools → Port: <Current Port (Board on Port)> → <TBD Based on OS> (Arduino Nano 33 BLE). Where <TBD Based on OS> is most likely to come from the list below where <#> indicates some integer number.
  - Windows → COM<#>
  - macOS → /dev/cu.usbmodem<#>
  - Linux → ttyUSB<#> or ttyACM<#>


Blink | Arduino 1.8.13 (Windows Store 1.8.42.0)
File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "Arduino Nano 33 BLE"
Port: "COM3 (Arduino Nano 33 BLE)"
Get Board Info
Programmer
Burn Bootloader
Serial ports
COM3 (Arduino Nano 33 BLE)

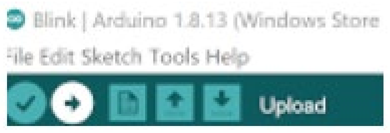
4. As always, it is best practice to then verify that the code is valid. Note that this may take a while the first time you run this as all of TFLM is being compiled (it will compile much faster in the future). Also, you may, or may not, see a series of compiling warnings (but not errors) as the code compiles, this is entirely normal either way and is the result of TFLM being bleeding edge software.


Blink | Arduino 1.8.13 (Windows Store 1.8.42.0)
File Edit Sketch Tools Help
Verify

Deploying (Uploading) the Sketch

Once we know that the code at hand is valid, we can flash it to the MCU:

1. Use the rightward arrow next to the 'compile' checkmark to upload / flash the code.

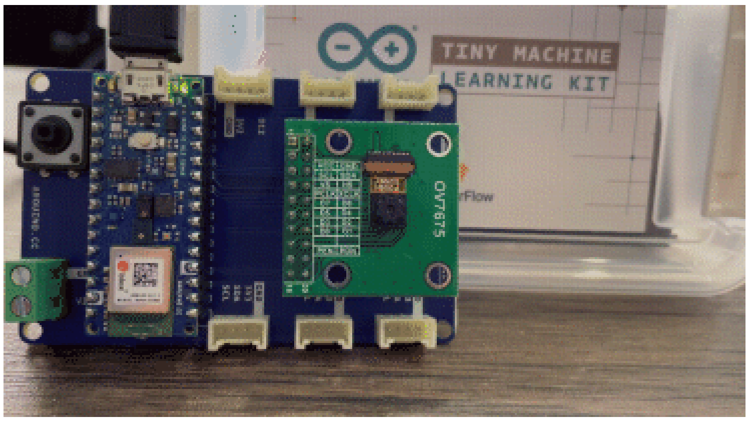

Blink | Arduino 1.8.13 (Windows Store
File Edit Sketch Tools Help
Upload

You'll know the upload is complete when you see red text in the console at the bottom of the IDE that shows 100% upload of the code and a statement that says something like "Done in <#.#> seconds."

If you skipped the verification step, do note that this may take a while the first time you run this as all of TFLM is being compiled (it will compile much faster in the future). Also, you may, or may not, see a series of compiling warnings (but not errors) as the code compiles, this is entirely normal either way and is the result of TFLM being bleeding edge software.

If you receive an error, you will see an orange error bar appear and a red error message in the console (as shown below). Don't worry -- there are many common reasons this may have occurred. To help you debug, please check out our [FAQ appendix]() with answers to the most common errors!

2. At this point, you'll want to look to the board itself. The orange LED opposite the green LED power indicator about the USB port should now be *fading* at the rate set by the sinusoidal model. Note that the default rate is quite fast so it should almost appear to be blinking very quickly. However, if you look closely you'll notice that it is fading in and out vs. blinking.



Congratulations! At this point, if all has gone well, you can be sure that your embedded microcontroller is configured properly to run TensorFlow Micro.

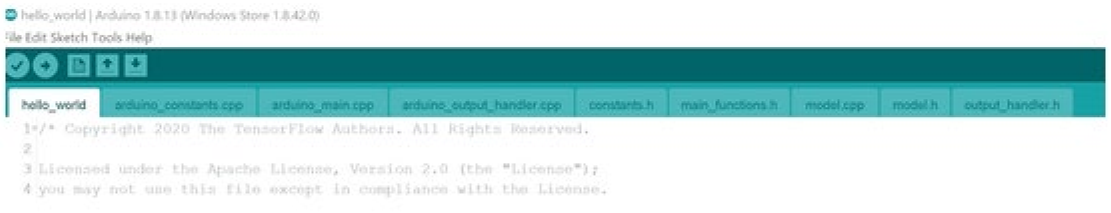
Understanding the TFLM Hello World Example

03_TinyML_C03_04-02-09-B_final-en.mp4 (Command Line)

The first thing you will notice when looking at this example is that there is a lot more code in this TFLM version of blink! However, at a high level the code is structured the same. There is still a setup() function that runs once to initialize things and a loop() function that runs continuously to blink the LED.

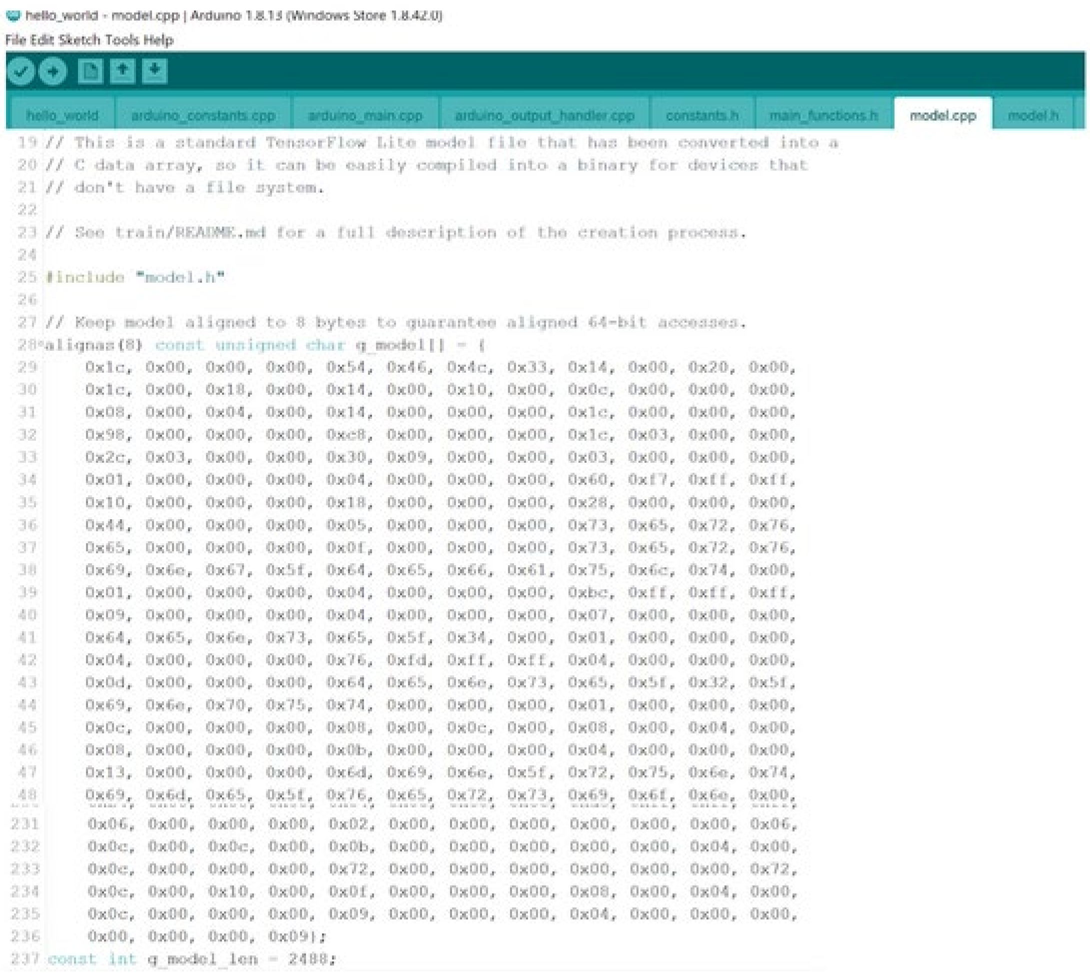
As far as the high level differences go, in the main hello_world file, before either of those functions, there are a set of #include lines and global variable definitions. The #include lines are used to (in Python speak) import the various libraries we need for this example. In this case it is the TFLM library and some other helper functions. The global variables are defined outside of the setup()and loop() functions, so that there are references to variables that can be used in both the setup() function as well as in every loop of the loop() function. For example, the global tensor_arena memory is used to initialize and store our neural network once and then is used to execute the neural network every loop iteration. This is much more efficient than re-initializing the neural network in each loop iteration.

One other high level difference is that you'll notice that there are a series of tabs across the top of the IDE window. These tabs each represent different files that make up this hello_world project. Each holds some helper functions or definitions that our main hello_world file uses.



```
/* Copyright 2020 The TensorFlow Authors. All Rights Reserved.

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
```

For example, the model.cpp file contains a large array which contains the binary data for the neural network model (in fact, it is simply the binary version of the TensorFlow Lite file) as well as an integer, which tells TFLM how long that array is. We'll explore how that binary file is generated later in the course!


hello_world - model.cpp | Arduino 1.8.13 (Windows Store 1.8.42.0)
File Edit Sketch Tools Help
hello_world
arduino_constants.cpp
arduino_main.cpp
arduino_output_handler.cpp
constants.h
main_functions.h
model.cpp
model.h

```
// This is a standard TensorFlow Lite model file that has been converted into a
// C data array, so it can be easily compiled into a binary for devices that
// don't have a file system.

// See train/README.md for a full description of the creation process.

#include "model.h"

// Keep model aligned to 8 bytes to guarantee aligned 64-bit accesses.
alignas(8) const unsigned char g_model[] = {
    0x1c, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x33, 0x14, 0x00, 0x20, 0x00,
    0x1c, 0x00, 0x18, 0x00, 0x14, 0x00, 0x10, 0x00, 0x0c, 0x00, 0x00, 0x00,
    0x08, 0x00, 0x04, 0x00, 0x14, 0x00, 0x00, 0x00, 0x1c, 0x00, 0x00, 0x00,
    0x98, 0x00, 0x00, 0x00, 0xc8, 0x00, 0x00, 0x00, 0x1c, 0x03, 0x00, 0x00,
    0x2c, 0x03, 0x00, 0x00, 0x30, 0x09, 0x00, 0x00, 0x03, 0x00, 0x00, 0x00,
    0x01, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0x60, 0xf7, 0xff, 0xff,
    0x10, 0x00, 0x00, 0x00, 0x18, 0x00, 0x00, 0x00, 0x28, 0x00, 0x00, 0x00,
    0x44, 0x00, 0x00, 0x00, 0x05, 0x00, 0x00, 0x00, 0x73, 0x65, 0x72, 0x76,
    0x65, 0x00, 0x00, 0x00, 0x0f, 0x00, 0x00, 0x00, 0x73, 0x65, 0x72, 0x76,
    0x69, 0x6e, 0x67, 0x5f, 0x64, 0x65, 0x66, 0x61, 0x75, 0x6c, 0x74, 0x00,
    0x01, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0xbc, 0xff, 0xff, 0xff,
    0x09, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0x07, 0x00, 0x00, 0x00,
    0x64, 0x65, 0x6e, 0x73, 0x65, 0x5f, 0x34, 0x00, 0x01, 0x00, 0x00, 0x00,
    0x04, 0x00, 0x00, 0x00, 0x76, 0xfd, 0xff, 0xff, 0x04, 0x00, 0x00, 0x00,
    0x0d, 0x00, 0x00, 0x00, 0x64, 0x65, 0x6e, 0x73, 0x65, 0x5f, 0x32, 0x5f,
    0x69, 0x6e, 0x70, 0x75, 0x74, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00,
    0x0c, 0x00, 0x00, 0x00, 0x08, 0x00, 0x0c, 0x00, 0x08, 0x00, 0x04, 0x00,
    0x08, 0x00, 0x00, 0x00, 0x0b, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00,
    0x13, 0x00, 0x00, 0x00, 0x6d, 0x69, 0x6e, 0x5f, 0x72, 0x75, 0x6e, 0x74,
    0x69, 0x6d, 0x65, 0x5f, 0x76, 0x65, 0x72, 0x73, 0x69, 0x6f, 0x6e, 0x00,
    0x06, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x06,
    0x0c, 0x00, 0x0c, 0x00, 0x0b, 0x00, 0x00, 0x00, 0x00, 0x00, 0x04, 0x00,
    0x0c, 0x00, 0x00, 0x00, 0x72, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x72,
    0x0c, 0x00, 0x10, 0x00, 0x0f, 0x00, 0x00, 0x00, 0x08, 0x00, 0x04, 0x00,
    0x0c, 0x00, 0x00, 0x00, 0x09, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x09};
const int g_model_len = 2488;
```

Getting back to the main hello_world file, as mentioned above, the setup() function has a decent amount of code in it, but at a high level, it does exactly what you would expect, it initializes our neural network.

The loop() function also has a decent amount of code in it, but again at a high level, all it does is keep track of a counter over time for the x-value and uses the neural network to compute the approximate y-value of y = sin(x) and then set the brightness of the LED according to that approximate y value.

Now, there are a lot of details that go into making all of those helper functions work, and excitingly, Pete Warden, the Technical lead of TensorFlow Mobile and Embedded team at Google will dig deeper into many of these topics in more detail later on. But before that, you'll need to run one more set of sketches to test your sensors!

Testing the Sensors

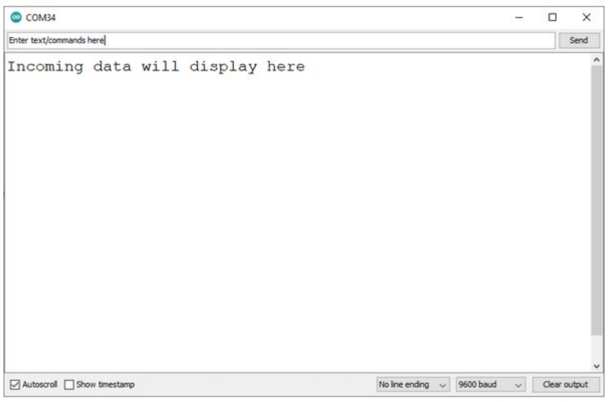
With tests of the microprocessor itself and TFLM stack completed, we can turn our attention to verifying that the sensors required for this course are functioning properly. To do so, we'll be calling on a predefined script that the course staff have put together. By the end of this reading, you will have seen live data streams for the on-board microphone, the STMicroelectronics, MP34DT05, the on-board internal measurement unit (IMU), the STMicroelectronics, LSM9DS1, as well as at least a static image return from the off-board OV7675 camera module, the OmniVision OV7675. We will also detail the optional extensions you'll need to do to obtain a live video feed from the camera as well.

The Serial Monitor & Plotter

The Serial Monitor & Plotter are two core tools that you can use to get information from your Arduino when it is connected to your computer with a data USB cable. In fact, as you develop your own Arduino applications, you'll probably find yourself opening up the Serial Monitor and using the Serial.println() command in your Arudino sketches much like how you have used the print() function in Python or printf() function in C++. Since the Arduino sketch runs in an infinite loop, you may find that it may be easier to plot the graph of the data you are sending back from the Arduino instead of continuously printing the raw values, and that is where the Serial Plotter will come in.

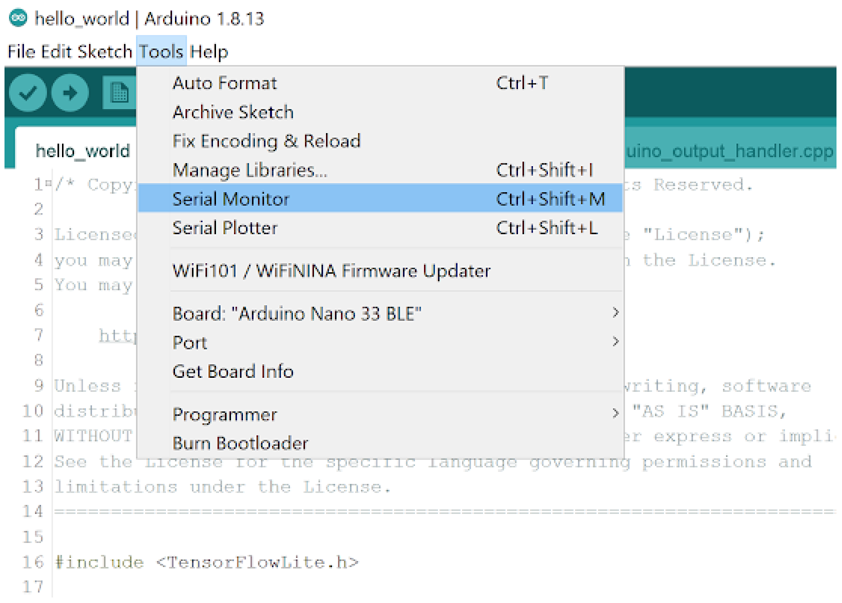
While the primary function of the Serial Monitor (and the Serial Plotter) is to view data incoming from the MCU, the Serial Monitor also features a text entry field that we will use to issue pre-determined commands conveyed in the console.

Below we have included a screenshot of the Serial Monitor elements to differentiate where we expect to see incoming data and where we can enter commands to send to the microcontroller:



```
Incoming data will display here
```

You can open up the Serial Monitor (and Serial Plotter) by navigating through the menu to Tools → Serial Monitor or Tools → Serial Plotter.


hello_world | Arduino 1.8.13
File Edit Sketch Tools Help
hello_world
uino_output_handler.cpp
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "Arduino Nano 33 BLE"
Port
Get Board Info
Programmer
Burn Bootloader

```
/* Copy                                    s Reserved.

License                                  "License");
you may                                  the License.
You may

    htt

Unless                                   writing, software
distrib                                  "AS IS" BASIS,
WITHOUT                                  er express or impli
See the License for the specific language governing permissions and
limitations under the License.
========================================================================

#include <TensorFlowLite.h>

```

Testing the Microphone

03_TinyML_C03_04-02-12-A_final-en.mp4 (Command Line)

1. Use a USB cable to connect the Arduino Nano 33 BLE Sense to your machine. You should see the green LED power indicator come on when the board first receives power.
2. Open the test_microphone.ino sketch, which you can find via the File drop-down menu. Navigate, as follows: File → Examples → Harvard_TinyMLx → test_microphone.
3. As always, use the Tools drop-down menu to select appropriate Port and Board.
   - Select the Arduino Nano 33 BLE as the board by going to Tools → Board: <Current Board Name> → Arduino Mbed OS Boards (nRF52840) → Arduino Nano 33 BLE. Note that on different operating systems, the exact name of the board may vary, but/and, it should include the word Nano at a minimum. If you do not see that as an option, then please go back to Setting up the Software and make sure you have installed the necessary board files.
   - Then, select the USB Port associated with your board. This will appear differently on Windows, macOS, Linux, but will likely indicate 'Arduino Nano 33 BLE" in parenthesis. You can select this by going to Tools → Port: <Current Port (Board on Port)> → <TBD Based on OS> (Arduino Nano 33 BLE). Where <TBD Based on OS> is most likely to come from the list below where <#> indicates some integer number.
     - Windows → COM<#>
     - macOS → /dev/cu.usbmodem<#>
     - Linux → ttyUSB<#> or ttyACM<#>

4. Use the rightward arrow next to the 'compile' checkmark to upload / flash the code.

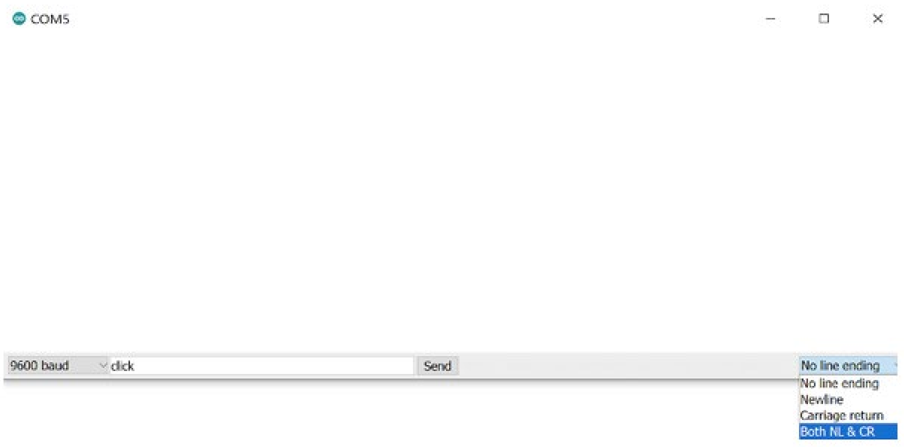
   You'll know the upload is complete when you see red text in the console at the bottom of the IDE that shows 100% upload of the code and a statement that says something like "Done in <#.#> seconds."

   If you receive an error, you will see an orange error bar appear and a red error message in the console (as shown below). Don't worry -- there are many common reasons this may have occurred. To help you debug, please check out our FAQ appendix with answers to the most common errors!

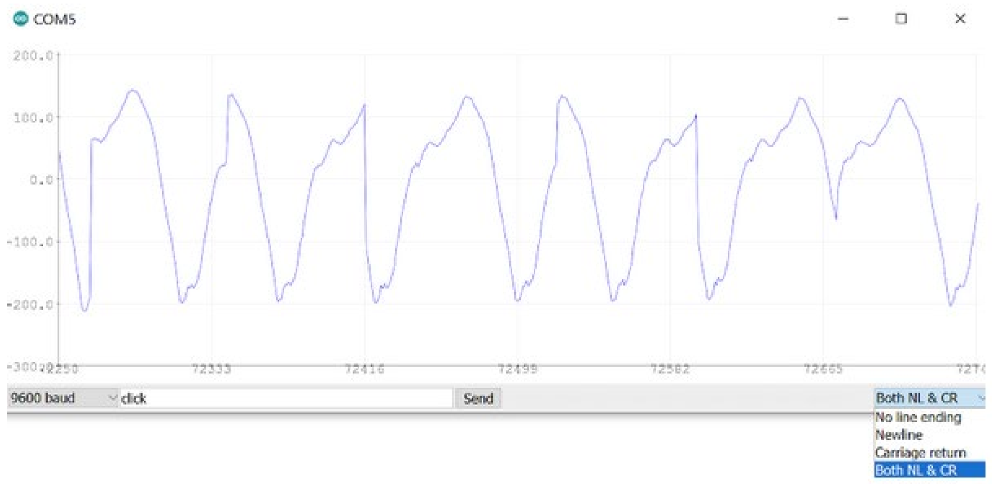
5. Open the Serial Monitor. You can accomplish this in three different ways as shown below. **If the Serial Monitor fails to open, check to make sure the appropriate Port is selected**. Sometimes, the port is reset during the upload process.
   - Use the menu to select: Tools → Serial Monitor.
   - Click on the magnifying glass at the top right of the Arduino Desktop IDE.
   - Use the keyboard shortcut: CTRL + SHIFT + M or CMD + SHIFT + M depending on your operating system.
6. When the Monitor opens, you should see the following text appear:

```
Welcome to the microphone test for the built-in microphone on the Nano 33 BLE Sense.

Use the on-shield button or send the command 'click' to start and stop an audio recording
Open the Serial Plotter to view the corresponding waveform.
```

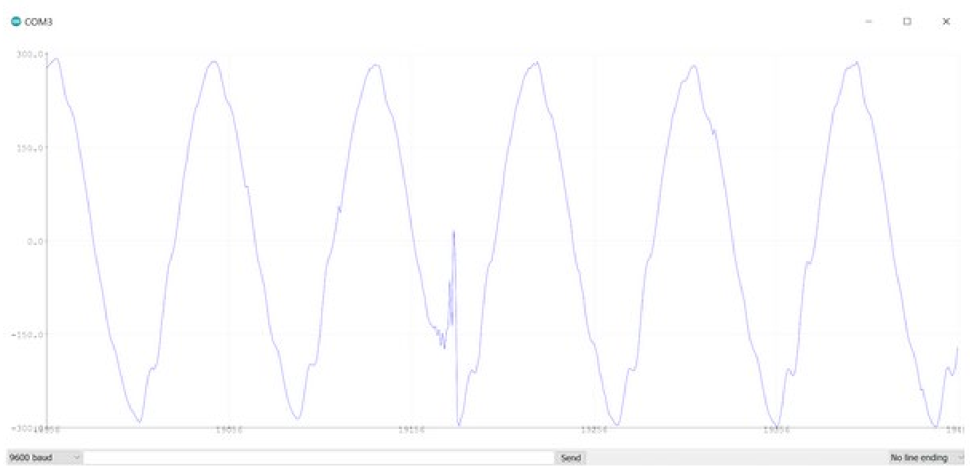
7. Press the on-shield button to start an audio recording. After you do, a stream of data should appear in the Serial Monitor. Press the button again to stop the recording. If the numbers were changing, then your microphone is most likely correctly recording audio! Congratulations! If you do not have the shield, you can also control the starting and stopping of the waveform by sending the command click through the serial monitor. In the drop-down menu that reads "No line ending" by default at the bottom right of the Serial Monitor, you need to select "**Both NL & CR**". This tells the Serial Monitor to send both a NL = new line character as well as a CR = carriage return character every time you send a message. This is important for our application as our Arduino sketch is looking for that to differentiate between each input.



