CEC Continuing Education Center

DesignNews

Getting Started in TinyML with Arduino

DAY 2 : Build/Train a TinyML Model for Arduino in TensorFlow

Sponsored by

DigiKey

informa markets

Webinar Logistics

- Turn on your system sound to hear the streaming presentation.
- If you have technical problems, click “Help” or submit a question asking for assistance.
- Participate in ‘Attendee Chat’ by maximizing the chat widget in your dock.

Dr. Don Wilcher

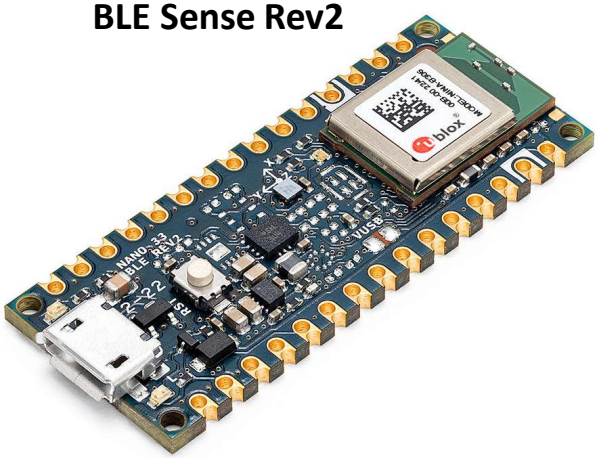
Visit ‘Lecturer Profile’ in your console for more details.

LinkedIn Page:
https://www.linkedin.com/in/dr-don-wilcher-ed-d-mseit-ee-ceta-2735151/

Patreon Page:
https://www.patreon.com/c/DrDon683

Course Kit and Materials

Arduino Nano 33 BLE Sense Rev2



Research Literature and Documentation

Tiny Machine Learning: Progress and Futures

Ji Lin Ligeng Zhu Wei-Ming Chen Wei-Chen Wang Song Han

Massachusetts Institute of Technology

https://tinyml.mit.edu

Abstract—Tiny Machine Learning (TinyML) is a new frontier of machine learning. By squeezing deep learning models into billions of IoT devices and microcontrollers (MCUs), we expand the scope of AI applications and enable ubiquitous intelligence. However, TinyML is challenging due to hardware constraints: the tiny memory resource makes it difficult to hold deep learning models designed for cloud and mobile platforms. There is also limited compiler and inference engine support for bare-metal devices. Therefore, we need to co-design the algorithm and system stack to enable TinyML. In this review, we will first discuss the definition, challenges, and applications of TinyML. We then survey the recent progress in TinyML and deep learning on MCUs. Next, we will introduce MCUNet, showing how we can achieve ImageNet-scale AI applications on IoT devices with *system-algorithm co-design*. We will further *extend* the solution from *inference to training* and introduce tiny on-device training techniques. Finally, we present future directions in this area. Today's "large" model might be tomorrow's "tiny" model. The scope of TinyML should evolve and adapt over time.

Index Terms—TinyML, Efficient Deep Learning, On-Device Training, Learning on the Edge

I. Overview of Tiny Machine Learning

Machine learning (ML) has made significant impacts on various fields, including vision, language, and audio. However, state-of-the-art models often come at the cost of high computation and memory, making them expensive to deploy. To address this, researchers have been working on efficient algorithms, systems, and hardware to reduce the cost of machine learning models in various deployment scenarios. There are two main subdomains of efficient ML: EdgeML and CloudML (Figure 1). While CloudML focuses on improving latency and throughput on cloud servers, EdgeML focuses on improving energy efficiency, latency, and privacy on edge devices. These two domains also intersect in areas such as hybrid inference [1, 2], over-the-air (OTA) updates, and federated learning between the edge and cloud [3]. In recent years, there has been significant progress in extending the scope of EdgeML to ultra-low-power devices such as IoT devices and microcontrollers, known as **TinyML**.

TinyML has several key advantages. It enables machine learning using only a few hundred kilobytes of memory which greatly reduces the cost. With billions of IoT devices producing more and more data in our daily lives, there is a growing need for low-power, always-on, on-device AI. By performing on-device inference near the sensor, TinyML enables better responsiveness and privacy while reducing the energy cost associated with wireless communication. On-device processing of data can be beneficial for applications where real-time decision-making is crucial, such as autonomous vehicles.

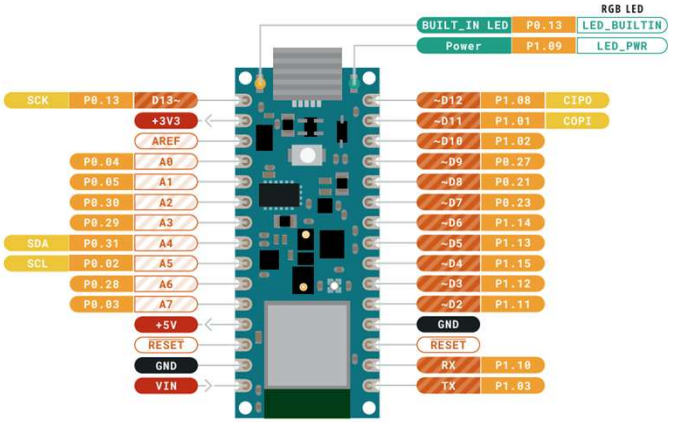
Fig. 1. Efficiency is critical for CloudML, EdgeML, and TinyML. CloudML targets high-throughput accelerators like GPUs, while EdgeML focuses on portable devices like mobile phones. TinyML further pushes the efficiency boundary, enabling powerful ML models to run on ultra-low-power devices such as microcontrollers.

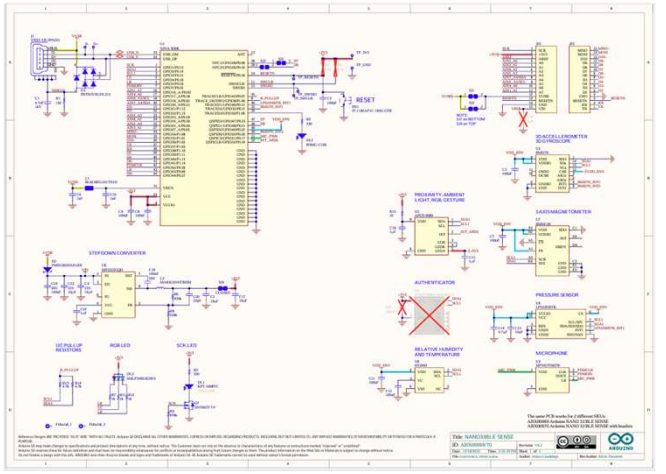
In addition to inference, we push the frontier of TinyML to enable on-device training on IoT devices. Irrevolutionizes EdgeAI through continuous and lifelong learning. Edge device can finetune the model on itself rather than transmitting data to cloud servers, which protects privacy. On-device learning has numerous benefits and a variety of applications. For example, home cameras can continuously recognize new faces, and email clients can gradually improve their prediction by updating customized language models. It also enables IoT applications that do not have a physical connection to the internet to adapt to the environment, such as precision agriculture and ocean sensing.

In this review, we will first discuss the definition and challenges of TinyML, analyzing why we can't directly scale mobile ML or cloud ML models for tinyML. Then we delve into the importance of system-algorithm co-design in TinyML. We will then survey recent literature and the progress of the field, presenting a holistic survey and comparison in Tables 1 and 10. Next, we will introduce our TinyML project, MCUNet, which combines efficient system and algorithm design to enable TinyML for both inference to training. Finally, we will discuss several emerging topics for future research directions in the field.

A. Challenges of TinyML

The success of deep learning models often comes at the cost of high computation, which is not feasible for use in TinyML applications due to the strict resource constraints of devices such as microcontrollers. Deploying and training AI models on MCU is extremely hard: No DRAM, no operating systems (OS), and strict memory constraints (SRAM is smaller than 256kB, and FLASH is read-only). The available resources on these devices are orders of magnitude smaller than those





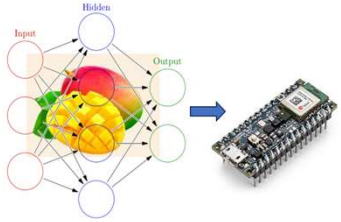
Research Perspective

"Today's large model might be tomorrow's *tiny model*."
[1] Lin et al., 2024

Agenda:

- What is TensorFlow?
- Common Terminology and Definitions
- What Can You Do With TensorFlow?
- Key Components In TensorFlow
- Lab: Sinewave Validation Device

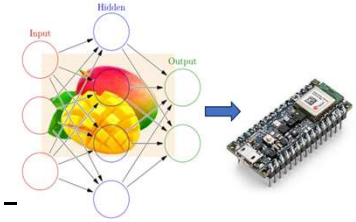
What is TensorFlow?



TensorFlow is an open-source machine learning and deep learning framework created by Google.

