Optional Lab: Deploying a Multi-Tenant Application

In this optional lab, you will deploy the pretrained multi-tenant KWS / person detection model onto your Arduino and walk through a little bit of the code to better understand how it works.

Deploying a Multi-Tenant Application

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 → (INCOMPATIBLE) → 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:

    a) 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.
    b) 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" inparenthesis. 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

        i. Windows → COM<#>
        ii. macOS → /dev/cu.usbmodem<#>
        iii. 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."
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.

    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.’

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().

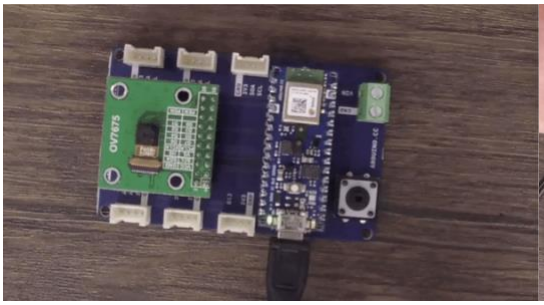
```
static tflite::MicroInterpreter vww_static_interpreter(
   vww_model, micro_op_resolver, allocator, error_reporter);
vww_interpreter = &vww_static_interpreter;

//allocate the VWW model from tensor_arena
TfLiteStatus allocate_status = vww_interpreter->AllocateTensors();
```

Next, we do the same for the micro speech interpreter. Again passing the allocator to it and then calling AllocateTensors().

```
static tflite::MicroInterpreter kws_static_interpreter(
    kws_model, micro_op_resolver, allocator, error_reporter);
kws_interpreter = &kws_static_interpreter;

//allocate the KWS model from tensor_arena. Some head space is saved
allocate_status = kws_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);
  }
```