8. To help visualize this a little better, we are going to use the Serial Plotter. **Note that you cannot have both the Serial Monitor and Serial Plotter open at the same time as the ARduino can only communicate over a single serial port. Importantly, this also means that you cannot upload new code to the Arduino if either the Serial Monitor or Plotter is open!** You can open the Serial Plotter in two ways:
    - Use the menu to select: Tools → Serial Plotter.
    - Use the keyboard shortcut: CTRL + SHIFT + L or CMD + SHIFT + L depending on your operating system.
9. Perform the same experiment, press the on-shield button (or send the serial command click) to start an audio recording and again to stop it. You should see a waveform appear on the Serial Plotter. This is simply a graphical representation of the numbers you saw earlier on the Serial Monitor. Note that the vertical axis of the Serial Plotter scales dynamically, so that the magnitude of the data conveyed is not fixed with respect to the enclosing window, but rather scaled to occupy most of the headroom.



10. Now, record some audio again. This time, try to whistle or hum a constant tone. If you can keep the tone constant, you should see the waveform start to look like a sinusoid. An example that the course staff made is below. Don't worry if you can't get this to work perfectly. You should in general find that a constant tone has a more constant pattern. If when you make different sounds and you find that the waveform changes, you can be confident that your microphone is working.



Testing the Inertial Measurement Unit

03_TinyML_C03_04-02-12-B_final-en.mp4 (Command Line)

1. Now, let’s open the test_IMU.ino sketch, which you can find via the File drop-down menu. Navigate, as follows: File → Examples → Harvard_TinyMLx → test_IMU.

2. As always, use the Tools drop-down menu to select appropriate Port and Board.

3. Then use the rightward arrow next to the ‘compile’ checkmark to upload / flash the code. If you get the error “Arduino_LSM9DS1.h: No such file or directory,” then please go back to Setting up the Software and make sure you install the accelerometer, magnetometry, and gyroscope library! To help you debug, please check out our FAQ appendix with answers to the most common errors!

4. Open the Serial Monitor and you should see the following. As a reminder, you can open the serial monitor in one of three ways as shown below. **If the Serial Monitor fails to open, check to make sure the appropriate Port is selected**. Sometimes, the port is reset during the upload process.
    - Use the menu to select: Tools → Serial Monitor.
    - Click on the magnifying glass at the top right of the Arduino Desktop IDE.
    - Use the keyboard shortcut: CTRL + SHIFT + M or CMD + SHIFT + M depending on your operating system.

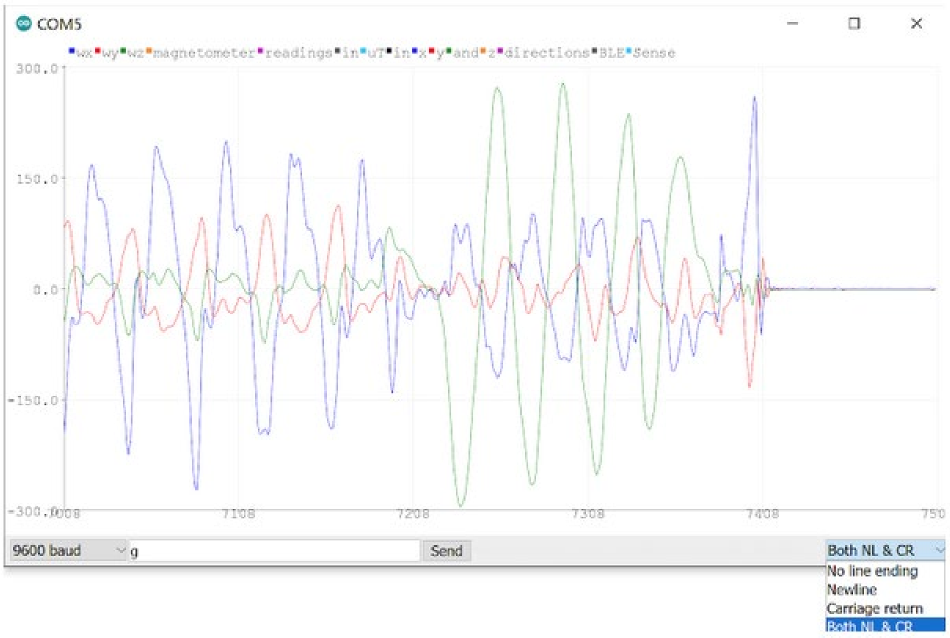
```
Welcome to the IMU test for the built-in IMU on the Nano 33 BLE Sense.

Available commands:
a - display accelerometer readings in g's in x, y, and z directions
g - display gyroscope readings in deg / s in x, y, and z directions
m - display magnetometer readings in uT in x, y, and z directions
```

5. Like with the microphone, in the drop-down menu that reads “No line ending” by default at the bottom right of the Serial Monitor, you need to select “**Both NL & CR**”.

6. Now, you can enter one of the possible arguments (a, g, or m) into the text entry bar across the top of the Serial Monitor and click “Send.” As mentioned in the Serial Monitor, this will start to print the data coming from the [a]ccelerometer (computing acceleration in the x, y, or z direction), the [g]yroscope (computing the angular velocity -- the change in the roll, pith, and yaw), or the [m]agnetometer (computing the magnetic forces present on the IMU). Depending on your selection, you should now see a stream of corresponding data. However, like with the microphone, this data is hard to interpret in raw form. So, instead, lets us open the Serial Plotter.

7. Close the Serial Monitor and open the Serial Plotter. As a reminder, you can open the Serial Plotter in two ways:
    - Use the menu to select: Tools → Serial Plotter.
    - Use the keyboard shortcut: CTRL + SHIFT + L or CMD + SHIFT + L depending on your operating system.

8. With the Plotter open, you should see the same stream of data presented graphically. The most interesting one to plot is the [g]yroscope. Again, make sure you have selected “Both NL & CR” and type “g” into the text entry box (now at the bottom of the window) and click “Send.”



9. Move the board around and observe the response. Can you figure out which axis of rotation is the x, the y, and the z (also known as roll, pitch, and yaw)? If by moving the board around in different directions you get different responses from the various lines in the Serial Plotter, you can feel confident that you have a working IMU.

Testing the OV7675 Camera

03_TinyML_C03_04-02-12-C_final-en.mp4 (Command Line)

1. Now, let's open the test_camera.ino sketch, which you can find via the File drop-down menu. Navigate, as follows: File → Examples → Harvard_TinyMLx → test_camera.

2. As always, use the Tools drop-down menu to select appropriate Port and Board. **Note**: if you did not purchase the Tiny Machine Learning Kit and sourced an OV7670, as compared to the OV7675 that comes with the kit, you will need to make one software changes as detailed at the end of this document.

3. Then, use the rightward arrow next to the 'compile' checkmark to upload / flash the code. If you get the error "Arduino_OV767X.h: No such file or directory," then please go back to Setting up the Software and make sure you install the accelerometer, magnetometry, and gyroscope library! To help you debug, please check out our FAQ appendix with answers to the most common errors! **Note:** if you get an error message that contains something like 'OV7675' was not declared in this scope this means you already had a conflicting copy of the Arduino_OV767X library installed on your system. We have bundled a forked copy of it with our library so please uninstall the conflicting version.

4. Open up the Serial Monitor and you should see:

```
Welcome to the OV7675 test

Available commands:

single - take a single image and print out the hexadecimal for each pixel
(default)
live - the raw bytes of images will be streamed live over the serial port
capture - when in single mode, initiates an image capture
```

5. Type "single" in the text entry field and press send or hit the enter / return key. The camera is now primed to take a picture. Now, you can either text "capture" or press the on-shield button to take a selfie (or photo). To get the photo oriented correctly, make sure the "OV7675" text on the camera module (or the "Tiny Machine Learning Shield" on the shield) is oriented such that it would be readable by the subject of the photo (e.g., you if you are taking a selfie). **The camera does not have a large field of view, so if you are taking a selfie, make sure to hold the camera far away from your face**.

6. To view your image, you will need to copy the sequence of hexadecimal digits that will print to the Serial Monitor and paste them into this Google Colab that we created to display the image (link included below as well). Note: in some versions of the IDE the large amount of digits printed will cause an error in the Serial Monitor and the text will seem to disappear. Don't worry the text is still there (its just white on a white background). Just highlight the whole line below "Image data will be printed out in 3 seconds..." and copy and paste it into the Colab and the digits will appear!

   https://colab.research.google.com/github/tinyMLx/colabs/blob/master/4-2-12-OV7675ImageViewer.ipynb

(Optional) Real-Time Video with Processing

If you are interested in seeing a live stream capture from the OV7675, you can do that, but not through the Arduino IDE. You'll have to use an additional piece of software, Processing. You can download the application for your OS, here (we've tested with version 3.5.4). We would like to note that we have found this to be buggy on Windows, which is why we provide this as an optional extension and not the default suggested process.

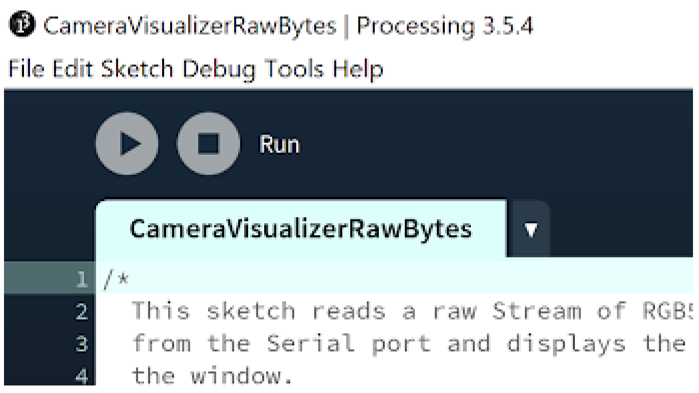
1. Launch the test_camera example through the Arduino IDE, open the Serial Monitor, and then type "live" in the text entry field and press send or hit the enter / return key. Your camera will now be live streaming the image to the computer at 1 frame per second. **Make sure to then close the Serial Monitor to free up the serial port as we will be using it again later.**
2. With Processing installed (which for some operating systems is as simple as unzipping the pre-built application folder), open the Processing Application. You'll find that the application looks a lot like the Arduino Desktop IDE.

3. Then find your Arduino folder. On most operating systems it is located either in your home or Documents directory.

4. In Processing, open the file Arduino → Libraries → Harvard_TiynMLx → extras → CameraVisualizerRawBytes → CameraVisualizerRawBytes.pde.

5. Go to lines 34-36 and update them to be the name of the serial port on which your Arduino is currently connected.

6. Now, click the "run" play button at the top left of the UI. A pop-up should appear streaming the camera image. Again, we have found this to be buggy on Windows, so don't be surprised if the image looks odd on Windows.


CameraVisualizerRawBytes | Processing 3.5.4

File Edit Sketch Debug Tools Help

```
/*
  This sketch reads a raw Stream of RGB5
  from the Serial port and displays the
  the window.
```

Changes for the OV7670

To use the OV7670 instead of the OV7675, simply change the fourth argument of the call to Camera.begin() on line 25 of the test_camera sketch, as highlighted below, from OV7675 to OV7670. That's it! The library will handle the rest!

```
// Initialize the OV7675 camera
void if(!Camera.begin(QCIF, RGB565, 1 OV7675)) {
   Serial.println("Failed to initialize camera");
   While(1);
}
```

Additional Resources

- If you'd like to learn more about debugging microcontrollers, we've put together a short guide in this appendix document.
- If you'd like to learn more about powering your Arduino using a battery, we've put together another short guide in this appendix document.

1.3: Embedded Hardware and Software

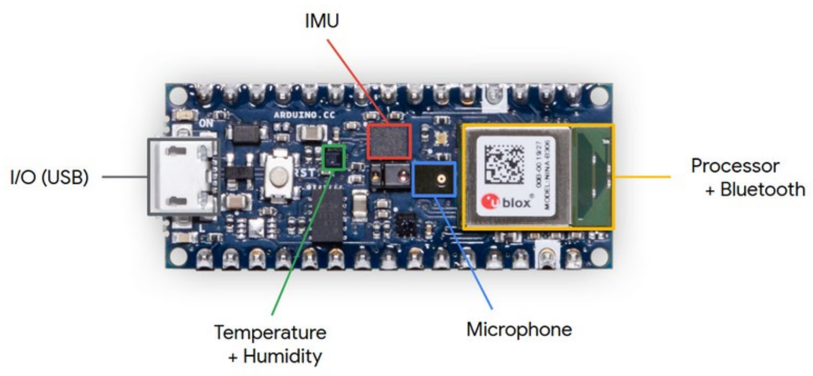
Embedded System

03_TinyML_C03_04-03-01_final-en.mp4 (Command Line)

In this reading, we will explore the diversity of embedded systems.

If you take a few minutes to look around the room you're in, you'll no doubt recognize the proliferation of embedded hardware within our world. Embedded systems are increasingly ubiquitous because they serve to complete specific computational tasks within their environment, allowing us to not only collect all sorts of data from distributed sensors but also to process the resulting information locally and act accordingly. That we can accomplish this in situ, or within the environment that this hardware is embedded within, at a fraction of the cost and physical scale of most general-purpose computing hardware, opens the door to many advancements in automation and generally to the creation of smart, connected things.

While the specifications and capabilities of a given embedded system ought to be tailored for its application, in general, we can look to the common construct for an embedded system that links sensing to processing and later actuation through the process of transduction, the conversion of one form of energy into another, for throughlines. In sense, we convert energy from a physical phenomenon into an electrical signal we can go on to digitize and compute with. In actuation, the converse. In this course, we'll focus predominantly on the central element of said construct: the computing hardware, and review specifications and technical considerations that vary between implementations of embedded systems.



Development Boards

In the context of this course, we have talked about and deployed a microcontroller unit (or MCU) development board from Arduino, the Nano 33 BLE Sense. Here, we distinguish the nRF52840 MCU from the development board it is soldered to. What's interesting about this board is that despite its relatively small size, it is jam-packed with many of the representative elements of an embedded system, all on one printed circuit board (PCB): a collection of surface-mount sensors, a form of an actuator in the on-board programmable RGB LED, and an integrated MCU-BLE module, where BLE is an acronym for a branch of the Bluetooth standard called Bluetooth Low Energy. Further still, the PCB that the MCU is soldered to serves to 'break out' additional IO from the controller to provide interfaces for external, off-board modules and connections to daughter boards. So, while in many ways, the Nano 33 BLE Sense is representative of an embedded system, it is separated from commercial implementations in that it serves no particular purpose, other than the open-ended development of an application that is. Put another way, a development board has the advantage of enabling many potential use cases. Further, you don't have to go through the process of designing circuitry, capturing schematics, or physical layout for such a board to get started prototyping a concept that could go on to be manufactured with specific intent later. The tradeoffs you make in employing a development board often involve board size and cost.

Board Size

Board size, and shape, have fairly meaningful implications for use in the field, given these simple parameters constrain where and how a system can be deployed. Some environments or contexts within which we'd like to embed a system are forgiving (like the Google Home, say), but other manifestations (smart glasses, for example) depend wholly on the requisite hardware being quite tiny and perhaps of unique form. Somewhat obviously, board size is determined by the number and physical size, or package, of the components that live upon it. At a minimum, an embedded system must include a MCU chip alongside a power source, often a lithium polymer battery, and power circuitry. The nRF52840 MCU on the Nano 33 BLE Sense lives within a MCU-BLE module, the U-Blox NINA-B306, which spans 10 by 15 mm, dimensions that include a trace antenna. The MPM3610 step-down converter used to down-regulate the 5V delivered to the board over USB occupies 3 by 5 mm, alongside small SMD passives. The entire board, meanwhile, spans 18 by 45 mm. While impressive, clearly a purpose-designed PCB could remove some of the sensors and IO breakout unnecessary for a specific application to reduce the overall scale even further, perhaps by about 60 to 70%.

Cost

Unsurprisingly, the cost of a development board scales with its MCU's compute capability and general feature set. In selecting the appropriate MCU for an application, it is important to remember that MCUs are often deployed to complete specific tasks, where the complexity they will face is predetermined or constrained. For very simple tasks, we highlight the ATTINY85 (an 8-bit processor with 8 kB of flash) that can, depending on the package selected, cost well less than $1 EA and even less at scale, perfect for simple computing requirements. In the context of TinyML, the AI-capable NINA-B306 module featuring the Nordic Semiconductor nRF52840 chip (an ARM Cortex M4) and all-in-one BLE hardware (including antenna) costs about $10 EA and is more generally representative. In view of the on-board sensors and design costs, the Arduino Nano 33 BLE Sense costs just over $30, not even an hour's time of an electrical engineer who might design such a board — a tremendous value. In general, the boards we originally considered for this class span from $2 (BluePill) to $54 (Disco-F746NG).

You can find a list of the boards we considered in the table below. Ultimately, our staff selected the Arduino Nano 33 BLE Sense for its versatility, in providing a large selection of on-board sensors, accessible IO via breakout pins, and a BLE module for projects that involve wireless communication, with reasonably representative compute specifications for resource-constrained hardware. We'll explore the compute specifications in a bit more detail in the next video and reading.

TinyML Development Board Comparison

▫officially TFLM supported ▢ unofficially compatible boards

Board	MCU	CPU	Clock	Memory	IO	Sensor(s)	Radio
Arduino Nano 33 BLE Sense	Nordic nRF52840	32-bit ARM Cortex-M4F	64 MHz	1 MB flash 256 kB RAM	x8 12-bit ADCs x14 DIO UART, I2C, SPI	Mic, IMU, temp, humidity, gesture, pressure, proximity, brightness, color	BLE
Espressif ESP32-DevKitC	ESP32 D0WDQ6	32-bit, 2-core Xtensa LX6	240 MHz	4 MB flash 520 kB RAM	x18 12-bit ADCs x34 DIO** UART, I2C, SPI	Hall effect, capacitive touch***	WiFi, BLE

Board	MCU	CPU	Clock	Memory	IO	Sensor(s)	Radio
Espressif EYE	ESP32 D0WD	32-bit, 2-core Xtensa LX6	240 MHz	4 MB flash* 520 kB RAM	SPI via surface pads	Mic, camera	WiFi, BLE
Teensy 4.0	NXP iMXRT1062	32-bit ARM Cortex-M7	600 MHz	2 MB flash 1 MB RAM	x14 10-bit ADCs x40 DIO** UART, I2C, SPI	Internal temperature, capacitive touch	None
MAX32630FTHR	Maxim MAX32620	32-bit ARM Cortex-M4F	96 MHz	2 MB flash 512 kB RAM	x4 10-bit ADCs x16 DIO UART, I2C, SPI	Accelerometer, gyroscope	BLE

*This board also features 4 MB flash and 8 MB of PSRAM external to the MCU, **shared programmable functions, ***with external touchpads

Board	ASIC	DSP	Clock	Memory	IO	Sensor(s)	Radio
Himax WiseEye WE-I Plus EVB	HX6537-A	32-bit ARC EM9D DSP	400 MHz	2 MB flash 2 MB RAM	x3 DIO I2C	Mic, accelerometer, camera	None

We include the Himax WiseEye as an officially supported example of hardware optimization for TensorFlow Lite that calls on an application-specific integrated circuit (ASIC).

Embedded Computing Hardware

03_TinyML_C03_04-03-03_final-en.mp4 (Command Line)

Diversity of Embedded Microcontrollers

Embedded Microcontrollers

In this reading, we will explore the diversity of embedded microcontrollers.

Microcontrollers sit at the bottom of the computing hardware hierarchy, at least in terms of computational capability. You might be wondering what differentiates an MCU from the CPU in the machine in front of you. Well, in some ways an MCU and CPU are quite similar, namely because an MCU is the integration of a low-performance CPU and other peripherals within a single chip. In this way, MCUs are also similar to, albeit less sophisticated than, a System-on-a-Chip (SoC). Typical peripherals integrated into an MCU include memory, analog-to-digital converters (ADC), digital-to-analog converters (DAC), timers, counters, general-purpose IO (GPIO), pulse-width modulation (PWM), direct memory access (DMA), interrupt managers, and serial protocol controllers. We can look at this integration and draw two conclusions: MCUs emphasize connections to the environments they live within and require that all on-board hardware be as tightly packaged as possible. Much like a CPU is seated upon a motherboard, MCUs are chips with packages soldered to much smaller PCBs that, at a minimum, must have supporting power circuitry and a programming interface. The latter prerequisite can be avoided if the MCU is programmed prior to assembly, but this is unusual, at least in development. Beyond this, the specifics of a custom embedded system are contingent upon the needs of the application. In leveraging a development board, you'll want to center your selection on the meeting, if not exceeding, the same requirements. We'll walk you through a comparison of MCU development boards and specifications in a moment.

To return to the computing hardware hierarchy: MCUs are clearly outstripped by the performance of your average personal computer, as well as computing clusters. They do fall somewhat adjacent to, if not arguably short of, what we'll call 'intermediate' computing in the form of single-board computers (SBCs), like the Raspberry Pi. SBCs and MCU development boards are similar enough conceptually and in terms of computing power but are differentiated in that SBCs aim to be low-power, single board computers with proper operating systems and IO typical of personal computers, but often lack the peripherals we've listed above, making them ill-suited in the context of embedded systems design. Looking at the computing hierarchy we've put forward (clusters > computers > intermediate computing > microcontrollers), you may wonder why exactly so much emphasis is placed on developing advanced software solutions for MCUs, like deploying deep learning models at the edge. Conventional wisdom would suggest that increasingly powerful computing hardware is required for and enables increasingly complex software applications. In recent years, however, there is an incentive for us to develop solutions that allow microcontrollers to take on such complex tasks to realize visions of distributed sensing and computation, where we need to strike a balance between cost (which can beget ubiquity), performance, and power efficiency. To put a name to such an incentive: we can compare the power required to transmit a stream of data from some remote controller to a server uplink and find that this energy requirement greatly outstrips the power we would need to perform a similar analysis on board. Further, there are complicated ethical and policy issues enveloping the deployment of distributed sensors that capture and then share streams of possibly identifying information, rather than de-identified reports of model returns. As such, we have seen MCUs take on increasingly complex tasks through the clever implementation of software, coupled with ever-advancing architectures that enable more powerful processing in even tinier packages.