- It provides a unified environment to build, train, deploy, and run Machine Learning (ML) models
- The ML Models can vary from
  a) small linear regressions to
  b) very large neural networks used in:
    i. computer vision
    ii. natural language processing,
    iii. robotics, and more.

What is TensorFlow?. . .



- The Google Brain was started in 2011 to explore the use of very large-scale deep neural networks:
  a) research
  b) for use in Google products
- DistBelief was built as the first-generation scalable distributed-training and inference system.
- DistBelief work focused on the following items
  a) unsupervised learning
  b) language representation
  c) image and video classification models
  d) object detection
  e) speech recognition [2]

Question 1

What is TensorFlow?

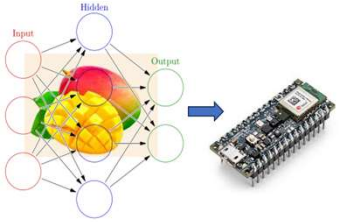
a) A closed-source machine learning and deep learning framework created by Google.

b) An open-source machine learning and deep learning framework created by Microsoft.

c) An open-source machine learning and deep learning framework created by Google.

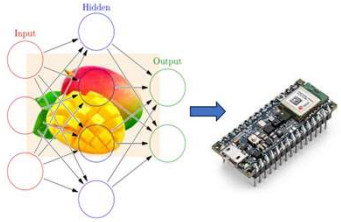
d) none of the above

What is TensorFlow?. . .



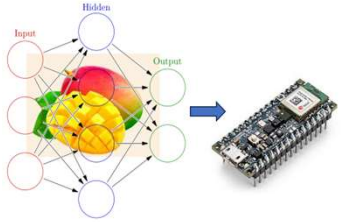
- In 2012 DistBelief was deployed in a variety of Google products, like:
  a) Google Search
  b) Speech Recognition
  c) Google Photos
  d) Google Maps
  e) Google Translate
  f) YouTube
- TensorFlow was built as the second-generation system for the implementation and deployment of large-scale machine learning models.

What is TensorFlow?. . .

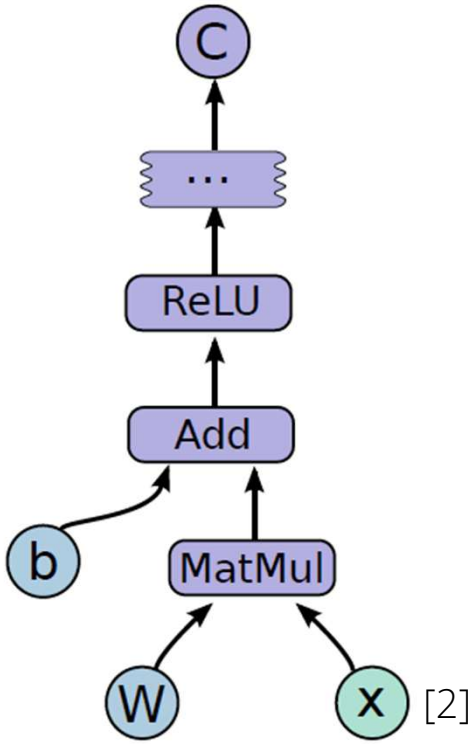


- TensorFlow uses computations described by a dataflow-like model and maps them onto a variety of different hardware platforms.
- TensorFlow usage ranges from running inference on mobile platforms like:
  a) Android
  b) iOS
  c) model size training and inference systems
- Graphics Processing Units (GPUs) are typically used on single-processor machines.
- Large-scale training systems using hundreds of specialized machines with thousands of GPUs

What is TensorFlow?. . .



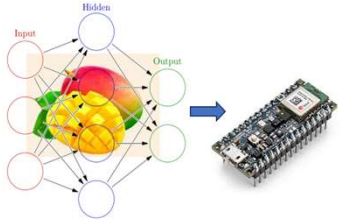
Example of a Computation Dataflow Graph



[2]

Note:
ReLU: Rectified Linear Unit-
An activation function
used in Deep Learning (DL).

Common Terminology and Definitions



What Is Inference?

Inference is the process of a trained model predicting new data.

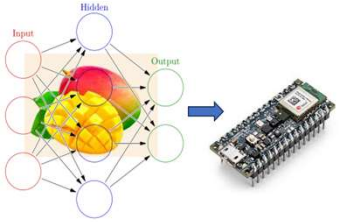
Once the model is trained:

a) You feed it a new sensor reading, audio signal, image, or IMU data

b) It computes outputs

c) You interpret the results (e.g., “gesture = up”)

Common Terminology and Definitions. . .



Inference in TinyML

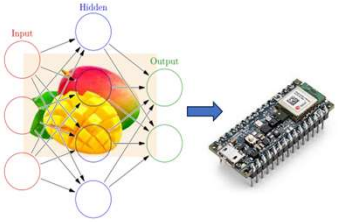
Inference happens on the microcontroller, not the cloud.

For example, with the Arduino Nano 33 BLE Sense:

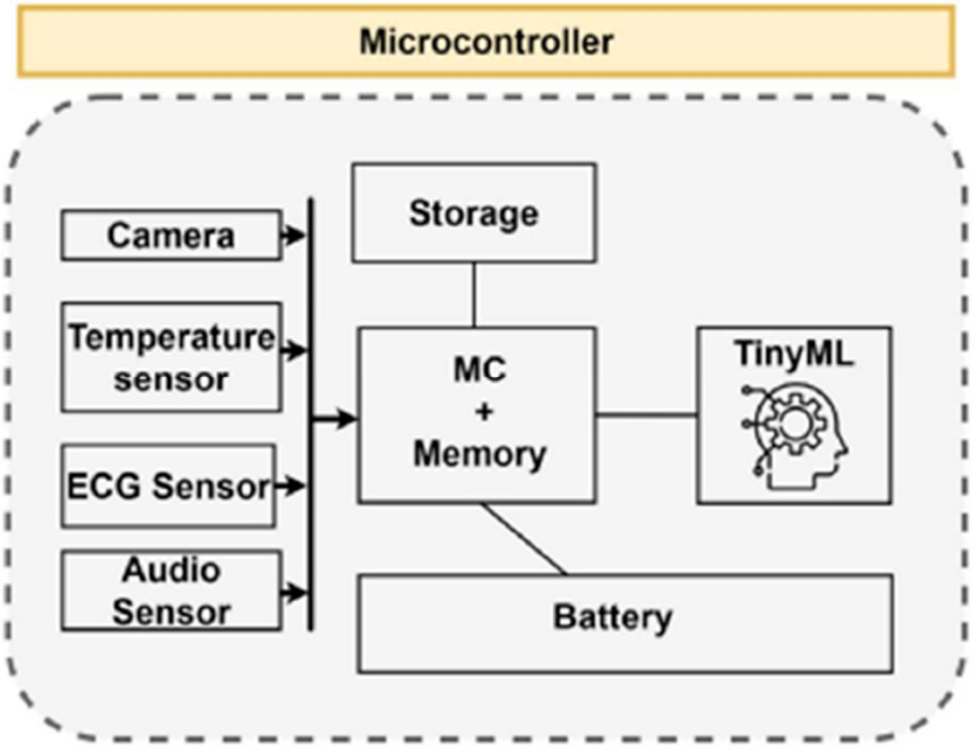
- Read 64 IMU samples
- Preprocess them
- Run the tiny neural network
- Output the predicted gesture
- Turn on an LED or send a BLE message

All of that is **one inference**.

Common Terminology and Definitions. . .



Inference in TinyML
Inference happens on the microcontroller, not the cloud.



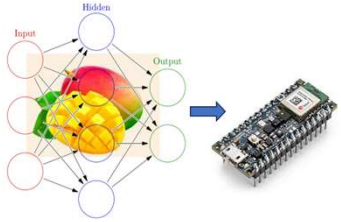
[3] Heydari & Mahmoud, 2025

Question 2

Which device inference runs on?

a) LoRaWAN
b) an integrated circuit
c) microcontroller
d) none of the above

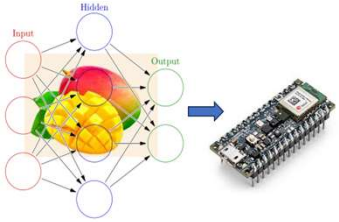
Common Terminology and Definitions…



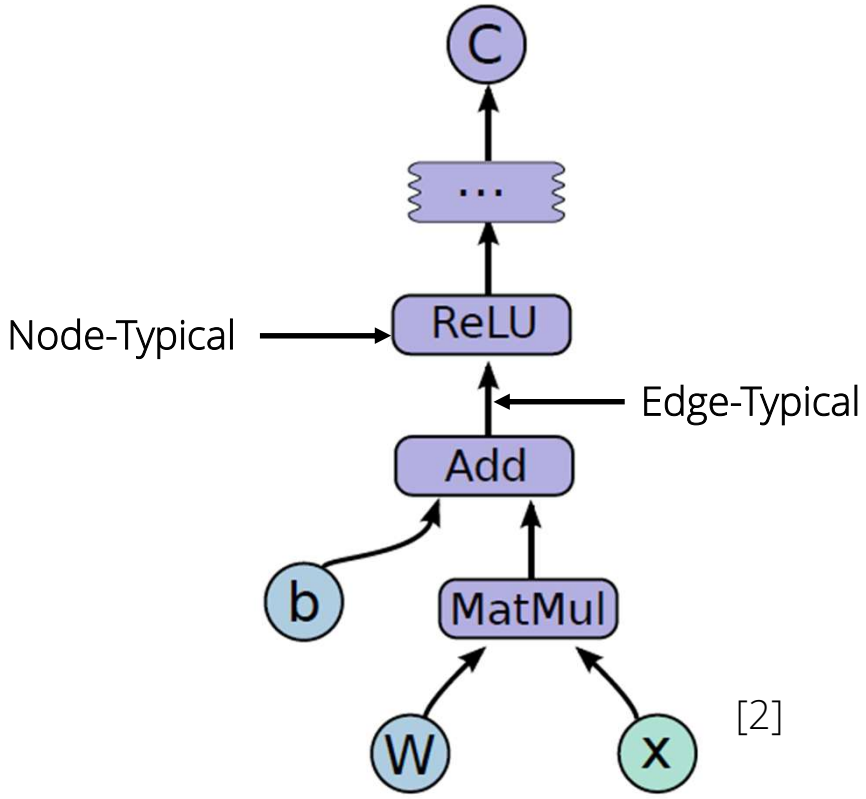
- TensorFlow is an interface for expressing ML algorithms, and an implementation for executing ML algorithms [3].
- TensorFlow represents computations as dataflow graphs:
  a) Nodes = mathematical operations (e.g., MatMul, Conv2D)
  b) Edges = multi-dimensional arrays called tensors

Note:
This is the fundamental structure used by TensorFlow to express machine learning computations.

Common Terminology and Definitions. . .

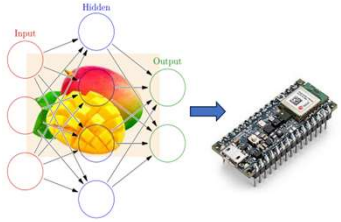


Anatomy of a Dataflow Graph



[2]

Common Terminology and Definitions…



Operation (Op)

A single computational unit in the graph (e.g., MatMul, Add, Conv2D).

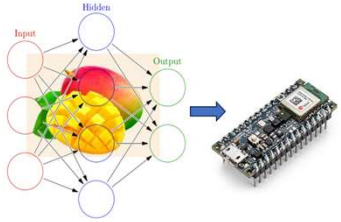
Nodes can have:

a) multiple inputs

b) multiple outputs

c) attributes that modify behavior

Common Terminology and Definitions. . .

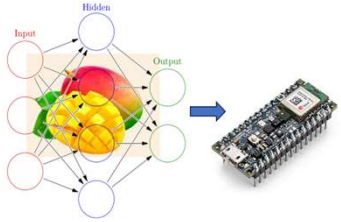


Kernel

a) A device-specific implementation of an operation (e.g., CPU kernel vs GPU kernel).

b) Multiple kernels may exist for the same operation, each optimized for a specific device.

Common Terminology and Definitions. . .



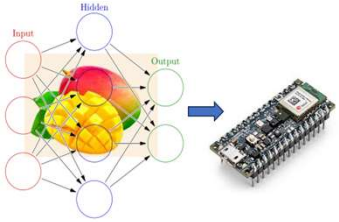
Session

A runtime environment used to execute parts of a dataflow graph on one or more devices.

The session handles:

a) graph execution
b) communication
c) memory management

What Can You Do With TensorFlow?



What You Can Do With TensorFlow

Build neural networks

Using APIs like:

a) `tf.keras` (high-level)

b) `tf.nn` (lower-level for custom layers)

Train and evaluate models

TensorFlow includes tools for:

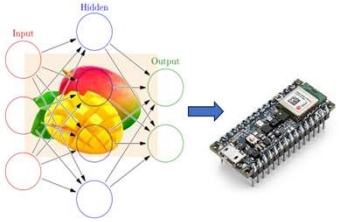
a) gradient calculation (`tf.GradientTape`)

b) backpropagation

c) optimizers (Adam, SGD, RMSprop)

d) loss functions

What Can You Do With TensorFlow?. . .

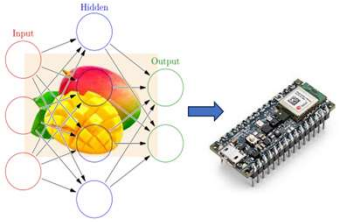


Do DL tasks

TensorFlow powers:

a) Image recognition
b) Speech recognition
c) Natural language models
d) Reinforcement learning
e) Time-series forecasting
f) Robotics control

What Can You Do With TensorFlow?. . .



Deploy models everywhere

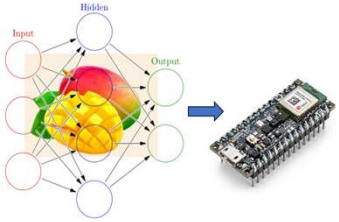
With TensorFlow, you can deploy to:

a) Servers
b) Mobile apps (TensorFlow Lite)
c) Web browsers (TensorFlow.js)
d) Embedded devices/microcontrollers (TensorFlow Lite Micro)

Note:

This makes TensorFlow suitable for TinyML and edge-AI applications as well.

What Can You Do With TensorFlow?. . .



Deploy models everywhere

With TensorFlow, you can deploy to:

a) Servers
b) Mobile apps (TensorFlow Lite)
c) Web browsers (TensorFlow.js)
d) Embedded devices/microcontrollers (TensorFlow Lite Micro)

Note:

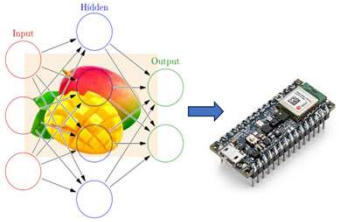
This makes TensorFlow suitable for TinyML and edge-AI applications as well.

Question 3

What can you do with TensorFlow?

a) Build an IT network
b) Train and evaluate systems
c) Build neural networks
d) none of the above

Key Components in TensorFlow



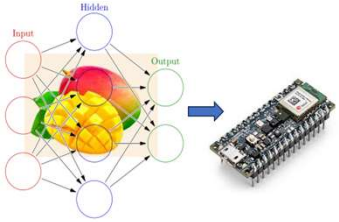
TensorFlow Core
The foundational API for tensors, ops, and math functions.

Keras (tf.keras)
The preferred high-level API for designing deep learning models.

Example structure:

```python
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(10)
])
```

Key Components in TensorFlow. . .



TensorFlow Lite

Allows running models on:

a) phones
b) microcontrollers (TinyML)
c) IoT devices

TensorFlow Model Optimization Toolkit

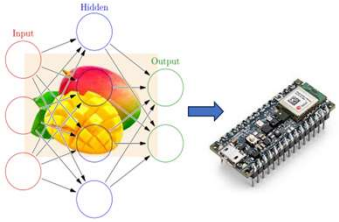
Tools for:

a) quantization
b) pruning
c) model compression

Note:

Essential for low-power devices like microcontrollers and small edge hardware.

Key Components in TensorFlow. . .