Compute Capability

Microcontroller units have varied CPU architectures. Of these, chipmakers can license the 32-bit ARM Cortex M architecture, making modifications from this common starting point to introduce features and differentiate their products. For instance, the Nano 33 BLE Sense calls on a U-Blox NINA-B306, which is a stand-alone MCU-BLE module, that integrates the nRF52840 MCU from Nordic Semiconductor, which licensed the underlying architecture from ARM, as the nRF52840 is an ARM Cortex-M4. While this is generally representative of the modern controller, it's worth noting that there is a great deal of variation in compute capability, with processing cores that step all the way down to 8-bits, carrying modern relevance in simple applications.

As with other computing hardware, a faster MCU clock enables greater temporal performance, at the cost of diminishing power efficiency. Typical clock speeds sit between 50 and 500 MHz. For example, the Nano 33 BLE Sense is clocked at 64 MHz. We've deliberately said 'MCU clock,' here, rather than CPU clock, since the core clock of an MCU enables functions beyond just the CPU and trickles down to other peripherals (ADCs, DACs, et cetera) via scalars so that a faster clock can not only enable faster processing but also create headroom (bandwidth) for non-compute functions as well. For instance, while a less than 0.5 GHz CPU clock might not sound impressive, this can translate to theoretical sample intervals for analog-to-digital conversion on the order of tens of nanoseconds, which is more than sufficient for most physical phenomena of interest. Furthermore, lower speeds may actually be beneficial, somewhat counterintuitively, for timing longer intervals accurately.

Ultimately, the best processing core for your application will depend on the complexity of the task, its performance requirement, and temporal dynamics. One thing that MCUs are known for is their role in facilitating real or near real-time responses to events via interrupts and event systems. The latency of these responses will of course be tied to the performance specifications for the controller. Another thing that MCUs are known for is lying largely dormant in the environment until they are called upon. Dynamic clocking can ensure that an MCU sips as little current as possible until an interrupt wakes the MCU from this low-power mode to respond to whichever event. Consider a TV remote that sits at home while you're at work that can respond to a button press, wake its controller, and begin emitting the necessary sequence of electromagnetic pulses from its IR transmitter. The ability to adjust the system clock to reduce power consumption is a powerful technical feature — pun intended.

Memory & Storage

An important consideration to make when approaching MCU memory and storage for the first time is that the scales you may be used to working with on computers will likely not apply. Microcontrollers call on flash as a non-volatile option for program memory so that the very same program and initialization will occur at each reset. Typically, MCUs feature about 1 MB of program memory. Microcontrollers call on random access memory (RAM) to store working data, but this is of course volatile, so data from one power cycle cannot persist to the next. In the context of this course, it is interesting to consider the size of various models. One of the primary constraints for TinyML is ensuring that MCUs, as resource-constrained hardware, can fit a desired model in the first place. Compact speech recognition models, for instance, require 20-30 kB, whereas more complex models like those required for visual feature detection, require hundreds of kilobytes, really at the extreme of what most MCUs provide.

We want to highlight here that some microcontroller boards interface with SD cards as a way to extend the storage capacity of their system as well as to create a non-volatile record of data, for applications that require it. For what it's worth, EEPROM is another form of on-board or on-chip memory that some embedded applications (smart cards, for example) leverage to store programmable, persistent data.

In the end, you should consider the scale of your application and the role that volatility might play in limiting or enabling the function you require.

Embedded I/O

03_TinyML_C03_04-03-04_final-en.mp4 (Command Line)

Transducer Modules and Wireless Communication

While some of the TensorFlow Lite Micro capable development boards feature onboard sensors, others do not and you may find yourself interested in connecting an external sensor or actuator transducer module. Further, your use case may require you to periodically transmit results to other devices using wireless communication protocols like Bluetooth. In this reading, we briefly discuss each of these topics at a high level, leaving more thorough explanations to external readings you can complete at your discretion.

Sensor and Actuator Modules

When you write code to compute the average of a set of variables, you rarely if ever add the terms and divide by their number. Instead, you call on a pre-built average function. Much in the same way, many of the desired circuits required to achieve sensing and actuation of various kinds have been commoditized to the point where it isn't a question of designing these circuits again from first principles but interfacing the modules with an MCU using an API mindset if you will. The inputs and outputs of various modules are our concern.

We'll turn to consider possible interfaces in just a moment, but first, we want to highlight that there is a catalog of largely plug-and-play sensors and actuators that will work with the Arduino Tiny Machine Learning Kit, recalling that the role that each category plays in a canonical embedded system (sense → process → actuate), converting energy from and to a physical phenomenon, respectively, in a process called *transduction*.

The modules we mention are available from SeeedStudio, given their compatibility with the Grove connectors on the Tiny Machine Learning Shield and are as such the easiest way to extend the functionality of your system, but there's no reason why you couldn't leverage a solderless breadboard and some hookup wire to unlock compatibility with an enormous number of available modules. In the US, we like to source such modules from Adafruit and SparkFun, but they are widely available if you do some digging.

IO & Serial Protocols

A microcontroller has three main types of IO: digital, analog, and serial, listed in order of relative complexity. The foremost is very simple, as digital pins have only two states, high (or equivalently 1 or true) and low (or 0, false), and can be written to as outputs or read from as inputs. For clarity, these binary states are physically conveyed in voltage extremes, if you will: either 0V with respect to circuit "ground" or the full-extent of the logical reference voltage for the given microcontroller, for the nRF52840, this is 3.3V.

Analog inputs are a bit more complex in that they seek to translate a continuous representation of voltage in time and amplitude into a binary representation, that requires discretization and quantization, respectively. For more on analog-to-digital conversion (ADC) and digital-to-analog conversion (DAC), including approximations made by pulse-width modulation (PWM) for slow-response loads or circumstances, you can read on at the links above.

Finally, we turn our attention to serial communication protocols, of which we'll highlight three: universal asynchronous receive transmit (UART), inter-integrated circuit (I2C), and serial peripheral interface (SPI). We've provided a brief appendix document describing them in more detail and note at a high level that the various serial protocols are designed for transmitting longer data messages with different applications in mind (e.g., one to one vs. one to many communication). We also note that variants of the I2C protocol are used by both the IMU on your Arduino Nano 33 BLE Sense and between the Arduino and the OV7675 camera module.

Efficient Wireless Communication

A major incentive for deploying deep learning models locally within an embedded system is to avoid the energetic cost of transmitting a constant stream of data from a sensor via some wireless communication interface, like WiFi or Bluetooth. Indeed, in most cases, it is favorable to compute a result that you might later transmit periodically, cutting down on the overall energy expenditure. This paradigm fits well within the dogma that led to the division of Bluetooth specification in 2011 and the creation of a new fork dedicated to transmitting information as needed, rather than in a continuous stream, Bluetooth Low Energy (BLE).

There is a BLE module, the U-Blox NINA B306, on your Nano 33 BLE Sense development board, which we will use in the Magic Wand example later in this course! If you'd like to learn more about BLE you can check out this short appendix document we created.

Embedded System Software

.mp4

03_TinyML_C03_04-03-08_final-en.mp4 (Command Line)

Arduino Cores, Frameworks, mbedOS, and 'Bare Metal'

In working with the Nano 33 BLE Sense, you have interacted with some of the underlying layers between your application and the board's physical hardware. Here, we quickly define some key terms (Arduino core, embedded frameworks, mbedOS, RTOS, and bare-metal) before discussing the implications of these concepts to your application.

What is an Arduino Core?

An Arduino "core" is central to a particular microcontroller's compatibility with the Arduino framework. You can think of it as a low-level software API for a specific chip or family, like

AVR or SAMD. Since each microcontroller has its own architecture, the concomitant core acts as an abstraction layer between your application and the physical hardware. In this context, libraries can be thought of as extensions to the core that are portable between chips or board variants for a particular chip.

More granularly, each Arduino core contains a pair of generic files: arduino.h that handles declarations for what are often called ‘built-in’ functions like pinMode(), digitalWrite(), analogRead(), delay(), et cetera as well as macros for constants, like HIGH or INPUT_PULLUP, and operations like bitToggle() or abs(); main.cpp declares the basic program structure for a sketch, as it relates to setup() and loop(). We should note that arduino.h also sets the stage for board-specific pin mappings defined in pins_arduino.h at <architecture>/variants/<board name>.

The remainder of a core is largely architecture-specific. There is a collection of C files with names wiring.c and wiring_<type>.c that define hardware initialization, timing, and functions related to IO of various types, like analogRead(). This nomenclature is a nod to the Wiring API that Arduino utilizes. There are also the header and C++ files for serial classes Stream, Print, and HardwareSerial, alongside a light USB stack. If you’re interested in digging deeper into a specific core, check out this appendix document.

What is an Embedded Framework?

Loosely, to account for variation between manifestations, an embedded framework is a collection of development tools for microcontrollers that may or may not be portable and often serve as some combination of low-level API, software development kit (SDK), and hardware abstraction layer (HAL). For instance, the Arduino framework comprises its integrated development environment (IDE), built-in functions (defined by cores), as well as library extensions. As an open-source project, the Arduino framework is perhaps a bit more difficult to put bounds around, given its reliance on related projects, namely Wiring and its predecessor Processing, born out of the MIT Media Lab. Here are some other examples:

CMSIS - Arm Cortex Microcontroller Software Interface Standard

FreeRTOS - a real-time operating system (RTOS) kernel for embedded devices

Mbed - an embedded RTOS for Internet-of-Things, drivers for I/O devices

STM32Cube - HAL between STM32 devices and other libraries

If you work with more than one framework, we recommend PlatformIO, an extension of Visual Studio Code, that enables you to call on many frameworks, with a unified workflow.

What is an RTOS?

Okay, so two of the examples we provide for embedded frameworks make mention of a real-time operating system or RTOS. In a circular definition, an RTOS is an operating system that serves real-time applications. More helpfully, an embedded RTOS is extremely light-weight system software that organizes the attention of microcontroller processing cores, often singular, to minimize unnecessary task switching and to prioritize time-sensitive computation so that the processing time required for a given task is ideally less than the interval to the next input. We won't dwell on the details here, but you can learn more about task scheduling, apparent multitasking (threads), and other RTOS fundamentals, here.

What are Mbed and mbedOS?

Mbed is a competing embedded framework to Arduino, with a fair bit of overlap. A glance at the 'What is Mbed?' section of their landing page illustrates this. Both are centered around a C++ API and feature IDEs, example code, libraries, and drivers for common components. Perhaps most striking is the difference in accessibility and other professional-grade considerations Mbed makes, like security. Notably, while mbedOS can be and usually is an RTOS, for applications that require thread management, there is also a limited 'bare metal' profile that you can utilize to cut down on the memory utilization tied to RTOS overhead.

Why are we talking about mbedOS? Well, the Nano 33 BLE Sense runs on it. Here is an Arduino blog post from Martino Facchin, the head of Arduino's firmware development team, that explains the Nano 33 BLE Sense unique core, and its reliance on mbedOS.

What is Bare Metal?

You may happen upon variations of this definition, but at a high, inclusive level, 'bare metal' programming in an embedded systems context means utilizing microcontroller hardware independent of an operating system or scheduler, at the least. Typically, there is a base super loop, not unlike the Arduino sketch structure, and all processing is defined by your application. In many cases, this also means your code is written independently of pre-existing frameworks. This is where you'll find some variation in definitions. The colloquial spirit of bare-metal programming is counter to most developer mindsets, in that because microcontroller tasks are often simple but mission-critical, their developers want strict control over processing that precludes reliance on abstraction.

A 'true' bare-metal project, then, will have a developer that either writes or, if supplemented by a light-weight framework, fully understands each layer, from the physical hardware on up, using an MCU/SoC datasheet as their guide to registering definitions and hardware architecture.

Implications for your Application

As you can imagine, your choice of the framework (or lack thereof) should stem from the needs of your application. If you have no hard time-constraints, an RTOS could be eating away at precious resources that may be constrained. However, there are obvious challenges to developing a bare-metal solution, which can largely be summarized by 'recreating known solutions.' If your task is simple, time-insensitive, mission-critical, or needs to be deployed on very constrained hardware, a bare-metal approach may be appropriate, assuming you have the knowledge and time required for the development. If you have a complex web of tasks in front of you that could be organized by a scheduler, an RTOS is really a no-brainer. In most cases, there is room for innovation in the middle, where you place trust in proven frameworks and libraries, but manage processing attention in a simple base loop, perhaps organized as a finite state machine or FSM.

1.4: TensorFlow Lite Micro - Embedded ML Software

03_TinyML_C03_04-03-10_final-en.mp4 (Command Line)

What is TensorFlow Lite for Microcontrollers?

As a future TinyML engineer, it is essential to understand the inner workings of the software you use to know its capabilities and limitations. So, in the upcoming series of videos and readings, you will learn more about the challenges that led to the development of TF Lite Micro, straight from the source. Pete Warden from Google, who leads the team that works on TF Lite Micro, will introduce TF Lite Micro and give us a sneak peek into its internal workings. Here is a preview of what is to come next.

TensorFlow has become the most popular deep learning framework, superseding other popular frameworks such as PyTorch and Keras. TensorFlow, developed by Google, contains a Python frontend with highly optimized C++ code at its core, making it simple to program, fast, and efficient. The library has a large developer community and is now seen as the de facto standard for most machine learning applications.

Despite this, TensorFlow is not suitable for every scenario. The standard TensorFlow library is ~400 MB in size, and even running a relatively small model (e.g., 200 MB) can take up a considerable amount of random access memory (> 1 GB). Such large storage and memory requirements make running simple models on lightweight systems largely intractable.

Recognizing this issue, Google developed a more lightweight framework, TensorFlow Lite, also sometimes referred to as TensorFlow Mobile. The TFLite binary is approximately 1 MB in size, considerably more compact than the original library, making it possible to run deep learning models on mobile devices such as smartphones. This compression was achieved by removing superfluous functionality that is largely unnecessary for mobile deployment.

While this is an improvement, our problem still remains: even TFLite is not suitable for every scenario. Many important deep learning applications exist at the microcontroller-level, which are significantly more resource-constrained than mobile devices, often equipped with less than 1 MB of storage and 256 KB RAM. Clearly, deploying TFLite models is not feasible for microcontrollers, so an alternative solution was needed.

Enter TF Lite Micro. TF Lite Micro takes the compression of the TensorFlow library to the extreme, removing all but essential functionality. In fact, the core runtime of the library takes up only 16 KB, several orders of magnitude smaller than TFLite. With such a small memory footprint, this lightweight framework makes it possible to deploy deep learning models on the smallest of microcontrollers, such as an Arduino Nano.

However, this is not without its complications. Deploying models with TF Lite Micro is fraught with new and unique challenges when building models. For example, since all functionality for plotting and debugging is removed, troubleshooting model issues is difficult. Additionally, since many microcontrollers do not have floating-point units or use 8-bit arithmetic, the model weights and activations must be suitably quantized on the microcontroller system. Since model training requires near-machine precision to perform gradient descent, this largely precludes on-device training. Thus, TF Lite Micro models must first be trained on a device with greater computational resources before being ported to the microcontroller, adding an additional stage to the machine learning workflow.

Despite this, the benefits provided by TF Lite Micro - the ability to perform machine learning inference on microcontroller devices - far exceed the challenges, heralding a new era of machine learning that is often referred to as tiny machine learning.

TFMicro: The Big Picture

03_TinyML_C03_04-04-02_final-en.mp4 (Command Line)

TFLite Micro: Interpreter

03_TinyML_C03_04-04-05_final-en.mp4 (Command Line)

MCU Memory Hierarchy

Given the limited resources available on microcontroller units (MCUs), it is paramount for us as future tinyML engineers to understand the different types of on-device memory, including their interplay and performance capabilities. This understanding will become vital for optimizing machine learning models on given hardware architecture.

What is Memory?

Computer memory stores information as binary values, known as **bits**, which can take on a value or one or zero. However, storing a single bit is not particularly informative, and so multiple bits are often grouped together into quantities known as **bytes**, which consist of 8 bits. A byte can represent 256 distinct values, which can be used to represent anything from letters of the alphabet to the weights of a neural network.

Microcontrollers read from and write to memory using bytes, which are located by a **memory address**. This memory address is a hexadecimal value that tells the microcontroller where the memory it is looking for is located, much like a postal address. Microcontrollers have a limited amount of memory, which fundamentally limits the complexity of programs and computations they can run. Different forms of memory may also take longer to access or lose data when they are not constantly powered, leading to the use of multiple forms of memory for various purposes.

RAM and Flash Memory

The two most important forms of memory are **flash memory** and **random access memory (RAM)**. Our Arduino Nano 33 BLE Sense is equipped with 1 MB of flash memory and 256 KB of static RAM.

Flash memory is **non-volatile**, meaning that data persists even when the system is turned off. This memory is where we save our model weights and program code, and is what we are transferring when we “flash” our model onto the device. The process of saving to flash memory is slow and gradually degrades the memory over time. Consequently, to minimize this degradation, flash memory is effectively used as **read-only memory** and only overwritten during reprogramming, not when storing intermediate results during program execution.

RAM, however, is **volatile**, and thus data is lost when the device is powered off. Accordingly, RAM is used for the **temporary storage** of variables, such as input and output buffers, and intermediate tensors. RAM is much faster to read and write than flash memory, making it ideal as the primary memory used during program execution. RAM comes in two forms: static and dynamic.

Dynamic RAM (DRAM) uses a single transistor and capacitor to store a bit, but the capacitor quickly loses charge and must be periodically refreshed to prevent the stored information from being lost. **Static RAM** (SRAM) uses 6 transistors to store each bit but is able to maintain the stored information without refreshing. SRAM is more expensive due to the additional transistors needed but is faster and less power-intensive because no time or energy is spent performing a DRAM refresh between reading and write operations. These differences make DRAM a suitable candidate for main memory in modern computers, and SRAM more suitable for caches. In our microcontroller, we have access to SRAM and use it as our main memory during program execution.

The third type of memory is also used in microcontrollers but is often abstracted from the user; this type of memory is known as a **register**. Typically, special purpose registers exist that store various values needed to perform certain low-level computing functions, such as the program counter and stack pointer. In contrast, general-purpose registers are used to store values and memory addresses. It is unlikely you will need to play around with registers as a tinyML engineer unless you are working at the assembly code level.

Relevance to TF Lite Micro

Why do we care about introducing computer memory to you? Well, the size of your model weights and code cannot exceed the size of the available MCU flash memory (i.e., flash memory minus the 2 KB bootloader), and the size of input buffers and intermediate tensors cannot exceed the available SRAM on the device. These are hard constraints, and as tinyML engineers, we must be wary when working with large models that might exceed these memory constraints. In such cases, an MCU platform with additional resources or a simpler model may be necessary.

TFLite Micro: Model Format / FlatBuffer

.mp4

03_TinyML_C03_04-04-07_final-en.mp4 (Command Line)

TensorFlow Lite Flatbuffer Manipulation Example

In this Colab, we'll dig a little deeper into the TensorFlow Lite Flatbuffer file format. We'll load in the Flatbuffer library and the TensorFlow Lite Schema, which will allow us to access and directly modify the weights in the pretrained KWS model manually. You'll then get to see how your direct modifications impact the final model accuracy using the Speech Commands dataset!

https://colab.research.google.com/github/tinyMLx/colabs/blob/master/4-4-8-Flatbuffers.ipynb

TFLite Micro: Memory Allocation (a.k.a Tensor Arena)

.mp4

03_TinyML_C03_04-04-10_final-en.mp4 (Command Line)

TFLite Micro: NN Operator Support (OpsResolver)

.mp4

03_TinyML_C03_04-04-11_final-en.mp4 (Command Line)

TFLite Micro Developer Design Principles

There are four overarching design principles that TFMicro was built upon in order to address some of the challenges faced by developers when working with tinyML for embedded systems. This reading provides a synopsis of these core principles, as outlined in further detail in the TensorFlow Lite Micro paper.

Principle 1: Minimize Feature Scope for Portability

This principle proposes that an embedded machine learning (ML) framework should assume, by default, that the model, input data, and output arrays are in memory, and do not need to be loaded into memory. In addition, accessing peripherals, such as an on-device camera, should not be the job of the ML framework. These functions still need to be fulfilled, but principally should not be fulfilled by the ML framework.

While this may seem unimportant, some microcontrollers do not have memory management (e.g. malloc) and other capabilities. Thus, trying to accommodate all varieties of platforms would bloat the library in an attempt to provide sufficient portability. Fortunately, due to the self-contained nature of machine learning models, the model can be run on-device without the need to access peripherals and system functions.

Principle 2: Enable Vendor Contributions to Span Ecosystem

Embedded devices come in all shapes and sizes, and require kernels to perform tinyML functions. The more optimized these kernels are for a particular device, the better performance will be achieved. However, because of the many differences between device platforms, there is no one-size-fits-all optimization solution. Consequently, the TFMicro team by itself is unable to support the wide variety of platforms that may want to run tinyML, and thus, vendors with strong motivation (i.e., those involved in microcontroller development) are encouraged to contribute to help bridge the gap. These vendors often have little experience with deep learning, and thus, TFMicro must provide sufficient resources to allow these teams to easily contribute. One way this is accomplished is by encouraging vendors to submit to a library repository and to provide tests and benchmarks for vendors to assess their hardware performance.

Principle 3: Reuse TensorFlow Tools for Scalability

The third principle focuses on scalability. More than 1,400 operations (e.g. CONV2D) are supported by TensorFlow and other machine learning training frameworks. However, inference frameworks (i.e., those actually deploying the model) typically only support a fraction of these operations. For most use-cases, this will likely not cause issues since the most commonly used operations will likely be supported, but this inherent difference leads to a mismatch between the set of potential models produced by the training framework and the set of potential models that can be deployed by the inference framework.

An exporter is used to convert a model from a training framework, such as TensorFlow, to a model for an inference framework, such as TFLite or TFLite for Microcontrollers. This model can then be deployed directly to a device and run using the library interpreter. Often, the training and inference frameworks are developed by different entities, which can present difficulties for developers when there are compatibility issues between the various stages of the developmental pipeline. This may render otherwise functional models unusable when trying to be deployed to a client device, especially when the incompatibilities are abstracted in high-level libraries such as Keras.

Due to these concerns, the TFMicro developers decided to reuse as many TensorFlow tools available as possible to help minimize such complications and compatibility issues.

Principle 4: Build System for Heterogeneous Support

The last principle focuses on promoting a flexible build environment. There are a large number of different types of embedded devices that may wish to use tinyML, and thus TFMicro should be designed without preference to any particular platform. This prevents vendor lock-in and also attracts a larger developer ecosystem due to improved portability. To combat this, TFMicro prioritizes code that can be built across a wide variety of integrated development environments (IDEs) and toolchains.