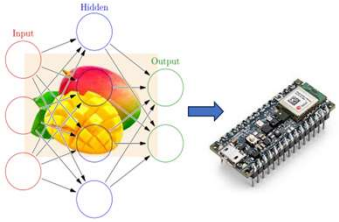


TensorBoard

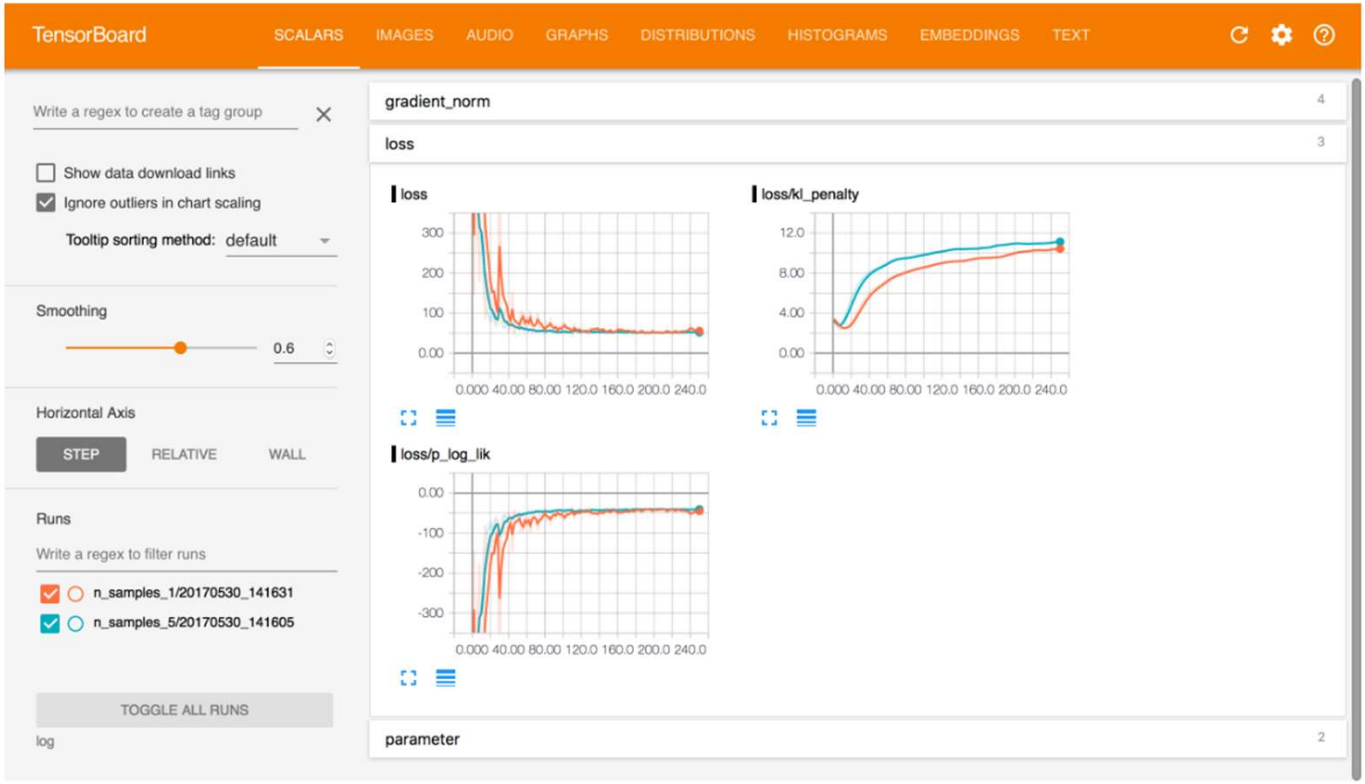
A visualization tool for:

a) training curves (loss/accuracy)
b) model graphs
c) histograms
d) embeddings
e) performance metrics

Key Components in TensorFlow. . .

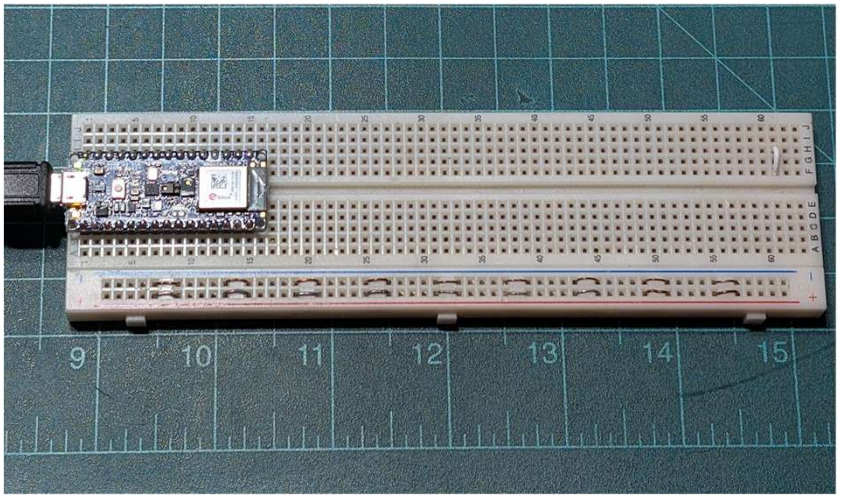


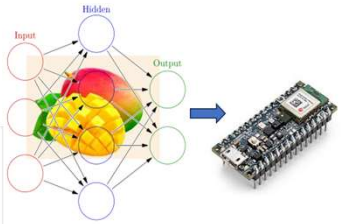
TensorBoard Example

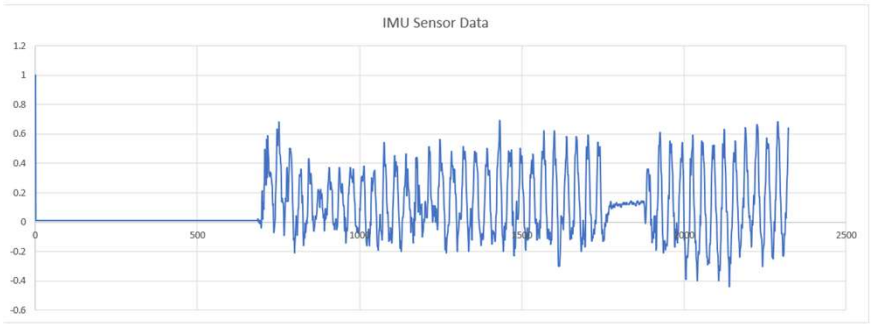


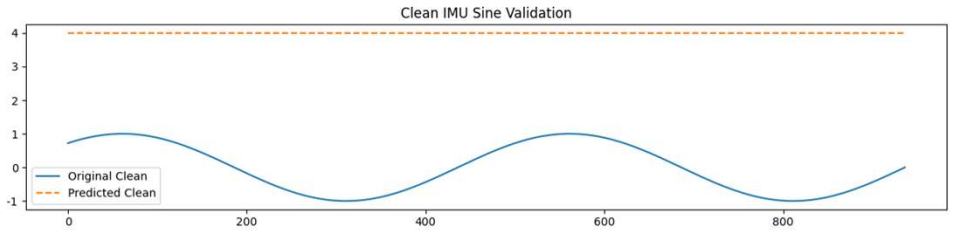
Edward

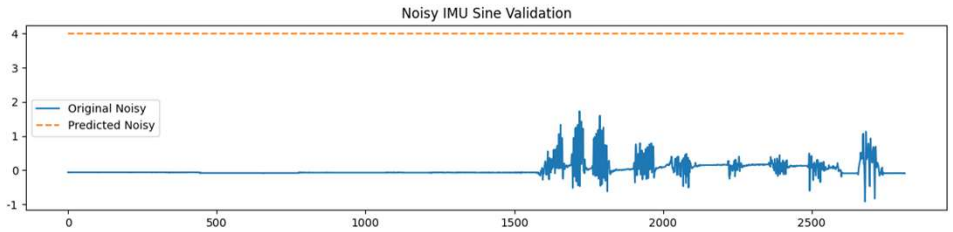
Lab: Sinewave Validation Device



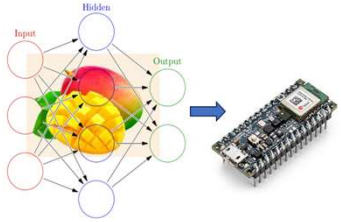








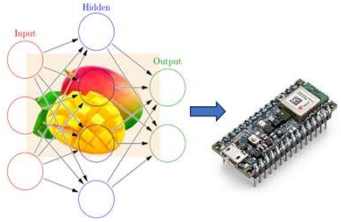
Lab: Sinewave Validation Device. . .



Lab Objectives:

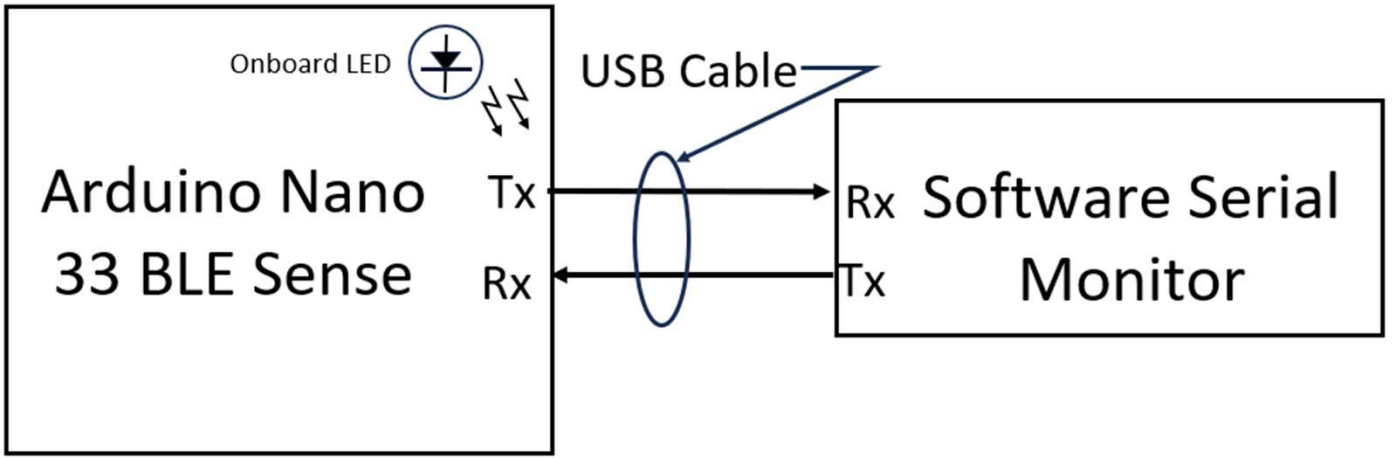
- Participants will learn to program the Arduino Nano 33 BLE Sense to obtain accelerometer data.
- Participants will learn to use a software terminal monitor to log accelerometer data obtained from the Arduino Nano 33 BLE Sense board.
- Participants will learn to program sine wave validation tools using TensorFlow and Python.
- Participants will learn to analyze the data produced by the TensorFlow sine wave validation..

Lab: Sinewave Validation Device. . .



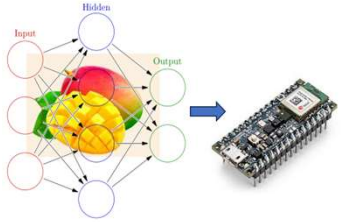
Concept System Block Diagram

Simple Data Logging Setup



Note:
Software Serial Monitor can be Tera Term or Putty

Lab: Sinewave Validation Device. . .



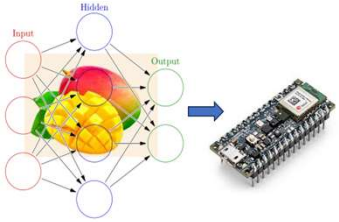
Software Code

Simple Data Logging Code for the Arduino Nano 33 BLE Sense

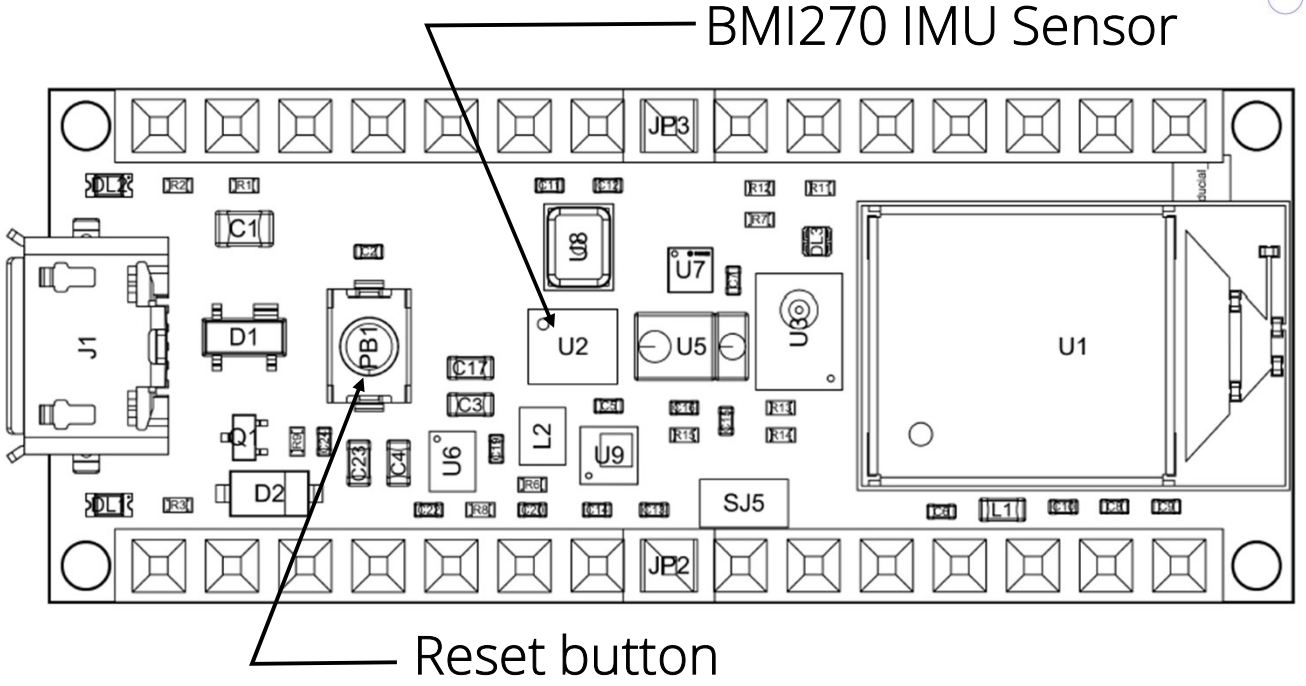
```
#include "Arduino_BMI270_BMM150.h"

void setup() {
  Serial.begin(115200);
  IMU.begin();
}

void loop() {
  float x, y, z;
  IMU.readAcceleration(x, y, z);
  Serial.println(x);
  delay(20);   // ~50 Hz sample rate
}
```

Lab: Sinewave Validation Device. . .

Software Code



To put Arduino Nano 33 BLE Sense into program mode, press the Reset button twice

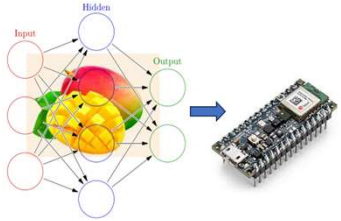


Question 4

In reviewing slide 35, which component is used to enable program mode?

a) IMU Sensor
b) USB cable
c) Reset button
d) none of the above

Lab: Sinewave Validation Device. . .
Sensor Output Data



IMU_Sensor.ino

```ino
#include "Arduino_BMI270_BMM150.h"

void setup() {
  Serial.begin(115200);
  IMU.begin();
}

void loop() {
  float x, y, z;
  IMU.readAcceleration(x, y, z);
  Serial.println(x);
  delay(20);   // ~50 Hz sample rate
}

```

Simple Data Logging Code for the Arduino Nano 33 BLE Sense

Output Serial Monitor ×

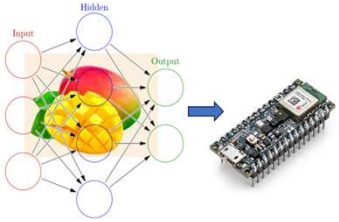
Message (Enter to send message to 'Arduino Nano 33 BLE' on 'COM12')

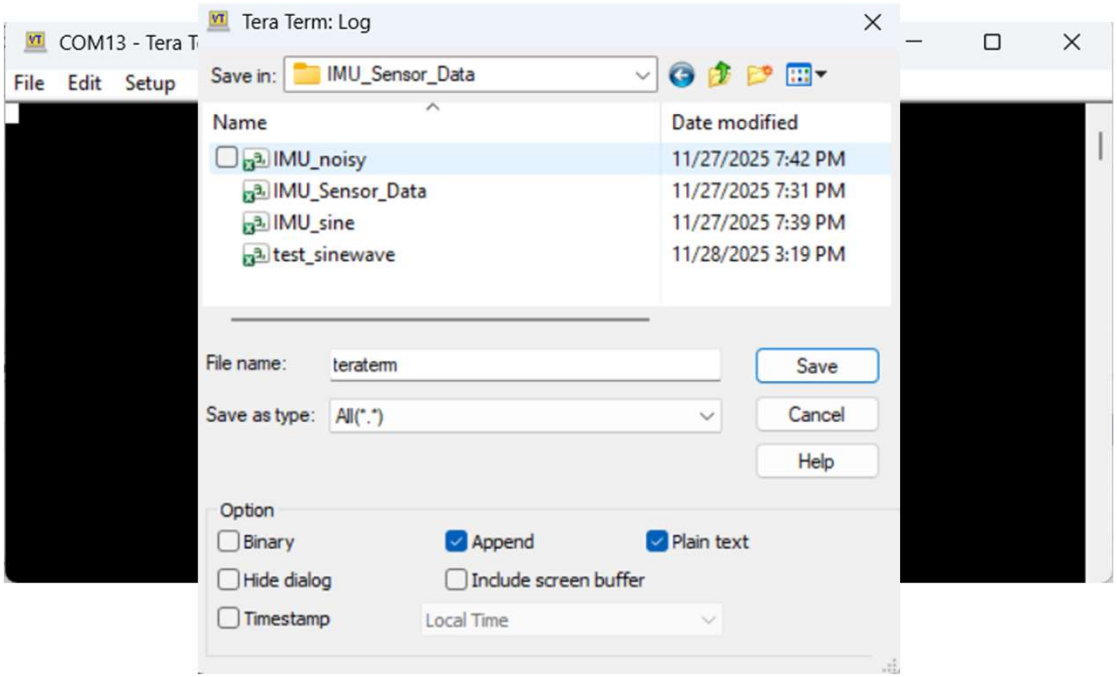
```
-0.08
-0.08
-0.08
-0.08
-0.08
-0.08
-0.08
-0.08
-0.08
-0.08
-0.08
-0.08
-0.08
-0.08
-0.08
```

Serial Monitor-IMU Sensor Data

Lab: Sinewave Validation Device. . .