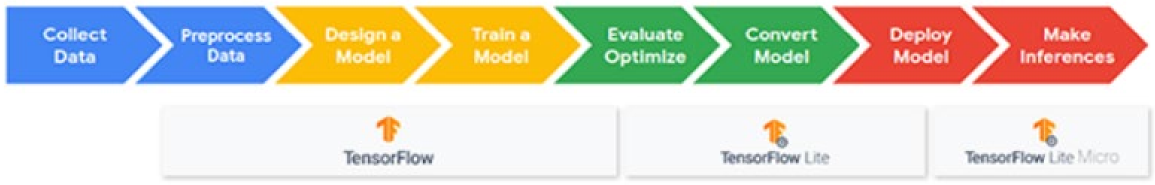
These four principles help to facilitate a developer ecosystem that is oriented towards maximizing portability between various hardware platforms, architectures, frameworks, and toolchains.

1.5: Keyword Spotting

TinyML "Keyword Spotting" Workflow

In Course 2, you were introduced to the machine learning workflow. In this reading, we will expand on this, introducing the additional steps necessary to get our real-world deployment of the "Keyword Spotting" application running on our own microcontrollers. The reading gives you an overview of what is to come, while the videos that follow will help you understand the challenges of performing ML deployment effectively on-device.

The first stages of the workflow are essentially the same as a traditional machine learning workflow: we collect and process our data, and then design and train a model for a specific machine learning task using the TensorFlow framework. The model will likely require an iterative design to meet performance goals. Following this, we move to TFLite to optimize our model (e.g., quantizing model weights into a suitable format) and then convert it into a suitable format, often a hexdump file (i.e., a binary file with its contents represented by hexadecimal values). Finally, we flash this hexdump file to the flash memory of the microcontroller, which is then able to perform inferences on the embedded device. The rest of this article will expand upon these stages with reference to our hello world application. The workflow is illustrated below.



Step 1: Data collection

As we explored in Course 2, collecting the dataset can often be the most challenging part of a tinyML application. Datasets must be large and specific in order to be applicable. For example, in the Keyword Spotting application we need a dataset **aligned** to individual words that include thousands of examples that are representative of real-world audio (e.g., including background noise). For our Keyword Spotting application, we will start off using the Speech Commands dataset from Pete Warden. Later on, we will show you how to collect your own data to train a new model.

Step 2 - Data Preprocessing

For efficient inference, we may need to **extract features** from the input signal to use for classification with our NN. First, we'll need to take any analog input signals collected from our sensors and convert them into digital signals. Then we may need to apply additional transformations. For example, in the Keyword Spotting application, we will convert the audio signals into images through the use of spectrograms, all on-device!

Step 3 - Model Design

In order to deploy a model onto our microcontroller, we need it to be **very small**. We explored the tradeoffs of such models and just how small they need to be (hint: it's tiny)! Since our Keyword Spotting application uses the "tinyConv" model, we should not need to worry about the size of the model, but parsimony should always be exercised when designing these models for the sake of efficiency - always have in mind, what is the smallest number of parameters that can be used to achieve a highly accurate model?

Step 4 - Training

When we build our own dataset, we will train our model using standard training techniques explored in Courses 1 and 2. Remember, training requires highly accurate computation (close to machine precision) since we are often working with small gradients during the training process, which largely precludes us from performing training on our embedded device. Because of this, we train our models using TensorFlow, as done previously.

Step 5 - Evaluation

We then test the final accuracy. In most industrial settings, initial testing is often poor and requires iterative design in order to match performance goals and the accuracy required by end-users. These performance goals are application-dependent and range from inference speed to model accuracy. For our Keyword Spotting application, we are only hoping to see a flashing light as our output, so there are no real performance goals beyond having a reasonable accuracy to detect a keyword of interest properly.

Step 6 - Conversion

In this stage, we take our functioning model and aim to compress and port it to our microcontroller device. First, we quantize our model weights and activations appropriately so that they can be represented by 8-bit or fixed-point arithmetic, and subsequently convert the model to a hexdump file. This stage is relatively simple to implement but is an important step to get right to ensure our model continues to function correctly when deployed on the microcontroller. The Keyword Spotting application needs no special treatment in this stage, all application models must go through this.

Step 7 - Write Deployment Code

Note that once we have our model ready, we also need all of the preprocessing code in C++ to be able to deploy our model to the Arduino with TFMicro. This involves the TFMicro runtime, as well as code to tell the Arduino how to respond to the information from the machine learning model. For our Keyword Spotting application, we have to tell it to turn on our LED and to report results over Serial!

Step 8 - Deploy the Full Application

The last thing to do is deploy the model. This stage involves loading the binary file into the application and using the Arduino IDE to flash the file to the microcontroller. Voila!

KWS Application Architecture

03_TinyML_C03_04-05-02_final-en.mp4 (Command Line)

KWS Initialization

03_TinyML_C03_04-05-03_final-en.mp4 (Command Line)

KWS Initialization Screencast

03_TinyML_C03_04-05-04_final-en.mp4 (Command Line)

KWS Pre-processing

03_TinyML_C03_04-05-06_final_2-en.mp4 (Command Line)

KWS Pre-processing Screencast

03_TinyML_C03_04-05-07_final-en.mp4 (Command Line)

KWS Inference

03_TinyML_C03_04-05-09_final-en.mp4 (Command Line)

KWS Inference Screencast

03_TinyML_C03_04-05-10_final-en.mp4 (Command Line)

KWS Post-processing

03_TinyML_C03_04-05-12_final-en.mp4 (Command Line)

KWS Post-processing Screencast

03_TinyML_C03_04-05-13_final-en.mp4 (Command Line)

KWS Summary

03_TinyML_C03_04-05-14_final-en.mp4 (Command Line)

Deploying the Pretrained KWS Model

In this reading, we are going to first generate a binary file representing the pretrained KWS model and then deploy that model to our Arduino using the Arduino IDE.

Converting the TFLite Model File into a Binary Array

The first step in deploying the pre-trained KWS model is to take the quantized TensorFlow Lite model you created in Course 2 and convert it into a binary array that we can load onto our microcontroller. And don't worry if you don't still have the .tflite file from Course 2; we have provided a direct link to the staff copy in the following Colab.

Please run this Colab and then, either leave the tab open with the printout of the final binary file and/or download the final .cc model file and open it up in a separate text editor. In either case, we will need to copy the data in that file into our Arduino sketch in the next section, so make sure you keep it in an easy to access place!

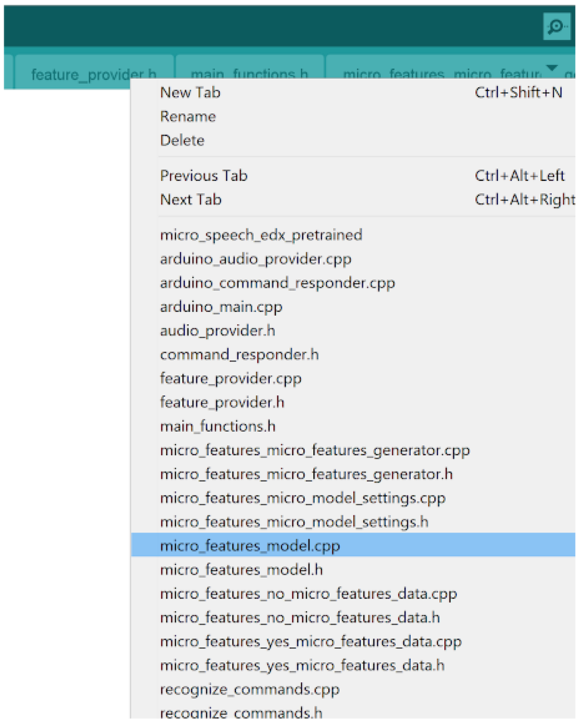
https://colab.research.google.com/github/tinyMLx/colabs/blob/master/4-5-16-KWS-PretrainedModel.ipynb

03_TinyML_C03_04-05-16_final-en.mp4 (Command Line)

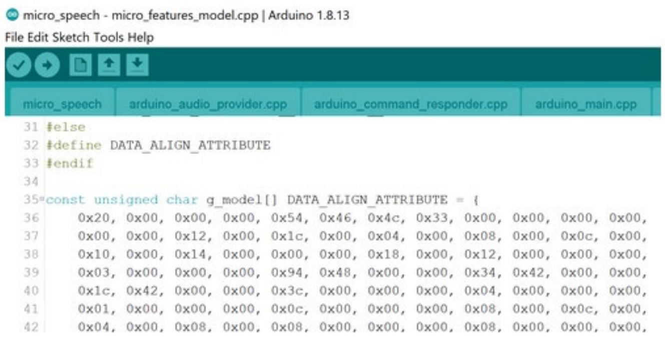
Updating the Arduino Code

1. Open the micro_speech.ino sketch, which you can find via the File drop-down menu. Navigate, as follows: File → Examples → Harvard_TinyMLx → micro_speech.

2. Navigate to the micro_features_model.cpp file and update the model. You can find that file by selecting it from tabs across the top of the Arduino IDE. If that file is not visible, you can navigate to it (or other additional files) by clicking on the downward facing triangle at the end of the tabs, which will open up a dropdown showing all of the files.

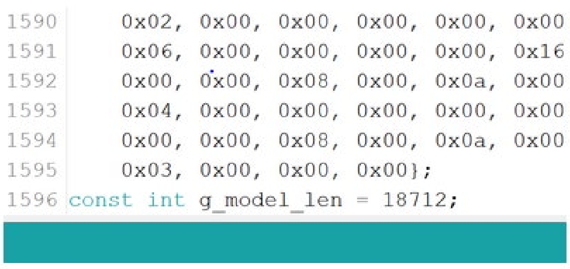

feature_provider.h
main_functions.h
micro_features_micro_featur
New Tab Ctrl+Shift+N
Rename
Delete
Previous Tab Ctrl+Alt+Left
Next Tab Ctrl+Alt+Right
micro_speech_edx_pretrained
arduino_audio_provider.cpp
arduino_command_responder.cpp
arduino_main.cpp
audio_provider.h
command_responder.h
feature_provider.cpp
feature_provider.h
main_functions.h
micro_features_micro_features_generator.cpp
micro_features_micro_features_generator.h
micro_features_micro_model_settings.cpp
micro_features_micro_model_settings.h
micro_features_model.cpp
micro_features_model.h
micro_features_no_micro_features_data.cpp
micro_features_no_micro_features_data.h
micro_features_yes_micro_features_data.cpp
micro_features_yes_micro_features_data.h
recognize_commands.cpp
recognize_commands.h

- Copy the binary model file contents from the KWS_yes_no.cc file into the micro_features_model.cpp file. **Make sure to only copy the binary data inside the { } as the variable type is different in the downloaded or printed KWS_yes_no.cc file** (it is of type unsigned char in the .cc file, but it needs to be of type const unsigned char with the DATA_ALIGN_ATTRIBUTE in the .cpp file). If you lost your KWS_yes_no.cc file, don't worry, you can use the staff's copy!


micro_speech - micro_features_model.cpp | Arduino 1.8.13
File Edit Sketch Tools Help
micro_speech
arduino_audio_provider.cpp
arduino_command_responder.cpp
arduino_main.cpp

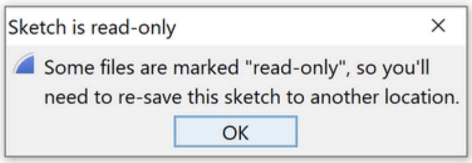
```
#else
#define DATA_ALIGN_ATTRIBUTE
#endif

const unsigned char g_model[] DATA_ALIGN_ATTRIBUTE = {
    0x20, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x33, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x12, 0x00, 0x1c, 0x00, 0x04, 0x00, 0x08, 0x00, 0x0c, 0x00,
    0x10, 0x00, 0x14, 0x00, 0x00, 0x00, 0x18, 0x00, 0x12, 0x00, 0x00, 0x00,
    0x03, 0x00, 0x00, 0x00, 0x94, 0x48, 0x00, 0x00, 0x34, 0x42, 0x00, 0x00,
    0x1c, 0x42, 0x00, 0x00, 0x3c, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00,
    0x01, 0x00, 0x00, 0x00, 0x0c, 0x00, 0x00, 0x00, 0x08, 0x00, 0x0c, 0x00,
    0x04, 0x00, 0x08, 0x00, 0x08, 0x00, 0x00, 0x00, 0x08, 0x00, 0x00, 0x00,
```

- Next, scroll all the way down to the bottom of the file and replace the model length. Again, note that the .cpp file needs the variable to be of type const int while the .cc file will show unsigned int. Our suggestion again is to simply copy the numerical value.



```
    0x02, 0x00, 0x00, 0x00, 0x00, 0x00
    0x06, 0x00, 0x00, 0x00, 0x00, 0x16
    0x00, 0x00, 0x08, 0x00, 0x0a, 0x00
    0x04, 0x00, 0x00, 0x00, 0x00, 0x00
    0x00, 0x00, 0x08, 0x00, 0x0a, 0x00
    0x03, 0x00, 0x00, 0x00};
const int g_model_len = 18712;
```

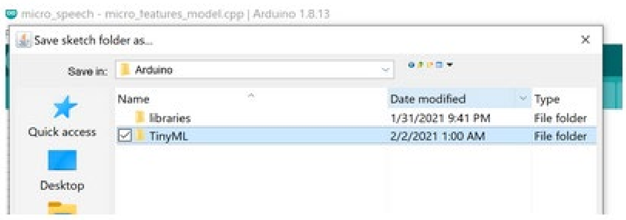
3. Next save your changes. You will most likely see a popup as shown below asking you to save a copy of the example as all examples are treated as "read-only" by default. treated as "read-only" by default.



Sketch is read-only

Some files are marked "read-only", so you'll need to re-save this sketch to another location.

We suggest that you make a folder called e.g., TinyML inside of your Arduino folder. You can find your main Arduino folder either inside of your Documents folder or in your Home folder, and save it in that folder with a descriptive name like micro_speech_edx_pretraied. That said, you can save it wherever you like with whatever name you want!



micro_speech - micro_features_model.cpp | Arduino 1.8.13

Save sketch folder as...

Save in: Arduino

Name	Date modified	Type
libraries	1/31/2021 9:41 PM	File folder
TinyML	2/2/2021 1:00 AM	File folder

Deploying the Pretrained Model

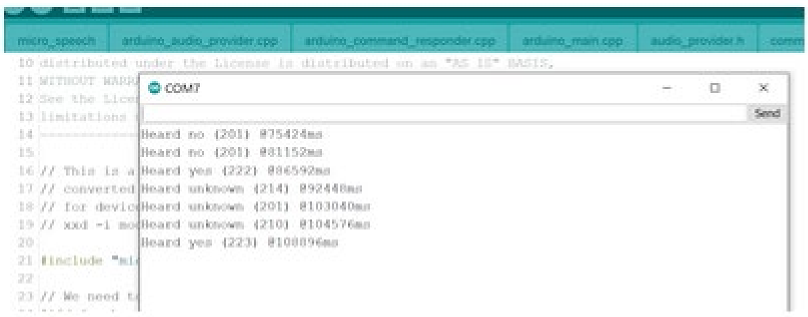
1. Use a USB cable to connect the Arduino Nano 33 BLE Sense to your machine. You should see the green LED power indicator come on when the board first receives power.
2. As always, use the Tools drop-down menu to select appropriate Port and Board.
    - Select the Arduino Nano 33 BLE as the board by going to Tools → Board: <Current Board Name> → Arduino Mbed OS Boards (nRF52840) → Arduino Nano 33 BLE.
    - Then, select the USB Port associated with your board. This will appear differently on Windows, macOS, Linux, but will likely indicate 'Arduino Nano 33 BLE" in parenthesis. You can select this by going to Tools → Port: <Current Port (Board on Port)> → <TBD Based on OS> (Arduino Nano 33 BLE). Where <TBD Based on OS> is most likely to come from the list below where <#> indicates some integer number.

- Windows → COM<#>
- macOS → /dev/cu.usbmodem<#>
- Linux → ttyUSB<#> or ttyACM<#>

3. Use the rightward arrow next to the 'upload' / flash the code. You'll know the upload is complete when you see red text in the console at the bottom of the IDE that shows 100% upload of the code and a statement that says something like "Done in <#.#> seconds."

   If you receive an error you will see an orange error bar appear and a red error message in the console (as shown below). Don't worry -- there are many common reasons this may have occurred. To help you debug, please check out our FAQ appendix with answers to the most common errors!

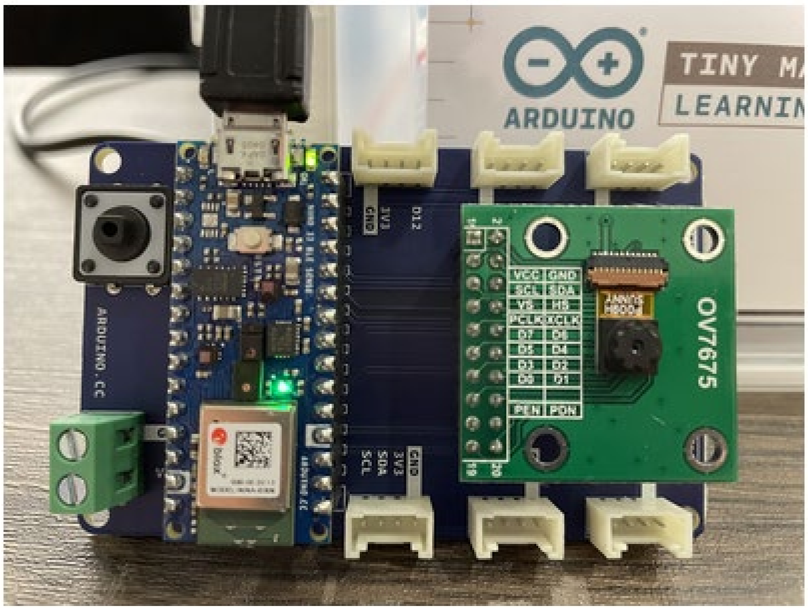
4. You should now have a working Keyword Spotting model for "Yes" and "No" deployed to your Arduino! To test it out, open the Serial Monitor. You should start to see the result of the model "Yes, No, Unknown, Silence" begin to display in the Serial Monitor.



```
Heard no (201) @75424ms
Heard no (201) @81152ms
Heard yes (222) @86592ms
Heard unknown (214) @92448ms
Heard unknown (201) @103040ms
Heard unknown (210) @104576ms
Heard yes (223) @108896ms
```

This should match the multicolor LED near the Bluetooth module on the board changing colors as follows (we've shown a picture of it turning green below):

Yes = Green LED
No = Red LED
Unknown = Blue LED
Silence = Nothing



Deploying a KWS Model with Your Favorite Keyword(s)

In this reading, we are going to first generate a binary file representing the KWS model with our favorite keywords from Course 2 and then deploy that model to our Arduino using the Arduino IDE (which will include some code changes).

Converting the TFLite Model File into a Binary Array

Again we are going to first convert the .tflite file into a .cc file in Colab for use with the Arduino IDE. This time let's use a model with different keywords (and a different number of keywords), so we can explore more of the changes you will need to make to deploy custom models. We've provided you with one option for a model in the Colab below. That said, we also invite you to use your favorite KWS model you created in Course 2 if you happen to still have the .tflite file! If you'd like to go back and train a new .tflite file, you can find the training Colab at this link.

As with the pretrained model, we will be using the resulting .cc file, so make sure to download it or leave the tab open with the printout!

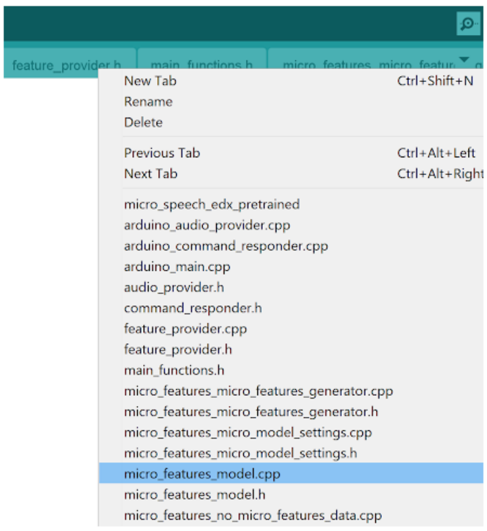
https://colab.research.google.com/github/tinyMLx/colabs/blob/master/4-5-18-KWS-FavoriteKeywords.ipynb

.mp4

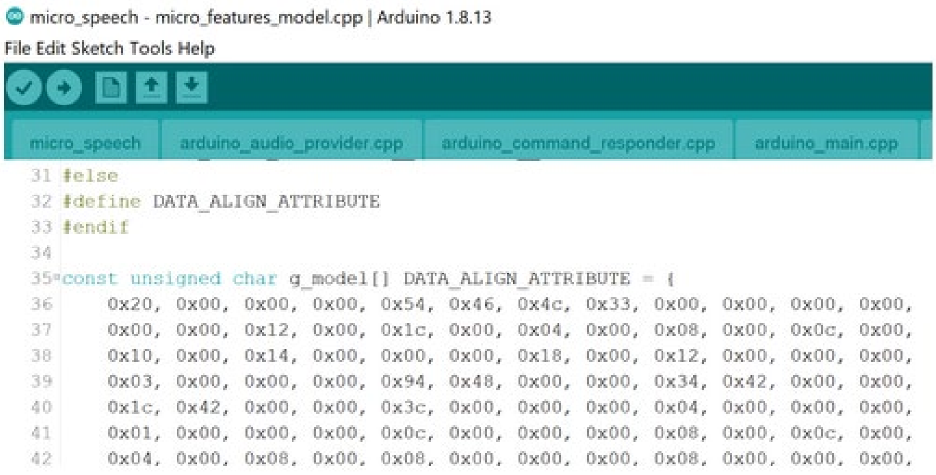
03_TinyML_C03_04-05-18_final-en.mp4 (Command Line)

Updating the Arduino Code

1. Open the micro_speech.ino sketch, which you can find via the File drop-down menu. Navigate, as follows: File → Examples → Harvard_TinyMLx → micro_speech.
2. As before, navigate to the micro_features_model.cpp file and update the model. You can find that file by selecting it from tabs across the top of the Arduino IDE. If that file is not visible you can navigate to it (or other additional files) by clicking on the downward facing triangle at the end of the tabs which will open up a dropdown showing all of the files.