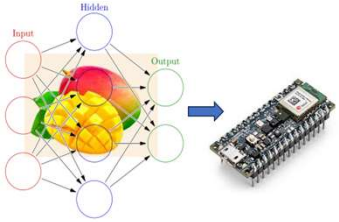
Data Logging Sessions



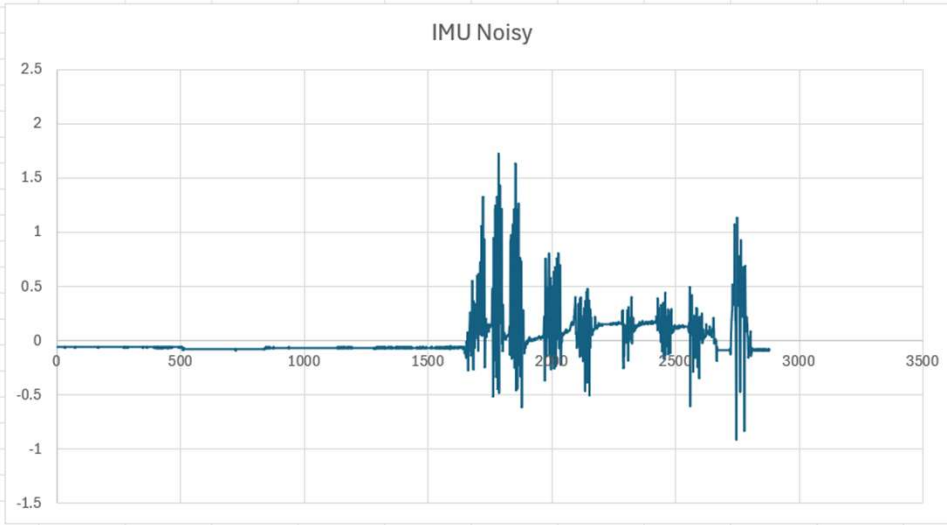


Tera Term Software Serial Monitor

Lab: Sinewave Validation Device. . .
Data Logging Session Example

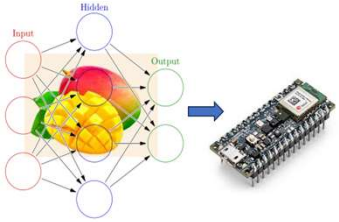


	A
1	-0.06
2	-0.06
3	-0.06
4	-0.06
5	-0.06
6	-0.06
7	-0.06
8	-0.06
9	-0.06
10	-0.06
11	-0.06
12	-0.06
13	-0.06
14	-0.06
15	-0.06
16	-0.06
17	-0.06
18	-0.06
19	-0.06
20	-0.06
21	-0.06
22	-0.06
23	-0.06
24	-0.06
25	-0.06

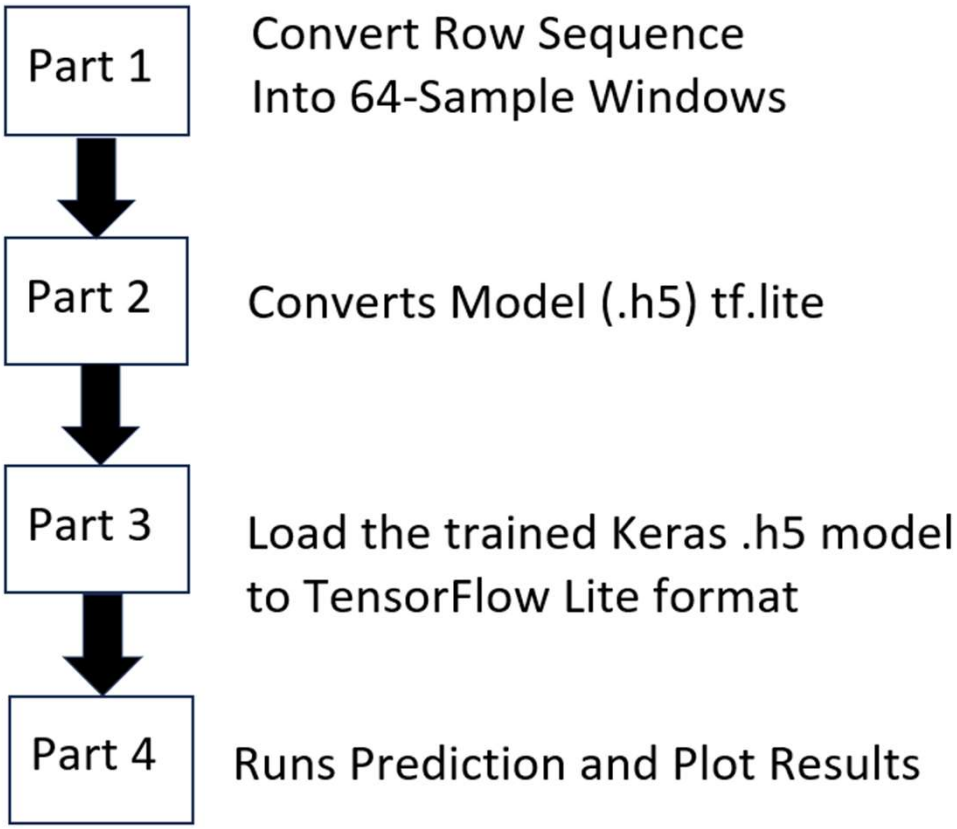


Noisy Data: Generated with Arduino Nano 33 BLE Sense IMU Sensor

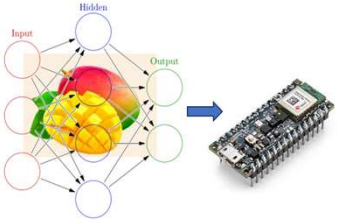
Lab: Sinewave Validation Device. . .



4 Part Python TensorFlow Code Partitioning:
Each Part is placed in a Google Colab programming cell.



Lab: Sinewave Validation Device. . .



Part 1

Convert Row Sequence
Into 64-Sample Windows

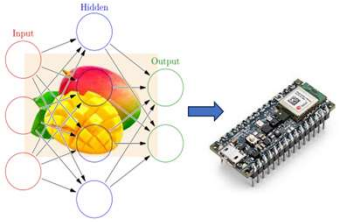
Python Partial Code:
Google Colab
Environment

```python
import numpy as np
import tensorflow as tf

# Load CSV data
sine_raw = np.loadtxt("IMU_sine.csv", delimiter=",")
noise_raw = np.loadtxt("IMU_noisy.csv", delimiter=",")

# ------------------------------
# FIX: Convert raw sequences into 64-sample windows
# ------------------------------

def create_windows(data, window_size=64, step=64):
    """
    Converts a 1D array into multiple fixed-length windows.
    step=64 means non-overlapping windows.
    """
    windows = []
    for i in range(0, len(data) - window_size + 1, step):
        window = data[i:i + window_size]
        windows.append(window)
    return np.array(windows)

sine = create_windows(sine_raw, 64, 64)
noise = create_windows(noise_raw, 64, 64)

print("Sine windows shape:", sine.shape)
```

Lab: Sinewave Validation Device. . .

Part 1

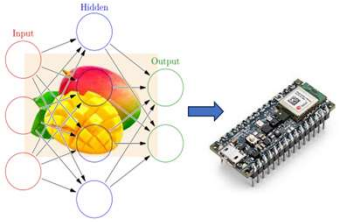
Convert Row Sequence
Into 64-Sample Windows



Epoch Results

```
Sine windows shape: (31, 64)
Noise windows shape: (44, 64)
Epoch 1/10
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass an `input_shape`/`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.6114 - loss: 0.6934
Epoch 2/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6615 - loss: 0.6772
Epoch 3/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.6431 - loss: 0.6686
Epoch 4/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6927 - loss: 0.6510
Epoch 5/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.6838 - loss: 0.6466
Epoch 6/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6720 - loss: 0.6502
Epoch 7/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6994 - loss: 0.6357
Epoch 8/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.6904 - loss: 0.6324
Epoch 9/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.7021 - loss: 0.6438
Epoch 10/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.7049 - loss: 0.6174
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is c
```

Lab: Sinewave Validation Device. . .



Part 2 Converts Model (.h5) tf.lite

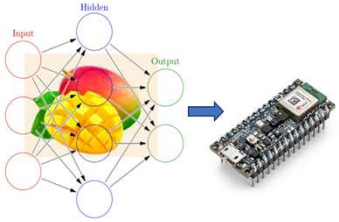
Python Partial Code: Google Colab Environment

```python
import tensorflow as tf

# Load model
model = tf.keras.models.load_model("sine_model.h5")

# Create converter
converter = tf.lite.TFLiteConverter.from_keras_model(model)

# (Optional) Quantization for TinyML
converter.optimizations = [tf.lite.Optimize.DEFAULT]

def representative_dataset():
    for i in range(100):
        yield [tf.random.uniform([1, 64, 1], dtype=tf.float32)]
converter.representative_dataset = representative_dataset

converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
converter.inference_input_type = tf.int8
converter.inference_output_type = tf.int8

# Convert
tflite_model = converter.convert()

# Save .tflite
with open("sine_model.tflite", "wb") as f:
    f.write(tflite_model)
```

Lab: Sinewave Validation Device. . .

Part 2

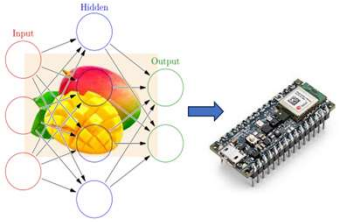
Converts Model (.h5) tf.lite



Conversion Results

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until y
Saved artifact at '/tmp/tmpfwaah_n5'. The following endpoints are available:

* Endpoint 'serve'
  args_0 (POSITIONAL_ONLY): TensorSpec(shape=(None, 64, 1), dtype=tf.float32, name='input_layer_3')
Output Type:
  TensorSpec(shape=(None, 1), dtype=tf.float32, name=None)
Captures:
  137697005519120: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137696991878160: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137696991883728: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137696991877968: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137696991875472: TensorSpec(shape=(), dtype=tf.resource, name=None)
  137696991875664: TensorSpec(shape=(), dtype=tf.resource, name=None)
Saved sine_model.tflite
/usr/local/lib/python3.12/dist-packages/tensorflow/lite/python/convert.py:854: UserWarning: Statistics for quantized inputs were exp
  warnings.warn(
```

Lab: Sinewave Validation Device. . .



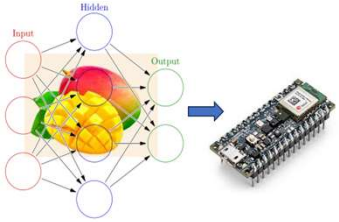
Part 3

Load the trained Keras .h5 model to TensorFlow Lite format

Python Partial Code: Google Colab Environment

```python
import tensorflow as tf

# ---------------------------------------------
#  STEP 1 – Load the trained Keras .h5 model
# ---------------------------------------------
model = tf.keras.models.load_model("sine_model.h5")
print("Loaded Keras model: sine_model.h5")

# ---------------------------------------------------
#  STEP 2 – Convert to TensorFlow Lite (quantized)
# ---------------------------------------------------
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]

# Representative dataset for quantization
def representative_dataset():
    for i in range(100):
        # Shape must match your Conv1D input shape
        yield [tf.random.uniform([1, 64, 1], dtype=tf.float32)]

converter.representative_dataset = representative_dataset

converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
converter.inference_input_type = tf.int8
converter.inference_output_type = tf.int8

```

Lab: Sinewave Validation Device. . .

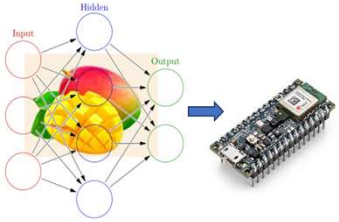


Part 4 Runs Prediction and Plot Results

Python Partial Code: Google Colab Environment

```python
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt

# ------------------------------
# Load CSV data
# ------------------------------
sine_df = pd.read_csv('IMU_sine.csv')
noisy_df = pd.read_csv('IMU_noisy.csv')

def extract_first_numeric_column(df):
    return df.select_dtypes(include=[np.number]).iloc[:,0].values

x_clean = extract_first_numeric_column(sine_df)
x_noisy = extract_first_numeric_column(noisy_df)

def normalize(data):
    return (data - np.min(data)) / (np.max(data) - np.min(data)) * 2 - 1

x_clean_norm = normalize(x_clean)
x_noisy_norm = normalize(x_noisy)

# ------------------------------
# Load TFLite model
# ------------------------------
interpreter = tf.lite.Interpreter(model_path="sine_model.tflite")
```

Lab: Sinewave Validation Device. . .

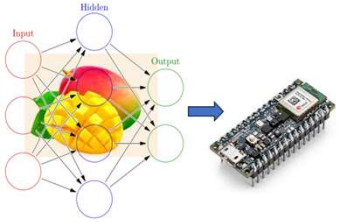


Part 4 Runs Prediction and Plot Results

Python Partial Code: Google Colab Environment

```
# ------------------------------
# Run predictions
# ------------------------------
y_pred_clean = run_tflite(clean_windows)
y_pred_noisy = run_tflite(noisy_windows)

# ------------------------------
# Plot (align lengths)
# ------------------------------
plt.figure(figsize=(12,6))

plt.subplot(2,1,1)
plt.plot(x_clean[window_size-1:], label="Original Clean")
plt.plot(y_pred_clean, '--', label="Predicted Clean")
plt.legend()
plt.title("Clean IMU Sine Validation")

plt.subplot(2,1,2)
plt.plot(x_noisy[window_size-1:], label="Original Noisy")
plt.plot(y_pred_noisy, '--', label="Predicted Noisy")
plt.legend()
plt.title("Noisy IMU Sine Validation")