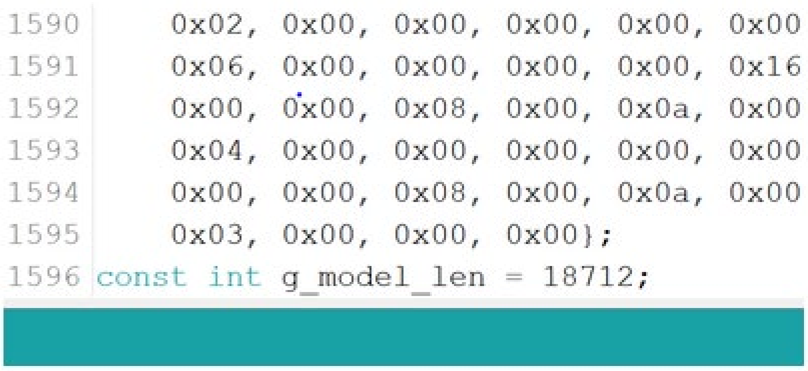

feature_provider.h
main_functions.h
micro_features_micro_featur
New Tab Ctrl+Shift+N
Rename
Delete
Previous Tab Ctrl+Alt+Left
Next Tab Ctrl+Alt+Right
micro_speech_edx_pretrained
arduino_audio_provider.cpp
arduino_command_responder.cpp
arduino_main.cpp
audio_provider.h
command_responder.h
feature_provider.cpp
feature_provider.h
main_functions.h
micro_features_micro_features_generator.cpp
micro_features_micro_features_generator.h
micro_features_micro_model_settings.cpp
micro_features_micro_model_settings.h
micro_features_model.cpp
micro_features_model.h
micro_features_no_micro_features_data.cpp

1. Copy the binary model file contents from the KWS_favorite.cc file into the micro_features_model.cpp file. **Make sure to only copy the binary data inside the { } as the variable type is different in the downloaded or printed model.cc file** (it is of type unsigned char in the .cc file, but it needs to be of type const unsigned char with the DATA_ALIGN_ATTRIBUTE in the .cpp file). If you lost your KWS_favorite.cc file, don't worry, you can use the staff's copy!


micro_speech - micro_features_model.cpp | Arduino 1.8.13
File Edit Sketch Tools Help
micro_speech
arduino_audio_provider.cpp
arduino_command_responder.cpp
arduino_main.cpp

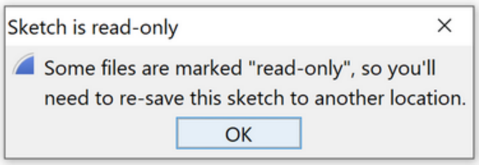
```
#else
#define DATA_ALIGN_ATTRIBUTE
#endif

const unsigned char g_model[] DATA_ALIGN_ATTRIBUTE = {
    0x20, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x33, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x12, 0x00, 0x1c, 0x00, 0x04, 0x00, 0x08, 0x00, 0x0c, 0x00,
    0x10, 0x00, 0x14, 0x00, 0x00, 0x00, 0x18, 0x00, 0x12, 0x00, 0x00, 0x00,
    0x03, 0x00, 0x00, 0x00, 0x94, 0x48, 0x00, 0x00, 0x34, 0x42, 0x00, 0x00,
    0x1c, 0x42, 0x00, 0x00, 0x3c, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00,
    0x01, 0x00, 0x00, 0x00, 0x0c, 0x00, 0x00, 0x00, 0x08, 0x00, 0x0c, 0x00,
    0x04, 0x00, 0x08, 0x00, 0x08, 0x00, 0x00, 0x00, 0x08, 0x00, 0x00, 0x00,
```

2. Next, scroll all the way down to the bottom of the file and replace the model length. Again, note that the .cpp file needs the variable to be of type const int while the .cc file will show unsigned int. Our suggestion again is to simply copy the numerical value.



```
    0x02, 0x00, 0x00, 0x00, 0x00, 0x00
    0x06, 0x00, 0x00, 0x00, 0x00, 0x16
    0x00, 0x00, 0x08, 0x00, 0x0a, 0x00
    0x04, 0x00, 0x00, 0x00, 0x00, 0x00
    0x00, 0x00, 0x08, 0x00, 0x0a, 0x00
    0x03, 0x00, 0x00, 0x00};
const int g_model_len = 18712;
```

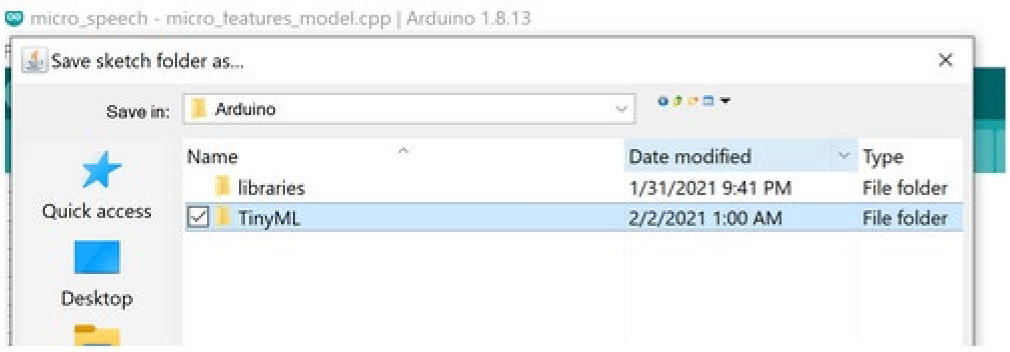
3. Next, save your changes. You will most likely see a popup as shown below asking you to save a copy of the example as all examples are treated as "read-only" by default.



Sketch is read-only

Some files are marked "read-only", so you'll need to re-save this sketch to another location.

Again, we suggest that you make a folder called e.g., TinyML inside of your Arduino folder. You can find your main Arduino folder either inside of your Documents folder or in your Home folder, and save it in that folder with a descriptive name like micro_speech_favorite. That said, you can save it wherever you like with whatever name you want!



micro_speech - micro_features_model.cpp | Arduino 1.8.13

Save sketch folder as...

Save in: Arduino

Name	Date modified	Type
libraries	1/31/2021 9:41 PM	File folder
TinyML	2/2/2021 1:00 AM	File folder

4. The two other things we need to change are the count and list of keywords in the model settings files and the output response file. As we do that in the rest of this document, we are going to assume that you used our model with the Keywords "up,down,go." If you choose to use different keywords, make sure that you update the following steps accordingly.
5. Navigate to the micro_features_micro_model_settings.h file and scroll down to line 40 and change the value of the kCategoryCount variable to be two more than the

number of keywords you selected (one extra for silence and one extra for unknown). For example, if you had 3 keywords (as we do in our example model), we need to update the variable to be 3 + 2 = 5 as shown:

```
constexpr int kCategoryCount = 5;
```

6. Then, navigate to the micro_features_micro_model_settings.cpp file. You'll find that it includes the .h file and otherwise has a single array. Update the values of that array to match your keywords. **Make sure to leave "silence" and "unknown" in the list as well, and make sure to list your keywords in the order that you listed them on the training script**. So, in the case of our example model, we would update the array to be:

```
const char* kCategoryLabels[kCategoryCount] = {
    "silence",
    "unknown",
    "up",
    "down",
    "go",
};
```

7. Finally, navigate to the arduino_command_responder.cpp file. Scroll down to line 54. There, you will see a series of if statements that control how the Arduino responds to each command. They are each structured to look at the value of the found_command variable. This variable will be the string of the keyword (or "silence" or "unknown") you entered into the kCategoryLabels array in the previous step. So, by comparing the [0] value in that string, the if statements are simply looking at the first letter! In each if statement, you'll also see a digitalWrite to either the LEDG, LEDR, or LEDB variable which is used to turn on the appropriate color LED. So, if we wanted to adapt this to our three keywords and, for example, only turn on the single colors for our keywords and turn all of the colors on for unknown, which will come out look white, we could update those three if statements to the following:

```
// Red for up -- note here that you do not need to index
//            into the first letter only, just a unique
//            letter combination in the keyword! That
//            said make sure you do not index beyond the
//            end of ANY keyword or you will get an error!,
if(found_command[1] == 'p') {
  last_command_time = current_time;
  digitalWrite(LEDR, LOW);
}
// Green for down
if(found_command[0] == 'd') {
  last_command_time = current_time;
  digitalWrite(LEDG, LOW);
}
```

```
// Blue for go
if(found_command[0] == 'g') {
  last_command_time = current_time;
  digitalWrite(LEDB, LOW);
}
// All three for unknown (white)
if(found_command[1] == 'n') {
  last_command_time = current_time;
  digitalWrite(LEDR, LOW);
  digitalWrite(LEDG, LOW);
  digitalWrite(LEDB, LOW);
}
```

Deploying the New Model

1. Use a USB cable to connect the Arduino Nano 33 BLE Sense to your machine. You should see the green LED power indicator come on when the board first receives power.
2. As always, use the Tools drop-down menu to select the appropriate Port and Board.
   - Select the Arduino Nano 33 BLE as the board by going to Tools → Board: <Current Board Name> → Arduino Mbed OS Boards (nRF52840) → Arduino Nano 33 BLE.
   - Then, select the USB Port associated with your board. This will appear differently on Windows, macOS, Linux, but will likely indicate 'Arduino Nano 33 BLE" in parenthesis. You can select this by going to Tools → Port: <Current Port (Board on Port)> → <TBD Based on OS> (Arduino Nano 33 BLE). Where <TBD Based on OS> is most likely to come from the list below where <#> indicates some integer number.
     - Windows → COM<#>
     - macOS → /dev/cu.usbmodem<#>
     - Linux → ttyUSB<#> or ttyACM<#>
3. Use the rightward arrow next to the 'upload' / flash the code. You'll know the upload is complete when you see red text in the console at the bottom of the IDE that shows 100% upload of the code and a statement that says something like "Done in <#.#> seconds."

   If you receive an error, you will see an orange error bar appear and a red error message in the console (as shown below). Don't worry -- there are many common reasons this may have occurred. To help you debug, please check out our FAQ appendix with answers to the most common errors!

4. You should now have a working Keyword Spotting model for either your favorite keywords from Course 2 or our model for "up,down,go" deployed to your Arduino! To test it out, open the Serial Monitor. You should start to see the result of the model begin to display in the Serial Monitor and the LED's light up according to how you instructed them in the previous section!

1.6: Custom Dataset Engineering for Keyword Spotting

Recap: Dataset Engineering

In this module, you will learn about the nuances of data engineering, and develop your own dataset which you will use to train and deploy a custom keyword spotting model!

Within these courses so far, we have used numerous datasets to train our algorithms. The existence of a dataset spared us from having to tackle the most time-consuming components of the machine learning pipeline: the collection and wrangling of data. In this section of the course, we are going to create our own dataset to perform keyword spotting, and expose you to some of the challenges that come along with the process, such as dataset bias, the need for edge cases, and the difficulty associated with procuring large quantities of data. This procedure is aptly named **dataset engineering**.

Data engineering is an important aspect of supervised learning and entails the specification of the dataset, as well as the procedures for collecting and processing data. Naturally, this involves defining what features will be present, the structure of the data (e.g., images, time series, row tuples), and other characteristics.

One of the key features of data engineering is identifying potential sources of data, which may come from external sources (free or purchased), crowdsourcing, existing users of a service, sensors, or other means. When developing our dataset, it is imperative that we consider our use case at all times, since this is our ultimate goal.

A natural corollary of this is that we must determine the environment our device will be in once deployed. For example, if an algorithm processing sound data expects to be used on the street, it is necessary to make recordings in the presence of adversarial sounds such as construction work, loud vehicles, as well as other background noise. We may have to decide what time of day images are taken, and also make sure that each of the classes in our dataset has roughly equal representation to prevent biasing to the majority class.

Another important concern when performing data engineering is data provenance. We must be cognizant of the origin of our data, most notably when we are using information procured from an external source, such as images from the internet. These images may be subject to copyright laws and other restrictions related to privacy. To preclude the possibility of legal issues, the origin of data should also be assessed, consulting with domain experts where necessary.

Certain types of data may require labeling, which may be done using crowdsourcing platforms such as Amazon Mechanical Turk in some cases. However, in other cases, domain experts may be required to label data. For example, radiologists may be necessary to assess x-rays or mammograms for abnormalities that would be vague or unnoticeable to the layman.

A further concern is the shifting of our objectives. If a plan is not sufficiently detailed or implemented correctly, dataset creep may occur, wherein more features are added in an attempt to improve the utility of the dataset. Alternatively, certain feature variables may be omitted or inputted incorrectly, such as with incorrect labeling.

Clearly, there are many ways in which the engineering of a dataset can go awry, and it is our job as a data engineer to ensure that the dataset has been properly collected with consideration of the deployment environment, as well as considerations of labeling, dataset bias, copyright, and other aforementioned concerns.

In the remainder of this section, you will learn to design, collect and develop your own dataset for the creation of a custom keyword spotting model. Good luck!

Introduce Custom Dataset for KWS

03_TinyML_C03_04-06-02_final-en.mp4 (Command Line)

Things to Consider for your Data Collection Plan

As you make a plan to collect data for a custom keyword spotting application, there are several important data engineering questions to consider in designing your data collection and testing scheme:

- Who are the anticipated end-users?
    - What will the age range be?
    - What languages will the user be speaking? Will the users have accents?
- What are the goals for using the application? How will this impact the requirements/needs of the ML model's performance?
    - For example, will the user be trying to turn specific lights on/off vs. turn the thermostat up/down vs. arm/disarm a security system vs play your favorite playlist from a smart speaker, etc.?
- What false positive or false negative rates can the application handle / what would the consequences be of such an event?
- In what environments will the users employ the application?
    - How much background noise do we expect?
    - How far will the users be from the device, and from noise sources in the environment?
    - Will these values vary over time?
- In what situations will the users use the application? Will the user’s speech sentiments vary?
    - Will the user be stressed vs. calm vs. panicked?
    - Will the user use different volumes of voice (whispered/normal/loud/shouted)?
    - Will different users have different sentiments?
- How do all of these previously mentioned factors affect how much and what kinds of target, non-target, and imposter/adversarial data you will need to train with?
    - How likely is it that these wake words will be triggered unintentionally during a conversation?
    - Who/how many people will be talking in the application environment?

- Given all of these previously mentioned factors, what will you collect?
  - What custom wake words will you select?
  - Are the background noise samples from the Speech Commands dataset sufficient for model robustness? Would it be helpful to collect additional background noise samples from specific environments?
  - Do you need to collect other words to ensure the model learns the difference between them and your wake words?
  - How much data will you need to collect? We all know more is better (usually), but you also live in the real world with time constraints so how much do you think will be enough?
  - In what environments will you collect these samples?
- How will you test your model?
  - What words will you try? How many times will you try these words? Will any of them be adversarial? Remember even current production systems struggle with close words at times!
  - Who will test the model? How diverse will the testers be?
  - Will the testers vary their speech sentiments?
  - Where will the testers be in relation to the device?
  - In what environments will the testing happen?
- Finally, based on your initial testing, how will you improve your results:
  - What didn't work the first time?
  - Will you need more or different data?
  - How can you iteratively improve your results?

Building a Custom Dataset

Collecting a Custom Dataset for KWS

NOTE: Our URL has changed from the video! See below for details!

There is another change from the video below -- please only record 10 words per recording session as Chrome will only allow you to download 10 files at a time. We are working to debug the zip library which will allow us to get back to ~30 files per recording session.

If you would like to run the open-speech-recording application locally (and therefore not have a limit on simultaneous downloads) please see our appendix document. Do note that you must be comfortable running python and python-pip from the command line to take this approach.

If you record using your own tools make sure to name your files **keyword_###.ogg** where keyword is whatever word you are recording and ### is any integer number. This is needed for the training scripts to process your files correctly.

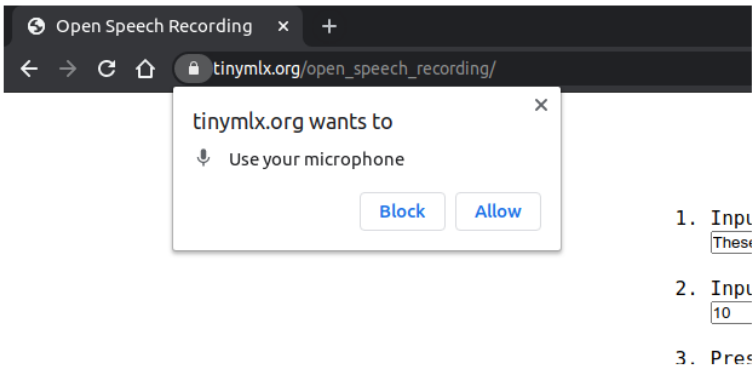
03_TinyML_C03_04-06-06_final-en.mp4 (Command Line)

We are going to use a modified version of Pete's open-speech-recording project that will keep all of your recording data local to your browser (and thus it will never leave your computer)! You can open up the application in your browser by navigating to: https://tinyml.seas.harvard.edu/open_speech_recording **NOTE: Our URL has changed from the video!**

Please make sure to open this application on a device that you can easily download and upload files from (e.g., a laptop as compared to a phone).

If you see a warning that says: Your device does not support the HTML5 API needed to record audio (this is a known problem on iOS).

1. If you are not using Chrome, we suggest you switch to Chrome. We have tested all of our applications on Chrome and cannot guarantee compatibility with other browsers.
2. Make sure you are navigating to https://. For some reason, if you do not use the secure protocol, then the HTML 5 API will not work.
3. Make sure you approve access to your microphone! The first time you load the webpage (or if you load it again in an incognito window), you should get a popup on the top left of your screen as shown below. Make sure to click the "allow" button!



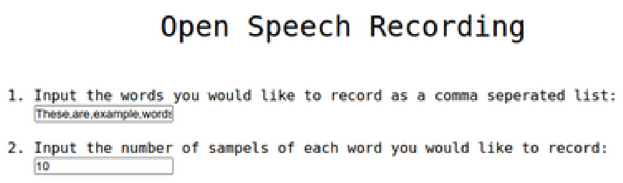
Using the Open Speech Recording App

1. In the middle of the screen, you will see the instructions for the App. Make sure you follow the steps in order. First, in step 1, type in **as a comma separated list**, the words you would like to record. And in step 2, type in **as a single integer**, the number of times you would like to record those words.

   Note that we have found that ~40 recordings each of 2-4 words often will be enough data to successfully train a moderately decent KWS application (i.e., it will work for your voice in whatever exact room you are in but may not generalize beyond that and may only work well for some of the keywords).

   Also, note that since all recordings will be saved in your browser memory and you can record multiple times by refreshing the page to reset the application, we do not suggest recording too many words in any one round of recording. **We have found that Chrome will only let you download 10 recordings per session. So do not try to record more than 10 words before refreshing the page.** Note that this is a change from the video as at the time of recording we were using a zip library that allowed ~30 words to be downloaded at a time. Unfortunately, that library, JSZip, now does not seem to be working. If you happen to be a web development guru and have ideas for a stable way to zip and download more files please do let us know!

As a final note, remember that you can record both keywords and other distractor words (e.g., if your keyword was "slow" you could also record "snow") to help. refine the model.


Open Speech Recording

1. Input the words you would like to record as a comma seperated list:
   These,are,example,words
2. Input the number of sampels of each word you would like to record:
   10

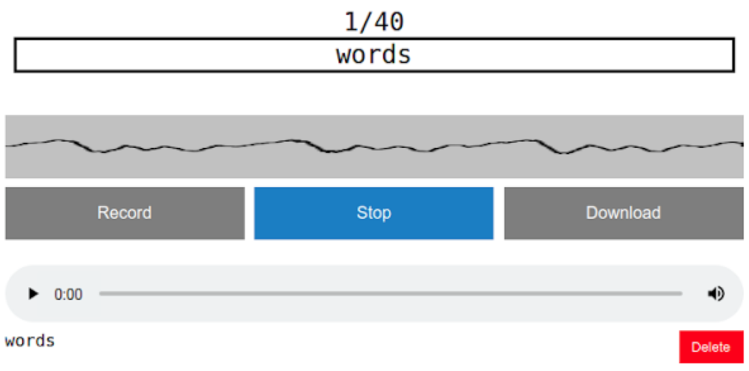
2. Then, click the blue record button. You will see a 3,2,1 timer countdown occur and then recording will start.

   You will see a counter of X/Y appear that will show you your progress in recording all of the words. You will also see the current word you are supposed to record appear in the white box. The words will come shuffled in a random order, so sometimes they will alternate and sometimes, you will get the same word multiple times in a row. You will also see the waveform of your current speech appear in the grey box below that.

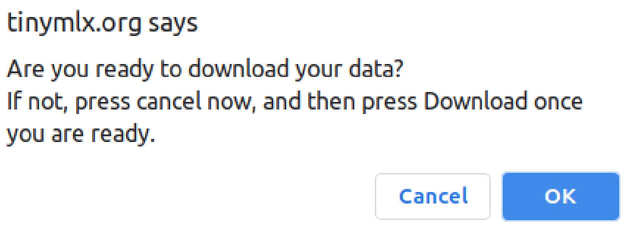
   It may take you a couple of tries to get the exact timing working. As a tip, the "Record" button will alternate between grey (as shown below) and red. Record the word shown in the white box each time the word record turns red.

   After each word is recorded, it will appear as a new row in the bottom of the screen.

   You can pause the recording session by clicking "Stop" at any time. The "Stop" button will then turn grey and the "Record" button will turn blue again. Click "Record" to restart the recording session.

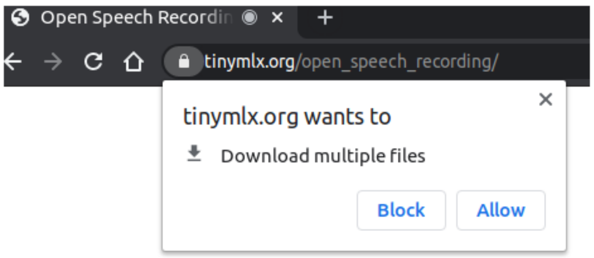


3. Once you complete recording the final word in this session, a pop-up will appear asking if you'd like to download your recordings now. If you are satisfied, click "OK" otherwise, click "Cancel" and you can inspect the recordings one by one and download them later.


tinymlx.org says

Are you ready to download your data?
If not, press cancel now, and then press Download once you are ready.

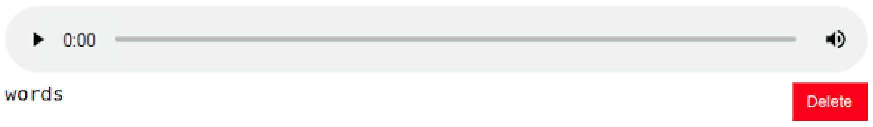
Note that if you use the application multiple times, you may get a pop-up asking for your permission to download multiple files. Click "Allow" and it should then automatically download all of your recordings again in the future without asking.



tinymlx.org wants to

Download multiple files

4. To inspect a recording, simply press the play button on the left of its row. You can delete any recordings you don't like by clicking the red "delete" button on the bottom right of its row. You can also adjust the volume of the recording as you listen to it by using the speaker icon on the right of each row.



5. Once you have removed any recordings, you do not like simply click the "Download" button in the middle of the screen that will have turned blue. Note the warning mentioned above about Chrome needing approval to download multiple items.
6. And that's it! You've successfully collected a KWS dataset. Feel free to record multiple times if you'd like to increase the size of your dataset and remember where your files are being downloaded. If you happen to be recording at a time where your download is a single zip file (instead of many .ogg audio files), you'll want to unzip all of your recordings to a folder from which you can upload them into the next Colab for training! On most operating systems, right-clicking a zip file will reveal an option that says either "extract" or "open archive manager" or something along those lines. Clicking that will either immediately unzip the files or open up a window which will have an option to unzip the files.

Train and Deploy Your Custom Dataset KWS Model

In this reading, we are going to walk through the steps for training a custom KWS model based on the dataset you just collected and then deploy that model onto your Arduino (including the necessary code changes in the Arduino IDE).

03_TinyML_C03_04-06-08_final-en.mp4 (Command Line)

Training with your Custom Dataset

The first thing you'll need to do is to upload your custom dataset into Colab and train a KWS model. Then, in that Colab, we'll need to convert that model first into a quantized .tflite file and then into a .cc file for use with the Arduino IDE.

As with the pretrained model and favorite keyword model, we will be using the resulting .cc file, so make sure to download it or leave the tab open with the printout!

https://colab.research.google.com/github/tinyMLx/colabs/blob/master/4-6-8-CustomDatasetKWSModel.ipynb

Updating the Arduino Code

1. Again, open the micro_speech.ino sketch, which you can find via the File drop-down menu. Navigate, as follows: File → Examples → Arduino_TensorFlowLite → micro_speech.
2. As before, navigate to the micro_features_model.cpp file and update the model.
    0. Copy the binary model file contents from the KWS_custom.cc file into the micro_features_model.cpp file. As always, make sure to only copy the binary data inside the { } as the variable type is different in the downloaded or printed KWS_custom.cc file.
    1. Next, scroll all the way down to the bottom of the file and replace the model length. Again, note that the .cpp file needs the variable to be of type const int while the .cc file will show unsigned int. Our suggestion again is to simply copy the numerical value.
3. Next, save your changes. Again, we suggest that you make a folder called e.g., TinyML inside of your Arduino folder. You can find your main Arduino folder either inside of your Documents folder or in your Home folder, and save it in that folder with a descriptive name like micro_speech_edx_custom_keywords. That said, you can save it wherever you like with whatever name you want!
4. The two other things we need to change are the count and list of keywords in the model settings files and the output response file. As we do that in the rest of this document, we are going to assume that you used our model with the keywords "brian," "vijay," "pete." If you choose to use different keywords, make sure that you update the following steps accordingly.
5. Navigate to the micro_features_micro_model_settings.h file and scroll down to line 40 and change the value of the kCategoryCount variable to be two more than the number of keywords you selected (one extra for silence and one extra for unknown). For example, if you had 3 keywords (as we do in our example model), we need to update the variable to be 3 + 2 = 5 as shown below:

```
constexpr int kCategoryCount = 5;
```

6. Then, navigate to the micro_features_micro_model_settings.cpp file. You'll find that it includes the .h file and otherwise has a single array. Update the values of that array to match your keywords. **Make sure to leave "silence" and "unknown" in the list as**

well, and make sure to list your key words in the order that you listed them on the training script. So, in the case of our example model, we would update the array to be:

```
const char* kCategoryLabels[kCategoryCount] = {
 "silence",
 "unknown",
 "brian",
 "vijay",
 "pete",
};
```

7. Finally, navigate to the arduino_command_responder.cpp file. Scroll down to line 54. There, you will see a series of if statements which control how the Arduino responds to each command. They are each structured to look at the value of the found_command variable. This variable will be the string of the keyword (or “silence” or “unknown”) you entered into the kCategoryLabels array in the previous step. So, by comparing the [0] value in that string, the if statements are simply looking at the first letter! In each if statement you’ll also see a digitalWrite to either the LEDG, LEDR, or LEDB variable, which is used to turn on the appropriate color LED. So, if we wanted to adapt this to our three keywords and, for example, only turn on the single colors for our keywords and turn all of the colors on for unknown, which will come out look white, we could update those three if statements to the following:

```
// Red for Vijay
if(found_command[0] == 'v') {
  last_command_time = current_time;
  digitalWrite(LEDR, LOW);
}
// Green for Pete
if(found_command[0] == 'p') {
  last_command_time = current_time;
  digitalWrite(LEDG, LOW);
}
// Blue for Brian -- note here that you do not need to index
//              into the first letter only, just a unique
//              letter combination in the keyword! That
//              said make sure you do not index beyond the
//              end of ANY keyword or you will get an error!
if(found_command[0] == 'b' && found_command[1] == 'r') {
  last_command_time = current_time;
  digitalWrite(LEDB, LOW);
}
// All three for unknown (white)
if(found_command[0] == 'u') {
   last_command_time = current_time;
   digitalWrite(LEDR, LOW);
   digitalWrite(LEDG, LOW);
   digitalWrite(LEDB, LOW); }
```

Remember, if you used different keywords make sure to update the if statements accordingly. And that's it! You're Arduino code should be all set to handle your new model!

Deploying the New Model

1. Use a USB cable to connect the Arduino Nano 33 BLE Sense to your machine. You should see the green LED power indicator come on when the board first receives power.
2. As always, use the Tools drop-down menu to select appropriate Port and Board.
    - Select the Arduino Nano 33 BLE as the board by going to Tools → Board: <Current Board Name> → Arduino Mbed OS Boards (nRF52840) → Arduino Nano 33 BLE.
    - Then, select the USB Port associated with your board. This will appear differently on Windows, macOS, Linux but will likely indicate 'Arduino Nano 33 BLE" in parenthesis. You can select this by going to Tools → Port: <Current Port (Board on Port)> → <TBD Based on OS> (Arduino Nano 33 BLE). Where <TBD Based on OS> is most likely to come from the list below where <#> indicates some integer number.
        - Windows → COM<#>
        - macOS → /dev/cu.usbmodem<#>
        - Linux → ttyUSB<#> or ttyACM<#>
3. Use the rightward arrow next to the 'upload' / flash the code. You'll know the upload is complete when you see red text in the console at the bottom of the IDE that shows 100% upload of the code and a statement that says something like "Done in <#.#> seconds."
4. If you receive an error, you will see an orange error bar appear and a red error message in the console (as shown below). Don't worry -- there are many common reasons this may have occurred. To help you debug, please check out our FAQ appendix with answers to the most common errors!
5. You should now have a working Keyword Spotting model trained on your own custom dataset deployed to your Arduino! To test it out, open the Serial Monitor. You should start to see the result of the model begin to display in the Serial Monitor and the LED's light up according to how you instructed them in the previous section!

1.7: Visual Wake Words

Recap: What are Visual Wake Words?

In this module, you will learn to build and implement an algorithm for detecting visual wake words on your local microcontroller.

In the previous section focused on keyword spotting, we empowered our microcontroller with a lightweight auditory modality, allowing the system to monitor the microphone input in real-time to be able to recognize and respond to several keywords of interest to users. Visual wake words (VWW) are the natural extension of this technique to visual data, providing a lightweight visual modality to our microcontroller system.

Image information is inherently more complex than auditory information to analyze using machine learning due to the higher dimensionality and complex spatial correlations. Consequently, image data requires greater computational and memory resources, making it challenging to deploy on resource-constrained microcontrollers. An additional complication, similar to keyword spotting, is the difficulty in procuring sufficient data to train our algorithm. In some scenarios, we may have to generate the dataset ourselves.

In the previous classes, we have looked at the VWW task in some detail, from the implementation in Colab to discussions of image privacy and copyright. We analyzed a specific set of architectures, known as the **MobileNet** architectures, which leverage **depthwise separable convolutions** to minimize required memory and computational resources. We also looked at how **transfer learning** can be used to build upon pre-trained models that were used for similar tasks in the same domain to reduce our need to procure large datasets and train large deep learning models.

The main VWW task we focused on was **person detection**, which focuses on determining the presence or absence of an individual in an image. In this task, we used the VWW dataset to train our person detection model and then performed transfer learning using a relatively small amount of images to adapt our model to perform mask detection. However, we have yet to demonstrate how to port and deploy such a model to our end-device, which will be the main focus of this section.

By the end of this section, you should have a fully functioning VWW detection system. This shares some similarities to keyword spotting, but the deployment procedure will require some pre-and post-processing (e.g., image downscaling, encoding/decoding, and interfacing with the onboard camera) unique to the VWW detection task.

Not only this, but we will also demonstrate the deployment of keyword spotting and VWW simultaneously using model "**multitenancy**." More often than not, the typical focus is on running individual models in isolation, independent of other contextual activities. But increasingly there is a growing need to combine information from multiple sensors together to perform an intelligent action. To that end, we will learn how to perform audio-visual recognition, using both the microphone and camera. Our goal is to simultaneously operate two models at the same time in order to mimic sensor fusion (i.e., the ability to tie together inputs from different sensors in close temporal proximity).

We will learn how to do this and more in TensorFlow Lite Micro.

Person Detection Application Architecture

03_TinyML_C03_04-07-02_final-en.mp4 (Command Line)

Person Detection Screencast

03_TinyML_C03_04-07-03_final-en.mp4 (Command Line)

Person Detection with Keyword Spotting: MultiModal

03_TinyML_C03_04-07-05_final-en.mp4 (Command Line)

Person Detection with Keyword Spotting: Multi-Tenancy

03_TinyML_C03_04-07-06_final-en.mp4 (Command Line)

Multi Tenancy in TensorFlow Lite Micro

03_TinyML_C03_04-07-08_final-en.mp4 (Command Line)

Deploying the Pretrained Person Detection Model

03_TinyML_C03_04-07-07_final-en.mp4 (Command Line)

1. Use a USB cable to connect the Arduino Nano 33 BLE Sense to your machine. You should see the green LED power indicator come on when the board first receives power.

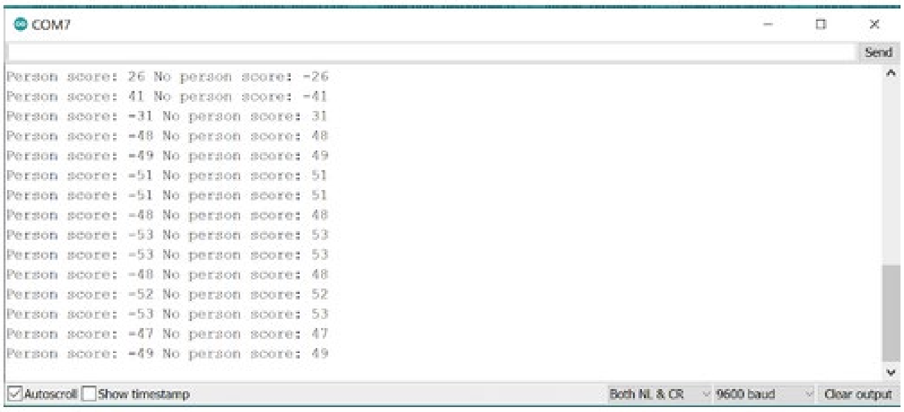
2. Open the person_detection.ino sketch, which you can find via the File drop-down menu. Navigate, as follows: File → Examples → Harvard_TinyMLx→ person detection. Our library version of the example is very similar to the one you'll find in the TensorFlow Lite for Microcontrolers library, but includes a couple of important tweaks to work better with the TinyML Shield and both the OV7675 and OV7670 so make sure you open the example out of our library.
3. Use the Tools drop-down menu to select the appropriate Port and Board. This is important as it is telling the IDE which board files to use and on which serial connection it should send the code. In some cases, this may happen automatically, but if not, you'll want to select:
    - Select the Arduino Nano 33 BLE as the board by going to Tools → Board: <Current Board Name> → Arduino Mbed OS Boards (nRF52840) → Arduino Nano 33 BLE. Note that on different operating systems the exact name of the board may vary but/and it should include the word Nano at a minimum. If you do not see that as an option, then please go back to Setting up the Software and make sure you have installed the necessary board files.
    - Then, select the USB Port associated with your board. This will appear differently on Windows, macOS, Linux, but will likely indicate 'Arduino Nano 33 BLE" in parenthesis. You can select this by going to Tools → Port: <Current Port (Board on Port)> → <TBD Based on OS> (Arduino Nano 33 BLE). Where <TBD Based on OS> is most likely to come from the list below where <#> indicates some integer number:
        - Windows → COM<#>
        - macOS → /dev/cu.usbmodem<#>
        - Linux → ttyUSB<#> or ttyACM<#>
4. Use the rightward arrow to upload / flash the code. Do not be alarmed if you see a series of orange warnings appear in the console. This is expected as we are working with bleeding edge code. You'll know the upload is complete when you red text in the console at the bottom of the IDE that shows 100% upload of the code and a statement that says something like "Done in <#.#> seconds."

   If you have the OV7670 camera, make sure to first make the changes indicated at the end of this document!

   If you receive an error, you will see an orange error bar appear and a red error message in the console. Don't worry -- there are many common reasons this may have occurred.

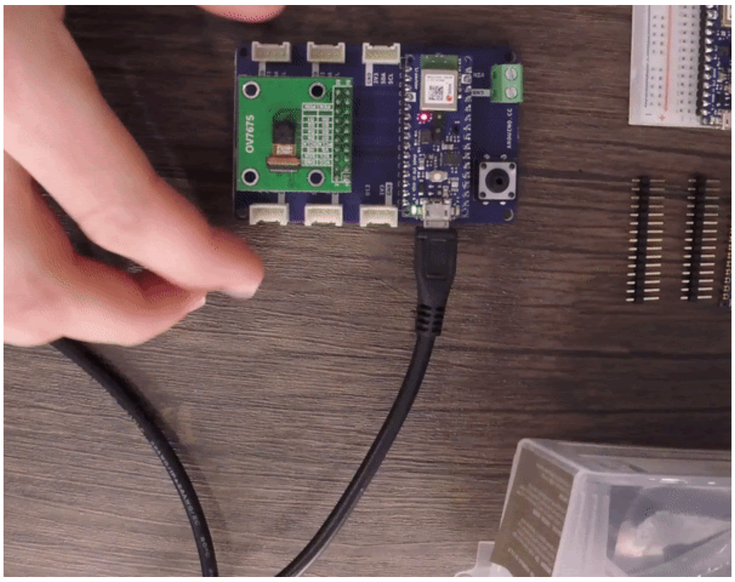
   To help you debug other issues, please check out our FAQ appendix with answers to the most common errors!

5. Open up the serial monitor. You will start to see it print a series of messages about the person score and the no person score. This is indicating the model's confidence in whether there is either a person or no person in the image!



```
Person score: 26 No person score: -26
Person score: 41 No person score: -41
Person score: -31 No person score: 31
Person score: -48 No person score: 48
Person score: -49 No person score: 49
Person score: -51 No person score: 51
Person score: -51 No person score: 51
Person score: -48 No person score: 48
Person score: -53 No person score: 53
Person score: -53 No person score: 53
Person score: -48 No person score: 48
Person score: -52 No person score: 52
Person score: -53 No person score: 53
Person score: -47 No person score: 47
Person score: -49 No person score: 49
```

At this point, you'll want to look at the board itself. The LED should be flashing blue when an image is being captured and then either red or green. Green indicates that the model detects that there is a person in the image and red indicates that it does not detect a person. Try holding the Arduino at arm's length and point it toward your face. The LED should flash green. As you may see, the field of view is very narrow, so if it's not working, try propping up the Arduino and standing further back.



Understanding the Code in the Person Detection Example

Now that you have gotten the person detection application deployed to your microcontroller, let's explore the code a little bit. The main person_detection.ino file consists of the standard Arduino setup() and loop() functions. These are both quite similar to the hello_world and micro_speech example. To really start to understand the code, let's take a look at a couple of the other files.

Open the arduino_image_provider.cpp file. Here, we will find the GetImage() function, which captures an image and loads it into the model's input.

First, let's take a look at the camera initialization. Here, we set the resolution of the image we receive from the camera to QCIF (176x144). In order to save space and complexity, the input to our model is a grayscale image, therefore, we set the camera to transmit a grayscale image. We also set our camera model to the OV7675.

Note: As shown in the changes to the OV7670 section below, this is where you'll need to change the OV7675 to OV7670.

```
// Initialize camera if necessary
if (!g_is_camera_initialized) {
  if (!Camera.begin(QCIF, GRAYSCALE, 5, OV7675)) {
    TF_LITE_REPORT_ERROR(error_reporter, "Failed to initialize camera!");
    return kTfLiteError;
  }
  g_is_camera_initialized = true;
}
```

Next, we can capture an image with Camera.readFrame(). Since the input of our model is a 96x96 image, we have to crop the 176x144 image we receive from the camera before passing it to the model. To do this, we copy over the desired pixel values from the center of our original image into the input vector of our model.

```
// Crop 96x96 image. This lowers FOV, ideally we would downsample
// but this is simpler and more efficient on device.
for (int y = min_y; y < min_y + 96; y++) {
   for (int x = min_x; x < min_x + 96; x++) {
       // convert TF input image to signed 8-bit
       image_data[index++] = static_cast<int8_t>(data[(y * 176) + x] - 128);
   }
}
```

Now that we are familiar with capturing and loading the image, let's take a look at how the person detection application handles the output of the model. To do that, open the arduino_detection_responder.cpp file.

The primary function of the detection responder is to control the LED output. In a real application, this function would drive some other core function that is triggered by the presence of a person.

The blue LED will flash every inference, demonstrating how often an inference is run. Then we will compare the person no_person scores that are the outputs of the model. In this case, we flash the green LED when the person_score is higher than the no_person_score (both of which are reported over serial). In some applications, we might be more concerned with the false positives or false negatives, in which case, we can adjust the criteria for detecting a person, potentially by using a threshold as we have discussed in Course 2.

```
//Flash the blue LED after every inference.
digitalWrite(LEDB, LOW);
delay(100);
digitalWrite(LEDB, HIGH);
// Switch on the green LED when a person is detected,
// the red when no person is detected
```

```
if (person_score > no_person_score) {
  digitalWrite(LEDG, LOW);
  digitalWrite(LEDR, HIGH);
}
else {
  digitalWrite(LEDG, HIGH);
  digitalWrite(LEDR, LOW);
}
```

TF_LITE_REPORT_ERROR(error_reporter, "Person score: %d No person score: %d", person_score, no_person_score);

And at a high level, that is how the person_detection example works! We simply take and crop the input images, pass them to the neural network, and then report the higher scoring result!

Changes for the OV7670

To use the OV7670 instead of the OV7675, simply make the change you made in the camera test! Change the fourth argument of the call to Camera.begin() as highlighted below, from OV7675 to OV7670. That's it! The library will handle the rest! You can find the camera initialization in:

```
// Initialize the OV7675 camera
void if(!Camera.begin(QCIF, GRAYSCALE, 5, OV7675)) {
  Serial.println("Failed to initialize camera");
  While (1);
}
```

Deploying the Multi Tenant Example

03_TinyML_C03_04-07-11_final-en.mp4 (Command Line)

1. Use a USB cable to connect the Arduino Nano 33 BLE Sense to your machine. You should see the green LED power indicator come on when the board first receives power.
2. Open the multi_tenant.ino sketch, which you can find via the File drop-down menu. Navigate, as follows: File → Examples → Harvard_TinyMLx→multi_tenant.
3. Use the Tools drop-down menu to select the appropriate Port and Board. This is important as it is telling the IDE which board files to use and on which serial connection it should send the code. In some cases, this may happen automatically, but if not, you'll want to select:
   - Select the Arduino Nano 33 BLE as the board by going to Tools → Board: <Current Board Name> → Arduino Mbed OS Boards (nRF52840) → Arduino Nano 33 BLE. Note that on different operating systems the exact name of the board may vary but/and it should include the word Nano at a minimum. If you do not see that as an option, then please go back to Setting up the Software and make sure you have installed the necessary board files.
   - Then, select the USB Port associated with your board. This will appear differently on Windows, macOS, Linux, but will likely indicate 'Arduino Nano 33 BLE" in parenthesis. You can select this by going to Tools → Port: <Current Port (Board on Port)> → <TBD Based on OS> (Arduino Nano 33 BLE). Where <TBD Based on OS> is most likely to come from the list below where <#> indicates some integer number.
     - Windows → COM<#>
     - macOS → /dev/cu.usbmodem<#>
     - Linux → ttyUSB<#> or ttyACM<#>
4. Use the rightward arrow to upload / flash the code. Do not be alarmed if you see a series of orange warnings appear in the console. This is expected as we are working with bleeding edge code. You'll know the upload is complete when you see red text in the console at the bottom of the IDE that shows 100% upload of the code and a statement that says something like "Done in <#.#> seconds."

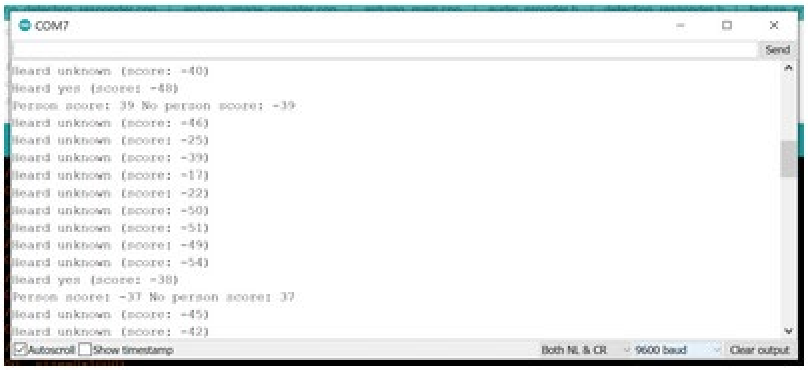
   If you have the OV7670 camera, make sure to first make the changes indicated at the end of this document!

   If you receive an error, you will see an orange error bar appear and a red error message in the console. Don't worry -- there are many common reasons this may have occurred.

   To help you debug other issues, please check out our FAQ appendix with answers to the most common errors!

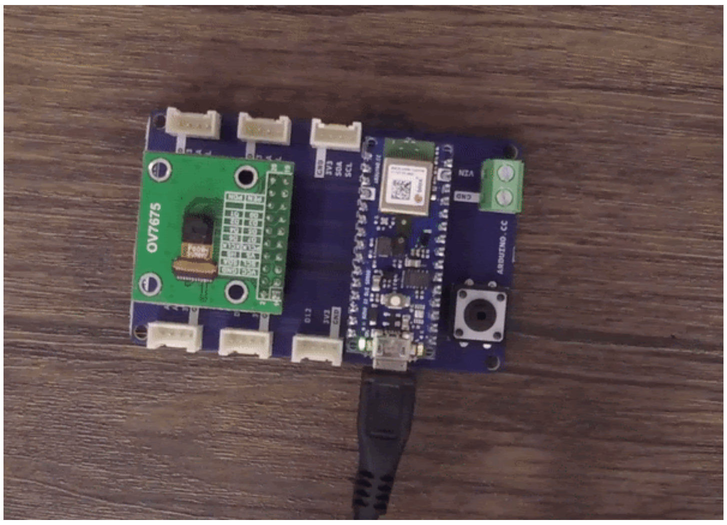
5. Open up the serial monitor. You will start to see the output from the "Yes, No" keyword spotting model. We can then trigger it with a "Yes." Once triggered, the person detection model will run and we will then see the output from that printed to the serial

monitor as well.



```
Heard unknown (score: -40)
Heard yes (score: -48)
Person score: 39 No person score: -39
Heard unknown (score: -46)
Heard unknown (score: -25)
Heard unknown (score: -39)
Heard unknown (score: -17)
Heard unknown (score: -22)
Heard unknown (score: -50)
Heard unknown (score: -51)
Heard unknown (score: -49)
Heard unknown (score: -54)
Heard yes (score: -38)
Person score: -37 No person score: 37
Heard unknown (score: -45)
Heard unknown (score: -42)
```

At this point, you'll want to look at the board itself. The yellow LED in the top left should be blinking rapidly. Try saying 'yes' and see if the board responds. If the micro_speech model detects 'yes,' it will turn the middle LED blue, and then, it will try to detect if a person is in the camera frame. If the person_detection model detects a person, it will flash the LED green, otherwise, it will flash the LED red. The field of view is quite narrow, so try holding the device at arm's length and then say 'yes' with an emphasis on the 'sss' sound.



Understanding the Code in the Multi-Tenant Example

Now that you have gotten the multi_tenent application deployed to your microcontroller, let's explore the code a little bit. The main multi_tenant.ino file is similar to the previous person detection example. The difference here is that we use the shared allocator from TFLite Micro to allocate space for two models using the same tensor arena.

Unlike previous examples where we directly passed the tensor_arena and kTensorArenaSize to the interpreter, we need to directly create an allocator and pass it to our two separate interpreters.

In the setup() function, we create our allocator that will be shared between both of our interpreters as follows:

```
tflite::MicroAllocator* allocator =
  tflite::MicroAllocator::Create(tensor_arena, kTensorArenaSize, error_reporter);
```

Then we create the interpreter for the person detection model, passing the allocator to it as a parameter, and then calling AllocateTensors().

```
static tflite::MicroInterpreter vww_static_interpreter(
   vww_model, micro_op_resolver, allocator, error_reporter);
vww_interpreter = &vww_static_interpreter;
// allocate the VWW model from tensor_arena
TfLiteStatus allocate_status = vww_interpreter->AllocateTensors();
```

Now that both interpreters are created using the same allocator, we can use them as we would normally.

In the loop() function, we first run the micro_speech model as we did in the KWS example, except this time if the model detects ‘yes’ we capture an image and run the person_detection model. The LED control is performed in arduino_detection_responder.cpp and follows the patterns used in both the micro_speech and person_detection examples.

```
bool heardYes = RespondToKWS(error_reporter, found_command,
                             is_new_command, score);
if(heardYes){
  //Our keyword spotting model heard 'yes' so we detect if a person is visible


  // Get image from provider.
  if (kTfLiteOk != GetImage(error_reporter, kNumCols, kNumRows, kNumChannels,
                            vww_input->data.int8)) {
    TF_LITE_REPORT_ERROR(error_reporter, "Image capture failed.");
  }


  // Run the model on this input and make sure it succeeds
  if (kTfLiteOk != vww_interpreter->Invoke()) {
    TF_LITE_REPORT_ERROR(error_reporter, "Invoke failed.");
  }
  // Process the inference results.
  int8_t person_score = vww_output->data.uint8[kPersonIndex];
  int8_t no_person_score = vww_output->data.uint8[kNotAPersonIndex];
  RespondToDetection(error_reporter, person_score, no_person_score);
}
```

Changes for the OV7670

To use the OV7670 instead of the OV7675, simply make the change you made in the camera test! Change the fourth argument of the call to Camera.begin() as highlighted below, from OV7675 to OV7670. That’s it! The library will handle the rest! You can find the camera initialization in:

```
// Initialize the OV7675 camera
void if(!Camera.begin(QCIF, GRAYSCALE, 5, OV7675)) {
  Serial.println(“Failed to initialize camera”);
  While (1);
}
```

1.8: Gesturing Magic Wand

Recap: Time Series for Anomaly Detection

In this reading, we will do a quick recap of the time series from Course 2 as we will be leveraging that knowledge for a slightly different application than what we discussed previously (i.e., anomaly detection): gesture recognition with a custom-built magic wand.

Supervised vs. Unsupervised Learning

So far, we have tackled many interesting machine learning tasks that cover a variety of sensory **modalities** (e.g., vision, audition). All of these tasks required a labeled dataset - a known output value for a given set of inputs - and were thus comfortably situated within the **supervised learning paradigm**, which we first discussed in Course 1.

However, frequently, we encounter datasets in which our output variable is unknown. Whether because we do not have access to the output variable, or we are merely interested in the internal structure of our dataset, we cannot resort to our standard machine learning framework, which relies on training and test sets. Instead, we must adopt an alternative, equally useful, a paradigm known as **unsupervised learning** to approach these tasks, which we discussed in Course 2.

There are still similarities between the supervised and unsupervised learning paradigms. Firstly, both assume the existence of a dataset that is used to train our machine learning algorithm. The performance of both generally improves as more data becomes available. Nevertheless, unsupervised learning is distinct in that we do not require the existence of a predictor variable. Unsupervised learning looks at the internal structure of a dataset and attempts to either group this data systematically, known as **clustering**, or look for abnormalities within the dataset, known as **outlier analysis**, or **anomaly detection**.

Anomaly Detection

In course 2, we were mostly interested in anomaly detection, predominantly due to its potential for predictive maintenance (i.e., noticing abnormalities in machines that can be resolved before catastrophic events). Anomaly detection can be performed **spatially** or **temporally**, either looking at associations between different physical locations, or the same location at different points in time.

Most sensory devices are fixed in discrete locations, such as those monitoring the temperature and vibrational characteristics of a piece of machinery. Inherently, these devices output a temporal signal that can be monitored and used to look for anomalies over time. These anomalies manifest in myriad ways: a sudden temperature change, a sudden increase in vibration or sound, or a sudden drop in pressure. Given sufficient explanatory variables, anomaly detection algorithms allow us to detect outliers and perform corrective actions in a timely manner to mitigate the possibility of catastrophic events, such as an error, a machine breakdown, or even an explosion.

Many algorithms exist for the detection of anomalies in time series data. One algorithm that was discussed in the previous course was **k-means**, which looks at the distance of a data point in feature space from a set of discrete clusters that represent the majority of data points. We saw that one drawback of k-means was its poor performance on high-dimensional data due to the **curse of dimensionality**. However, we were able to improve performance by reducing the dimensionality of the data with **t-SNE**. Anomaly detection can also be achieved using neural network structures known as **autoencoders**, which attempt to collapse knowledge of the dataset in a small set of neurons located at the mid-layer of the autoencoder (this formulation is very similar to principal component analysis). Autoencoders tend to have superior performance to traditional methods due to their ability to capture complex non-linear associations.

Magic Wand

In the remainder of this section, we will leverage this prior knowledge for a slightly different purpose: building in gesture recognition using a custom-built magic wand.

TinyML Sensor Ecosystem

03_TinyML_C03_04-08-02_final-en.mp4 (Command Line)

Anatomy of an IMU

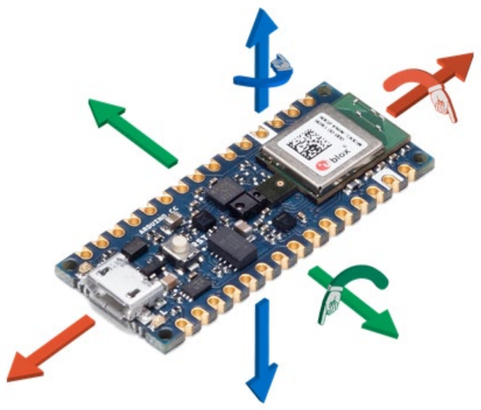
Anatomy of an IMU
Accelerometer
Magnetometer
Gyroscope

An inertial measurement unit (IMU) is a system composed of sensors that relay information about a device’s movement, such as accelerometers, gyroscopes, and magnetometers. In concert, each of the sensors provides a detailed account of a device’s instantaneous motion, orientation, and acceleration, which are often used for navigational and stabilizing purposes in vehicles and aircraft as part of electronic feedback control systems. We will be using it for our Magic Wand application. The Arduino Nano 33 BLE sense has a 9-axis IMU.

More recently, applications of the IMU to wearable devices have been stimulated by the quantified self-movement. The data generated from IMU's can be used for machine learning purposes in classifying different types of actions associated with a device. For example, by attaching an IMU to a specific body part, such as an arm, it may be possible to discern different gestures purely from time-series information, such as waving, shaking hands, or typing on a keyboard. This makes the IMU particularly useful for embedded machine learning applications that are gesture-activated, such as lighting up your phone screen when picking it up, when checking the time on a smartwatch, or even fall detection.

Structure of an IMU

An IMU consists of at least two of the following sensors: an accelerometer, a gyroscope, and a magnetometer. In principle, these devices can be designed to provide single- or multi-axis measurements, but most offer multi-axis measurements to provide an accurate set of information associated with the position and orientation of a device in three-dimensional space. The importance of the IMU for sensing applications is highlighted by our Arduino Nano BLE Sense, which comes packaged with a built-in 9-axis IMU.



The three sensors that constitute an IMU afford for the following capabilities:

Accelerometer. Measures changes in velocity (acceleration) and position (velocity), as well as absolute orientation. The accelerometer is the device in tablets and smartphones which ensures the image on-screen remains upright regardless of orientation. By itself, the accelerometer provides information about the linear and rotational X-, Y-, and Z- directions. The accelerometer allows the 3-axis of motion to be captured. An onboard accelerometer is modeled as a micromechanical damped mass-spring system, wherein the compression or extension of the mass-spring system can be mathematically related to an object's acceleration.

Magnetometer. Establishes cardinal direction (directional heading). A simple example of a magnetometer is a compass, which is used to measure the direction of the Earth's magnetic field. Smaller versions called microelectromechanical magnetic field sensors can be incorporated into integrated circuits, allowing them to be combined with other sensors as part of an IMU. The magnetometer measures orientation to magnetic north on the X, Y, and Z axes. Together with the accelerometer, the 6 degrees-of-freedom of a system can be accounted for, which fully describes the kinematics of a system. Oftentimes, magnetometers work via the Hall effect, which involves the creation of a potential difference across a conductor as a result of a perpendicular applied magnetic field.

Gyroscope. Measures changes in orientation (rotation) and rotational velocity. Microelectromechanical gyroscopes, often called gyro meters, are present in many consumer electronics such as gaming controllers. A gyroscope provides information about the rotational X- (roll), Y- (pitch), and Z- (yaw) directions. The 6 degree-of-freedom system created by combining a magnetometer and accelerometer suffers from several shortcomings, such as the sensitivity of magnetometers to time-varying magnetic disturbances, and the corruption of accelerometer readings due to the presence of linear acceleration distorting the Earth gravity vector.

To alleviate these shortcomings, a 3-axis gyroscope can be added, creating a 9 degree-of-freedom, or "gyro-stabilized" system. The gyroscope provides the system with an independent measurement of instantaneous rotation speed, complementing the original 6 degree-of-freedom system. The gyroscope works by the conservation of angular momentum; a spinning wheel or disc rotates at high speed, while the spin axis is freely allowed to rotate and assume any orientation. The inertia of the spinning wheel allows it to remain unperturbed in the same direction during tilting or rotation, allowing rotational information to be discerned.

Next Steps
In the following sections, we will focus on obtaining time-series information from the onboard IMU, and subsequently, use this information to train a machine learning model capable of detecting specific actions of a magic wand.

Magic Wand Application

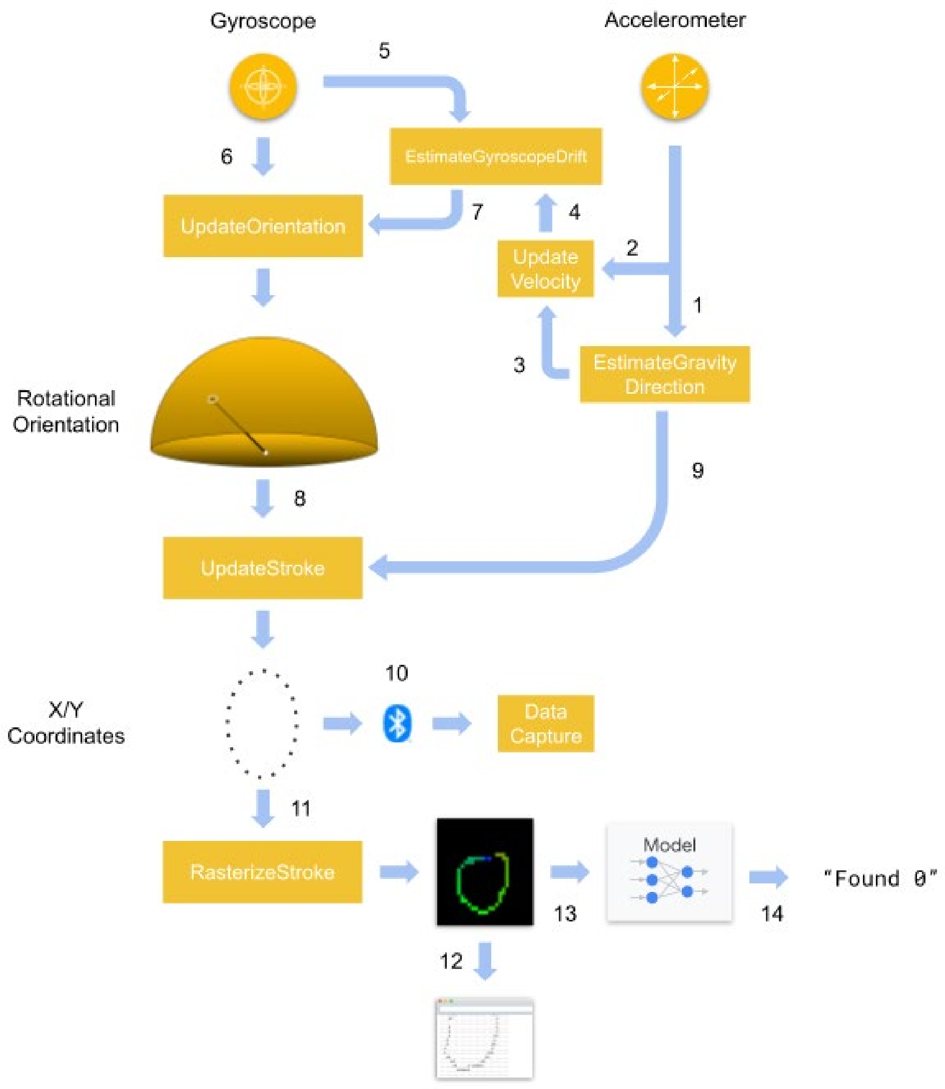
03_TinyML_C03_04-08-04_final-en.mp4 (Command Line)

Magic Wand Application Architecture

03_TinyML_C03_04-08-05_final-en.mp4 (Command Line)

Understanding the Magic Wand Application

This reading is about understanding the components of the magic wand application and how they generalize to other IMU-based projects. By gaining familiarity with the data flow, from the sensor to classification, you can apply similar concepts to other applications.



The Magic Wand application accomplishes a fairly complicated task (gesture recognition) by carefully crafting a 2D image from 3D IMU data. The dataflow displayed in the diagram above consists of 14 steps:

1. The accelerometer data is read and passed to the EstimateGravityDirection() where it is used to **determine the orientation** of the Arduino with respect to the ground.
2. The Accelerometer data is passed to UpdateVelocity() where it is used to **calculate the velocity** of the Arduino.
3. The direction of gravity is passed to UpdateVelocity() and is used to **cancel out the acceleration due to gravity** from the accelerometer data.

4. The velocity is then passed to EstimateGyroscopeDrift() where it is used to **determine if the Arduino is stationary or moving**.
5. The gyroscope data is passed to EstimateGyroscopeDrift() where it is used to **calculate the sensor drift of the gyroscope** if the Arduino is not moving (velocity is 0).
6. The gyroscope data is passed to UpdateOrientation() where it is integrated to **determine the angular orientation** of the Arduino.
7. The gyroscope drift is also passed to UpdateOrientation() and subtracted from the gyroscope reading to **cancel out the drift**.
8. The 3D angular orientation is passed to UpdateStroke() transformed into 2d positional coordinates. UpdateStroke() also handles whether the **current gesture has ended or if a new gesture has been started** by analyzing both the length of the gesture and testing whether the orientation data is still changing.
9. The direction of gravity is also passed to UpdateStroke() to **determine the roll orientation** of the Arduino.
10. The 2d positional coordinates are sent over BlueTooth to the **data collection** application in the browser.
11. The 2d positional coordinates are passed to RasterizeStroke() which takes the 2D coordinates and **draws lines between them on a 2D image**. The color of the lines shifts from red to green to blue to indicate the direction of motion during the gesture.
12. The 2D image of the gesture is converted to ASCII art and **printed on the serial monitor**.
13. The 2D image of the gesture is **passed to the model**.
14. The model **predicts the label of the gesture** and the label is printed to the serial monitor.

For a deeper understanding of the Magic wand application, try following the flow diagram while reading the code. The comments should provide the context needed to understand the function of each section. Try to piece the application together, one piece at a time, as there is a lot going on in this application. That said, each piece by itself is fairly self-contained.

Deploying the Magic Wand Project

03_TinyML_C03_04-08-07_final-en.mp4 (Command Line)

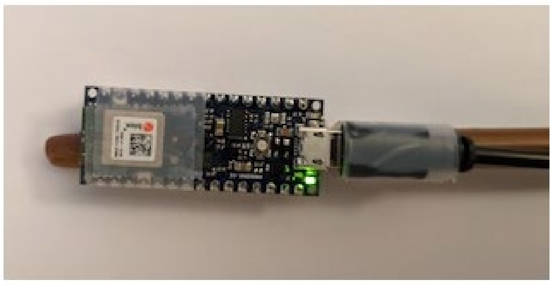
Building the Wand

Note that you do not have to build the full magic wand and can simply hold the Arduino (with or without the shield) in your hand. However, you will likely find that when you deploy the example with Pete’s pre-trained digit model, since he trained it using a ‘wand,’ it will be more accurate if you use a ‘wand’ too.

If you’d like to build a ‘wand,’ the 'wand' itself can be as simple as a stick. It doesn't need to do anything other than keep the board at its end as you hold the other end and wave it about.

For fun, the course staff members have found cheap wands from online retailers and used them, but a simple piece of wood, a ruler, or even a thick piece of cardboard should work just as well.

You should place the board at the end of the wand, with the USB socket facing downwards, towards where you hold it, so that the cable can run down the handle. The sketch is designed to compensate for any rotation of the board around the wand's shaft, so as long as it's parallel to the wand's length, the board's twist won't matter. Use some sticky tape or some other easy-to-remove method to attach the board to the wand, and hold the cable in place along the shaft. The end result should look something like the image below.



If an ASCII-art diagram is more helpful, here's what you should aim for:

```
 -----
|     | <- Arduino board
|     |
| ()  |  <- Reset button
|     |
 -TT-    <- USB port
  ||
  ||<- Wand
 ....
  ||
  ||
  ()
```

Deploying the Magic Wand Project

1. Use a USB cable to connect the Arduino Nano 33 BLE Sense to your machine. You should see the green LED power indicator come on when the board first receives power.
2. Open the magic_wand.ino sketch, which you can find via the File drop-down menu. Navigate, as follows: File → Examples → Harvard_TinyMLx → magic_wand.
3. As always, use the Tools drop-down menu to select appropriate Port and Board.

   - Select the Arduino Nano 33 BLE as the board by going to Tools → Board: <Current Board Name> → Arduino Mbed OS Boards (nRF52840) → Arduino Nano 33 BLE. Note that on different operating systems the exact name of the board may vary but/and it should include the word Nano at a minimum. If you do not see that as an option, then please go back to Setting up the Software and make sure you have installed the necessary board files.

- Then select the USB Port associated with your board. This will appear differently on Windows, macOS, Linux but will likely indicate 'Arduino Nano 33 BLE" in parenthesis. You can select this by going to Tools → Port: <Current Port (Board on Port)> → <TBD Based on OS> (Arduino Nano 33 BLE). Where <TBD Based on OS> is most likely to come from the list below where <#> indicates some integer number:
    - Windows → COM<#>
    - macOS → /dev/cu.usbmodem<#>
    - Linux → ttyUSB<#> or ttyACM<#>

4. Use the rightward arrow to upload / flash the code. Do not be alarmed if you see a series of orange warnings appear in the console. This is expected as we are working with bleeding edge code. You'll know the upload is complete when you see red text in the console at the bottom of the IDE that shows 100% upload of the code and a statement that says something like "Done in <#.#> seconds."

   If you receive an error, you will see an orange error bar appear and a red error message in the console. Don't worry -- there are many common reasons this may have occurred.

   To help you debug other issues, please check out our [FAQ appendix]() with answers to the most common errors!

5. Now open the serial monitor. The default magic wand will recognize the digits 0-9 based on data recorded by Pete, so your accuracy may vary depending on how close your "gesture handwriting" is to his! **The serial monitor will output first ASCII art showing the gesture you just performed and below it will be the best match label as well as a confidence score between -128 and 127**. The confidence score indicates how strongly the model believes you performed the gesture. Do note that every time you move the board and then stop, a new gesture will be processed, so don't be surprised to get some odd results as you move the board to prepare for a gesture.

```
................................
................................
................................
................................
................................
................................
................................
................................
................................
................................
...........###########..........
..........#.....##...##.........
..........#....##......#........
.......###......####...##.......
.......#................##......
......#..................#......
......#..................#......
....##....................#.....
..........................#.....
....#.....................#.....
....#.....................#.....
....#....................##.....
...#.....................#......
...#....................##......
..#.....................#.......
..#...................###.......
...##................##.........
.....##............##...........
......##.....#.#####............
........######..................
................................
................................
Found 0 (126)
```

Collecting Data for Your Custom Magic Wand Project

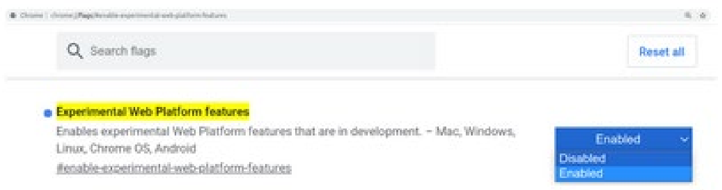
In this document, we are going to collect custom gestures, which we can then later use to train a custom magic wand project.

NOTE: Our URL has changed from the video! See below for details!

.mp4

03_TinyML_C03_04-08-09_final-en.mp4 (Command Line)

1. Make sure the default Magic Wand project is deployed as described in the previous reading!
2. Open up your browser and navigate to: https://tinyml.seas.harvard.edu/magic_wand. **NOTE: Our URL has changed from the video!**
   - If you see a warning that says: Error: This browser doesn't support Web Bluetooth. Try using Chrome. If you are not using Chrome, we suggest you switch to Chrome. If you are already using Chrome, then navigate to the following and make sure to "enable" the "experimental web platform features." We have found this step to be necessary on older versions of Chrome and users running the Linux operating system. chrome://flags/#enable-experimental-web-platform-features.



Search flags

Reset all

Experimental Web Platform features

Enables experimental Web Platform features that are in development. – Mac, Windows, Linux, Chrome OS, Android

#enable-experimental-web-platform-features

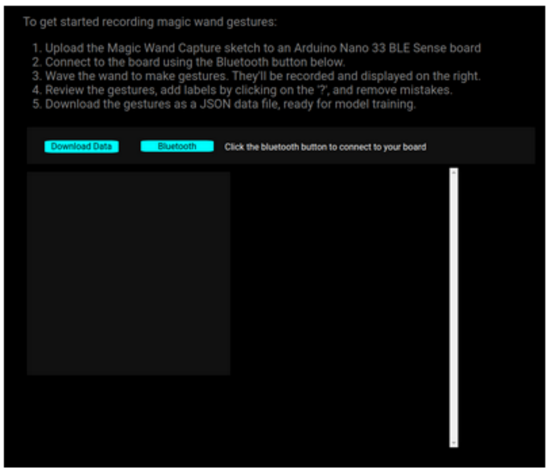
Enabled

Disabled

Enabled

     If this problem persists, make sure you are navigating to https://. For some reason, if you do not use the secure protocol, then the web Bluetooth will not work.ž

   - You should then arrive at a webpage that looks something like the following:



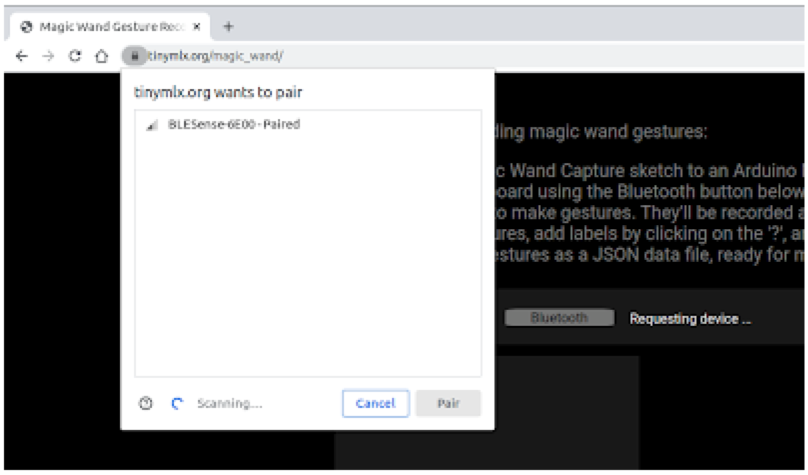
To get started recording magic wand gestures:

1. Upload the Magic Wand Capture sketch to an Arduino Nano 33 BLE Sense board
2. Connect to the board using the Bluetooth button below.
3. Wave the wand to make gestures. They'll be recorded and displayed on the right.
4. Review the gestures, add labels by clicking on the '?', and remove mistakes.
5. Download the gestures as a JSON data file, ready for model training.

Click the bluetooth button to connect to your board

3. You'll then need to connect your device over Bluetooth. To do that, simply click the blue Bluetooth button and a pop-up will appear asking to pair. Select your BLE Sense, which

should be called something like “BLESense-XXXX” where XXXX will vary, and click the pair button. **Do note that the course staff has found that sometimes you have to repeat this step twice**. Once you are connected, the Bluetooth button will turn green.



ling magic wand gestures:

c Wand Capture sketch to an Arduino
oard using the Bluetooth button below
o make gestures. They'll be recorded a
ures, add labels by clicking on the '?', a
stures as a JSON data file, ready for m

4. Once your Arduino is paired, it’s time to record some gestures. You’ll notice that every time you move the Arduino around and then stop, a new gesture is recorded. This is because the gestures are automatically split up by times when the wand is kept still. These pauses act like spaces between words, and so when you've finished, a gesture you should stop moving the wand, so that it ends cleanly. **Note that the direction of the gesture matters (e.g., a clockwise circle is different from a counterclockwise circle)!**

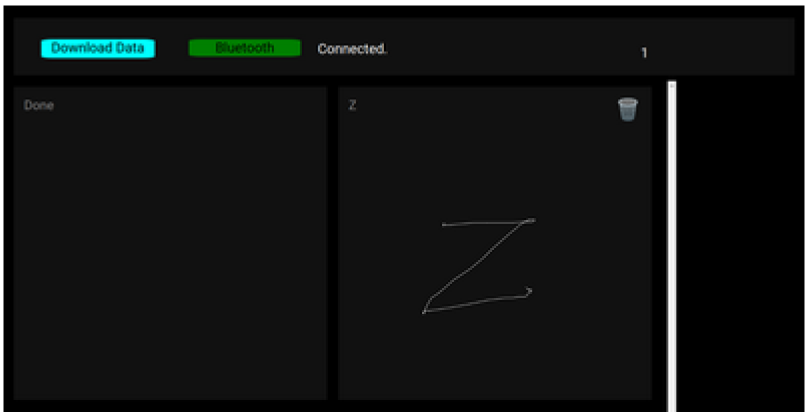
   These gestures you are drawing will start to show up in the list on the right side of the screen. You can look at the shapes shown there to understand whether the gestures came out cleanly. **A good rule of thumb is that if you can't tell what the gesture is by looking at it, then a model will have a hard time recognizing it too.**

   If you want to delete a recording, simply click the trashcan icon at the top right of each gesture recording (You may need to delete a lot of spurious recordings that were made as you moved the Arduino into position between each gesture).

   Also, make sure to label all of your gestures for training. You can label each recording by clicking on the question mark at the top left of each gesture and typing in your label. For example, the screenshot below shows the label “Z” added to the gesture of the letter “Z” recorded by the course staff. **Make sure you label all of your gestures and note that case matters (i.e., "z" and "Z" are different)!** Some students have noted that it can be easier to create one gesture per recording session in order to make sure all of the gestures are labeled identically but/and the staff has also gotten it to work with multiple gestures per recording session.

   The staff has found that collecting ~20 examples each of 2-3 different gestures often will be enough data to successfully train a moderately decent magic wand application (i.e., it will work often for you may not generalize to other users). To help you keep track of how many gestures you recorded, there is a number in the top right of the screen (e.g., the number 1 as shown in the image below). The staff has also found that gestures like a circle (O) or the (Z) for Zoro tend to work quite well!

Finally, you can upload multiple JSON files to the training script so don't feel pressured to do all of your gesture recordings in one shot!



5. When you are done collecting all of your data, simply click the blue "Download Data" button and a JSON file with all of the gestures will be automatically downloaded! We'll use that file in the Colab in the next section to train a custom model! **Be careful, when you leave or refresh the web page, your recorded gestures will be lost, so make sure you use the "Download Data" link to save them!**

Note: If you encounter an error where the number of gestures does not match the number of labels, check to ensure the data collection tool correctly labeled your gestures. Open the JSON file and check for "label":"" and replace all of the mislabeled gestures.

Training and Deploying Your Custom Magic Wand Project

In this document, we are going to train and then deploy a custom magic wand model based on the custom gestures we just collected.

03_TinyML_C03_04-08-11_final-en.mp4 (Command Line)

Training the Magic Wand Model

The first thing you'll need to do is to upload your gesture dataset into Colab and train a new magic wand model. Then in that Colab, we'll need to convert that model first into a quantized .tflite file and then into a .cc file for use with the Arduino IDE.

As with the KWS examples, we will be using the resulting .cc file, so make sure to download it or leave the tab open with the printout!

https://colab.research.google.com/github/tinyMLx/colabs/blob/master/4-8-11-CustomMagicWand.ipynb

Deploying the Trained Model

1. Use a USB cable to connect the Arduino Nano 33 BLE Sense to your machine. You should see the green LED power indicator come on when the board first receives power.
2. Open the magic_wand.ino sketch, which you can find via the File drop-down menu. Navigate, as follows: File → Examples → Harvard_TinyMLx → magic_wand.
3. You'll then need to make two changes to the magic_wand.ino file to alert it of your number of gestures and gesture labels. These changes occur on lines 55-59, which currently read as:

```
// -------------------------------------------------------------------------------- //
// UPDATE THESE VARIABLES TO MATCH THE NUMBER AND LIST OF GESTURES IN
YOUR DATASET //
// -------------------------------------------------------------------------------- //
constexpr int label_count = 10;
const char * labels[label_count] = {"0","1","2","3","4","5","6","7","8","9"};
```

- Update the label_count to reflect the number of gestures in your dataset.
- Update the list of labels to reflect the gestures in your dataset. **Note: the order matters!** Make sure it matches the alphanumeric order as printed out in the training script!

4. You'll also need to update the model data as we've done with the previous examples. The model data can be found in the magic_wand_model_data.cpp file. As we've done in the past, make sure to only update the binary values and leave the cpp syntax constant.
5. When you save, just like with the KWS examples, you will be asked to save a copy. Again, we suggest that you make a folder called e.g., TinyML inside of your Arduino folder. You can find your main Arduino folder either inside of your Documents folder or in your Home folder, and save it in that folder with a descriptive name like magic_wand_custom. That said, you can save it wherever you like with whatever name you want!
6. As always, use the Tools drop-down menu to select appropriate Port and Board.

- Select the Arduino Nano 33 BLE as the board by going to Tools → Board: <Current Board Name> → Arduino Mbed OS Boards (nRF52840) → Arduino Nano 33 BLE. Note that on different operating systems the exact name of the board may vary but/and it should include the word Nano at a minimum. If you do not see that as an option, then please go back to Setting up the Software and make sure you have installed the necessary board files.
- Then. select the USB Port associated with your board. This will appear differently on Windows, macOS, Linux but will likely indicate 'Arduino Nano 33 BLE" in parenthesis. You can select this by going to Tools → Port: <Current Port (Board on Port)> → <TBD Based on OS> (Arduino Nano 33 BLE). Where <TBD Based on OS> is most likely to come from the list below where <#> indicates some integer number:
    - Windows → COM<#>
    - macOS → /dev/cu.usbmodem<#>

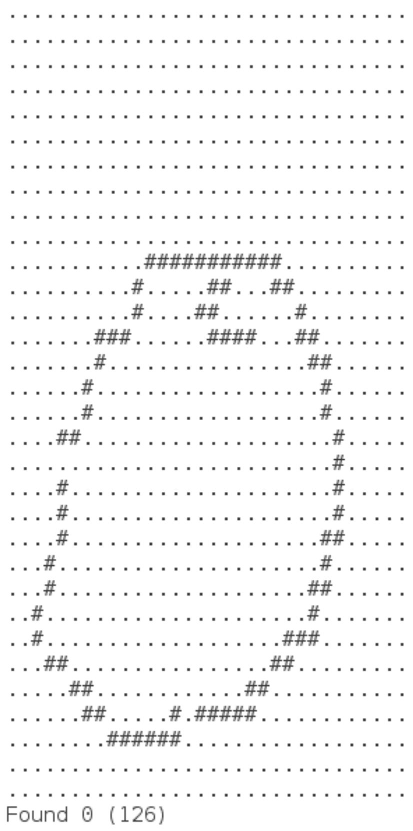
- Linux → ttyUSB<#> or ttyACM<#>

7. Use the rightward arrow to upload / flash the code. Do not be alarmed if you see a series of orange warnings appear in the console. This is expected as we are working with bleeding edge code. You'll know the upload is complete when you see red text in the console at the bottom of the IDE that shows 100% upload of the code and a statement that says something like "Done in <#.#> seconds."

   If you receive an error, you will see an orange error bar appear and a red error message in the console. Don't worry -- there are many common reasons this may have occurred.

   To help you debug other issues, please check out our FAQ appendix with answers to the most common errors!

8. Now, open the serial monitor and test out your custom model. As a reminder, **the serial monitor will output first ASCII art showing the gesture you just performed and below it will be the best match label as well as a confidence score between -128 and 127**. The confidence score indicates how strongly the model believes you performed the gesture. Do note that every time you move the board and then stop, a new gesture will be processed, so don't be surprised to get some odd results as you move the board to prepare for a gesture.



```
................................
................................
................................
................................
................................
................................
................................
................................
................................
................................
...........###########..........
..........#.....##...##.........
..........#....##......#........
.......###......####...##.......
.......#................##......
......#..................#......
......#..................#......
....##....................#.....
..........................#.....
....#.....................#.....
....#.....................#.....
....#....................##.....
...#.....................#......
...#....................##......
..#.....................#.......
..#...................###.......
...##................##.........
.....##............##...........
......##.....#.#####............
........######..................
................................
................................
Found 0 (126)
```

1.9: Responsible AI Deployment

Privacy

03_TinyML_C03_04-09-01_final-en.mp4 (Command Line)

Security

03_TinyML_C03_04-09-04_final-en.mp4 (Command Line)

Attacking a KWS Model

Now that we've talked about the need for security considerations with TinyML applications it's time for you to get some hands-on experience attacking a KWS model! Open up the Colab below to maliciously trigger the "Yes, No" KWS model with some seemingly innocent static noise!

https://colab.research.google.com/github/tinyMLx/colabs/blob/master/4-9-6-Attacking-KWSModel.ipynb

Why do ML Models Fail after Deployment?

Imagine you train and validate a machine learning (ML) model with high predictive accuracy and you have determined that it's ready for deployment. Once your model is deployed in the real world, you might think that this is cause for celebration as your hard work is now complete! However, the journey isn't over quite yet. When it comes to machine learning, you will need to continuously monitor your model's performance in the real world to ensure that it is still performing as well as it did when you validated its accuracy in production.

The reason ML models must be monitored after deployment is owing to the possibility of *model drift*, sometimes referred to as 'model decay', where the model degrades over time leading to a drop-off in predictive power. Why does model drift occur? Put simply, every model is created with the assumption that the way the world will be in the future is similar to how we interpreted the world at the time the model was developed. Of course, this assumption is not always warranted, as the real world is a dynamic environment that is constantly evolving.

When a model is deployed in an environment that changes in unforeseen ways, the model's performance is likely to change as well. In order to maintain the predictive accuracy of a model, successful deployment requires continuous monitoring over time to guard against model drift. Otherwise, the deployment of a model may lead to failure, with unhappy customers complaining that the model doesn't work as promised.

David Talby recounts his own experience with model drift in the article, "Why Machine Learning Models Crash and Burn in Production," in hopes that others may learn from his mistakes. After he deployed an ML model in hospitals to predict 30-day readmissions, he discovered a drop-off in performance after just 2-3 months. Moreover, the particular causes of model drift varied between hospitals, "or even buildings within the same hospital." For example, in some cases, the problem was 'missing values' owing to changes in the electronic health record system. In other cases, the population of patients presenting to the ER had shifted after the hospital began accepting a new type of insurance.

Talby's experience is informative as it shows how a variety of factors can contribute to model drift. At this point, we will shift our attention to learning how to distinguish between the two most common causes of model drift: data drift and concept drift.

Data Drift

When data drift occurs, the properties of the data inputs are different from the data that was used to train the model. Similar to how Talby saw a change in the population of patients presenting to the ER, it's easy to imagine how the distribution of data may change with respect to TinyML applications.

Let's suppose you are creating a keyword spotting application to activate a device. Since the intended context for deployment consists almost entirely of native English speakers, you acquire data from this population to train your model. Fast forward to a few months after deployment--the model's accuracy has held steady at around 85%. Now, let's suppose the context has suddenly changed, as the environment the device was deployed in has become a popular tourist destination. Since your model is encountering a large number of non-native English speakers with thick accents, the model's performance has dropped to 60%. This example shows how an unforeseen change in the real world led to a change in the distribution of data inputs, subsequently causing data drift.

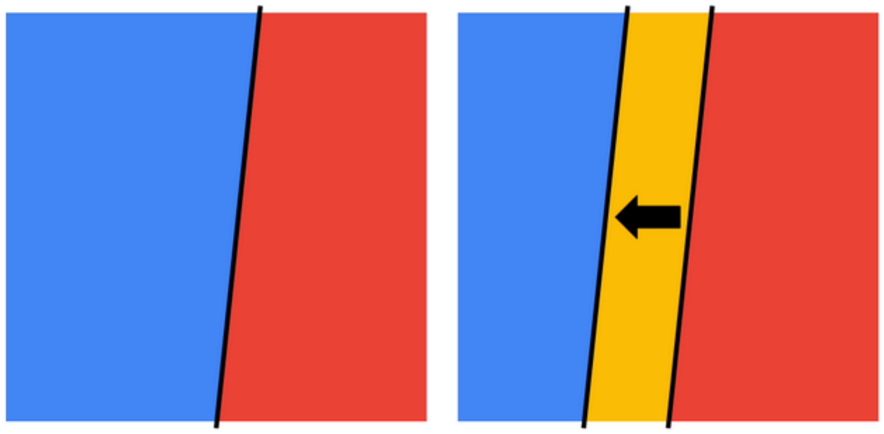
In addition to changes in the data inputs due to externalities, *upstream data changes* can also contribute to data drift. This is especially relevant to TinyML which exists at the intersection of machine learning and embedded IoT devices. On-device machine learning requires data inputs from sensors, and these sensors can raise a number of problems. For example, if a sensor is broken this might result in missing values that are required for making a prediction. Alternatively, the sensor itself may still be functioning properly but nevertheless result in faulty data inputs, e.g. if a camera lens is smudged and produces blurry images. Moreover, the sensor is used on the deployed device may be different than the sensor used to acquire the training data, and this can result in measurement changes, e.g. if the training data came from a high-resolution camera but the model is deployed on a device with a low-resolution camera.

Concept Drift

When concept drift occurs, the statistical properties of the target variable (which is what the model has been designed to predict) have changed. In other words, the actual relationship between the data inputs and outputs is different from what the model has learned. It is worth highlighting that concept drift can occur even when the distribution of incoming data remains the same and still accurately resembles the data used to train the model.

In Course 2, we discussed how defining the target variable is more like an art form than a perfect science, because it is a subjective process wherein one attempts to define some concept in quantitative measures e.g. defining 'healthy' in terms of measurable features like heart rate, number of steps, etc. Similar to how we might worry about how bias can influence the definition of the target variable, we also need to consider how the definition of the target variable might change or evolve over time in unforeseen ways.

For example, imagine we create a TinyML application to predict whether a plant is diseased by identifying features like yellow spotting and wilting leaves (see the left panel of the figure below). Now let's suppose that advances in research have led to a better understanding of this plant disease by identifying a new feature for accurately diagnosing it. As a result, some plants that would normally be classified as 'healthy' are now classified as 'diseased'. Since the concept of 'disease' has changed, our model's decision boundary is no longer accurate (see the right panel of the figure below). As a result, our model will incorrectly label data that falls between the decision boundary our model has learned and where the decision boundary is in reality.


The original decision boundary between the **red** and **blue** populations will misclassify the **orange** population as **blue** even though it should be **red** after the true boundary (the black line) shifts to the left.

How should model drift be addressed?

Now that you have gained familiarity with model drift and its common causes, the next step is learning how to detect it and address it. Proceed to the next video to learn more about model drift with a special focus on how to address it in the context of TinyML.

Monitoring after Deployment

03_TinyML_C03_04-09-09_final-en.mp4 (Command Line)

Congratulations! You Made it to the Finish Line!

03_TinyML_C03_04-10-01_final-en.mp4 (Command Line)

What Comes Next: Advanced Topics in TinyML

Throughout this course, you have developed multiple tinyML projects on your local microcontroller. You have designed a keyword spotting algorithm that was trained using your own customized dataset. You have designed a visual wake words algorithm and then, through transfer learning, adapted this to create a mask detection algorithm. You have even created your own gesturing magic wand!

Although you have now come to the end of this class, your journey as a tinyML engineer is only just beginning. You are now equipped with a set of tools and techniques that will be invaluable in the coming years as the field of tinyML reaches ubiquity and is incorporated beyond academia into commercial markets. The exponential increase of IoT devices, increasing prevalence of sensor networks, and transition towards industry 4.0 will continue to drive further development of tinyML at an accelerating pace. Consequently, keeping up-to-date with state-of-the-art methods, as well as advanced and novel techniques will be a necessity for your success as a tinyML engineer.

In this reading, we will touch on some of the major topics that we consider “advanced” as well as potential future developments and applications which may become relevant in the near future. Such predictions are, at best, informed guesses about how the field may evolve and may deviate depending on the conflicting needs of industry, end-users, and academia. However, we believe that these topics may act as a helpful stepping stone to you in your continuing development as a tinyML engineer.

Profiling and Benchmarking

Profiling is used in software engineering to optimize software performance. Similarly, in tinyML, profiling can be used to optimize a system for its deployment. This might include finding bottlenecks, computational inefficiencies, along with other properties which may negatively impact algorithmic performance. Profiling is typically used to determine where speedup can be achieved to aid in minimizing the computational costs of running an algorithm.

To accompany this, **benchmarking** is used to provide like-for-like comparisons between an algorithmic implementation running on different hardware, different implementations running on the same hardware, and so on. Benchmarking typically involves the use of comparative metrics, which for tinyML might include inference time, operations per second, and memory utilization.

Benchmarking and profiling will become increasingly important for commercial applications where large numbers of devices are running the same code since inefficiencies lead to higher costs and other undesirable effects. TinyMLPerf is a recent attempt to help provide a set of benchmarks for performing comparative measurements of typical tinyML tasks.

ML Operations (MLOps)

MLOps is very closely related to DevOps. DevOps focuses on seamless integration of automated testing and benchmarking capabilities to a software engineering environment, allowing updated code and algorithms to be automatically checked for errors and assessed for performance. MLOps would apply this to our tinyML algorithms, which would significantly increase the productivity of developers by providing real-time feedback which can be tailored to the deployment environment visible to the end-user. Several companies focusing on MLOps already exist, and there will likely be more as market penetration of tinyML applications increases.

Edge Impulse is one such example, which makes it easy to gather your dataset, train models, optimize and deploy them. Now that you know what is underneath the hood, you can go ahead and use such tools with good progress.

Cross-platform Interoperability

Algorithm interoperability across various hardware platforms remains an important barrier to tinyML. Microcontroller devices built upon different instruction sets may have (1) different sensor capabilities, (2) be bare-metal as opposed to having a lightweight operating system, (3) different memory capabilities, with or without a memory controller, and (4) different computational capabilities (e.g., clock speed, word lengths). These differences lead to a panoply of challenges for creating algorithms that can run on various hardware platforms.

Several attempts at overcoming this issue have already been attempted. OctoML provides a compiler stack that acts as a middleman that provides interoperability between different deep learning platforms and microcontroller devices. This challenge will become vital to overcoming once the tinyML ecosystem begins to propagate since multiple vendors will have to have products that can work together seamlessly, similar to how different computers and devices are able to interact on the internet today.

Neural Architecture Search

As we have seen throughout the course, it is difficult to find the optimal network configuration for a given application and dataset. Neural Architecture Search (NAS) is a common method used in machine learning to search and test multiple architectures in an attempt to find an optimal configuration automatically. However, we are more resource-constrained with tinyML devices, which must be incorporated into our NAS. Therefore, for tinyML devices, we must recast NAS to produce an optimal configuration given the hardware constraints imposed by our system, such that we can eke out the maximal performance.

Several attempts have been made so far to do this, such as with MCUNet and MicroNets. The commercial drivers for this are strong since devices with higher performance capabilities are preferable in any given application.

No doubt, there will be more items that could be added to this list in the near future, but we hope this will provide you with a valuable starting point for finding additional topics that may be useful for you in future applications.

What Do I Do Now?

03_TinyML_C03_04-10-03_final-en.mp4 (Command Line)

TinyMLx Project Extension (Optional)

At this point, you've amassed an incredible amount of knowledge on Tiny Machine Learning. You have seen how models are developed using the machine learning workflow, from data collection to training and deployment. You have seen how models are produced using TensorFlow and can be ported into light-weight frameworks, like TensorFlow Lite and Micro. You have also seen how these models can interact with various forms of input data, such as time-series data from sensors and microphones, to images from a camera.

In Course 3, we have looked closely at how to deploy TinyML models within embedded systems, using the Arduino Nano 33 BLE Sense microcontroller development board as a representative host. Further still, you have already leveraged TensorFlow Micro to deploy a range of TinyML applications on heavily resource-constrained hardware, creating a roadmap for end-to-end deployment.

We want everyone who completes the TinyML edX course to be confident that they can build custom TinyML applications using the knowledge they have gained. As such, we invite you all to join the public TinyMLx community on Discourse and to design your own TinyML projects beyond the scope of this class.

To help motivate you and foster assimilation to the wider TinyMLx community, **we've set up an open TinyMLx project competition with a number of prizes for exemplary work**. This project represents an opportunity to demonstrate what you have learned and we hope will give you an incentive and a forum to push beyond the course examples to introduce new use cases and applications of TinyML. We can't wait to see what you'll design and build collaboratively as a part of the growing TinyMLx community.

For more information about the optional project extension including FAQs, prizes, and submission details, please follow this link to Vijay's Open Call for Projects on Discourse.

Appendix

You may have noticed throughout the course that we have provided appendix links with additional supplementary material on various topics (e.g., powering your TinyML kit with a battery, details on serial protocols). All of those documents can be found in our appendix GitHub repository: https://github.com/tinyMLx/appendix