plt.tight_layout()
plt.show()
```

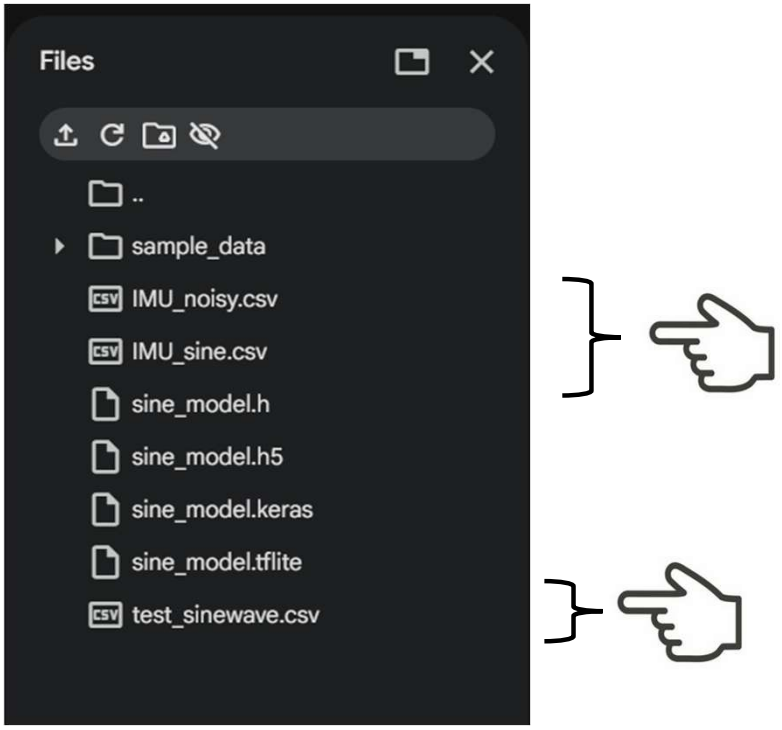
Lab: Sinewave Validation Device. . .



Part 4 Runs Prediction and Plot Results

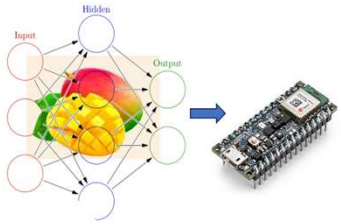
Files uploaded to and generated from Python TensorFlow Code

Python –Tensor Flow generated files



Uploaded files

Lab: Sinewave Validation Device. . .

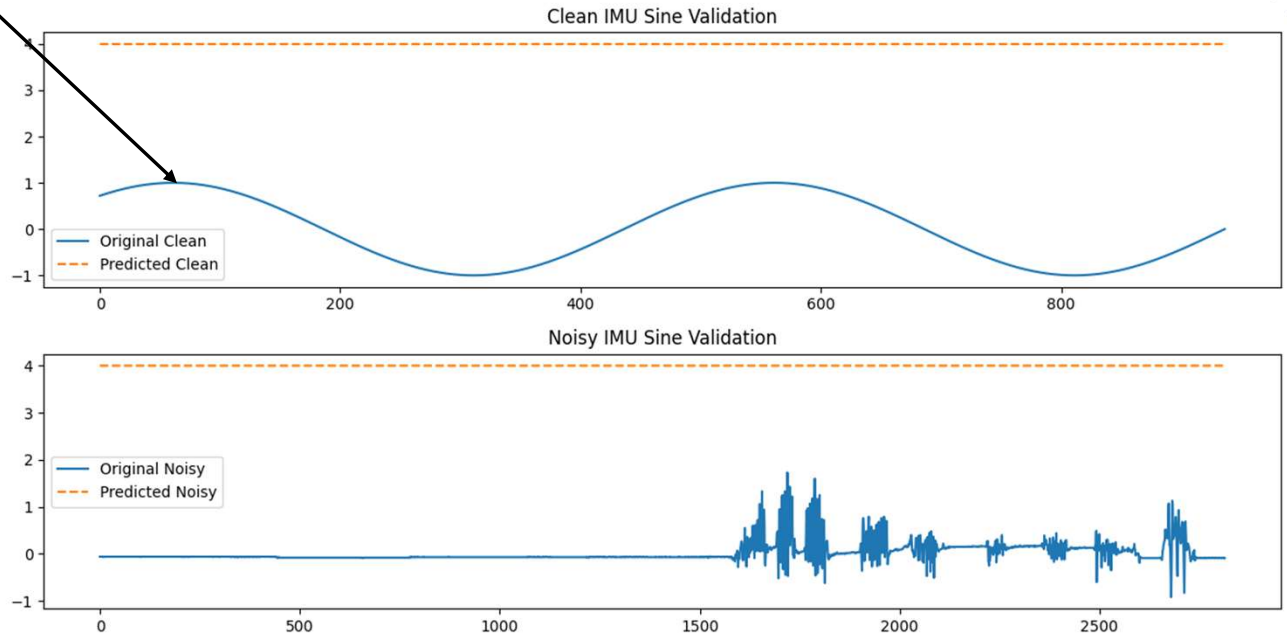


Part 4 Runs Prediction and Plot Results

Test sine wave

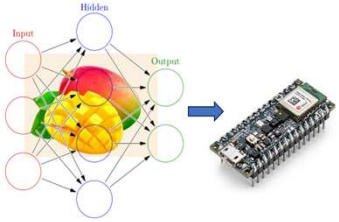
Clean Windows: probability ~1 model confident it's a sine wave

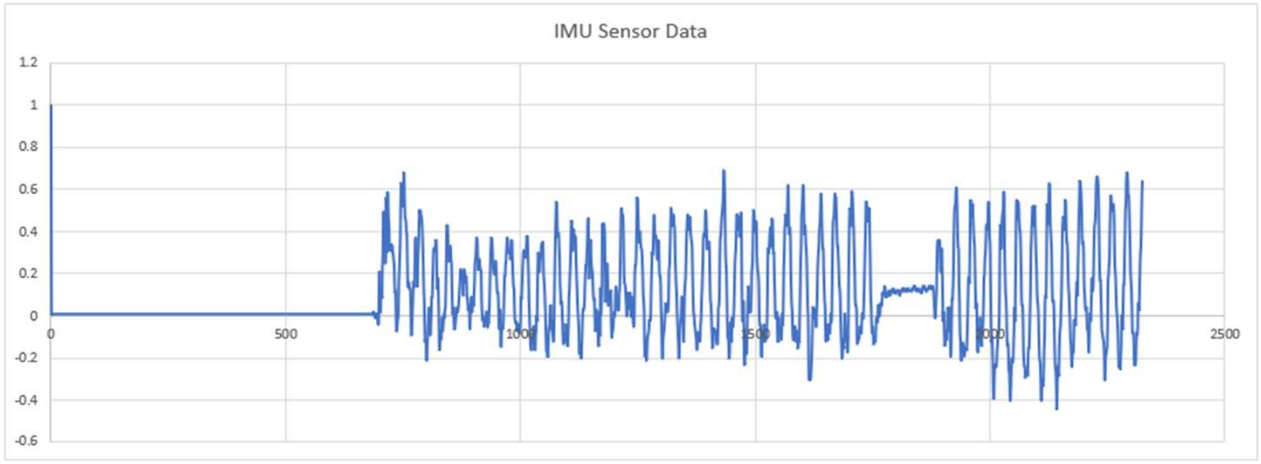
Noisy Windows: Probability <1 model detects deviations/noise

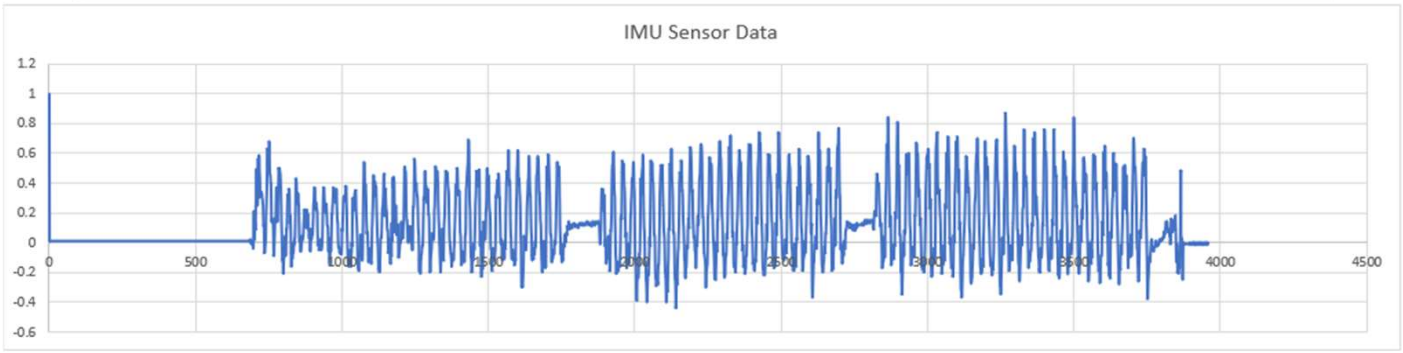


Lab: Sinewave Validation Device. . .

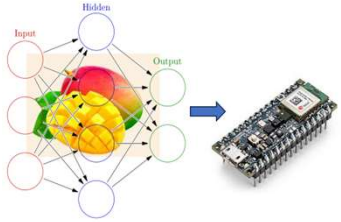
Arduino Nano 33 BLE Sense IMU Data





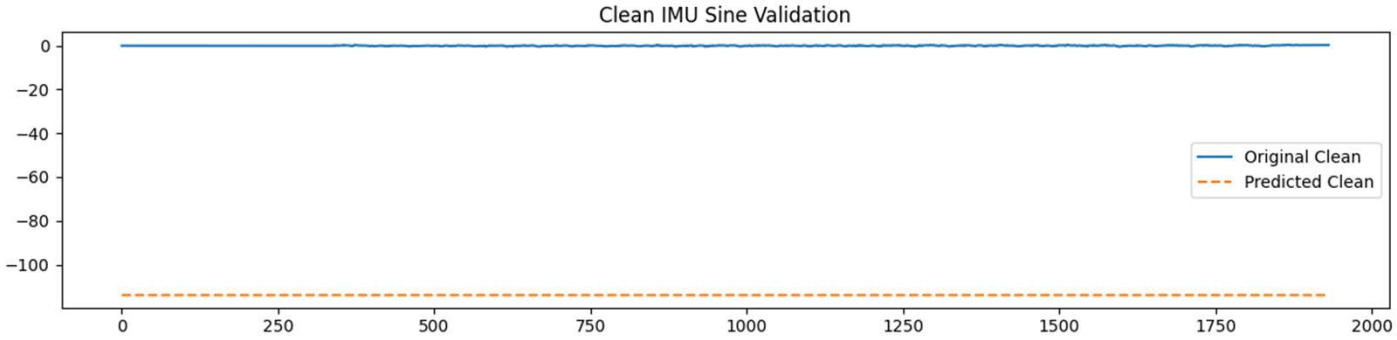


Lab: Sinewave Validation Device. . .

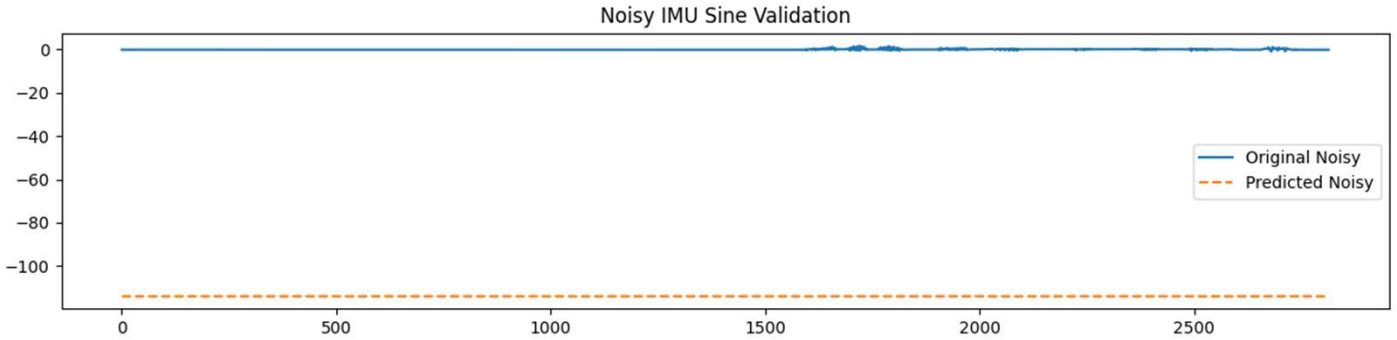


Part 4 Runs Prediction and Plot Results

Noisy Windows: Probability <1 model detects deviations/noise



Noisy Windows: Probability <1 model detects deviations/noise



Question 5

In reviewing slide 48, which file is not generated by the Python-TensorFlow code?

a) test_sinewave.csv
b) sine_model.h
c) sine_model.h5
d) none of the above

Thank you for attending

Please consider the resources below:

[1] J. Lin, L. Zhu, W. M. Chen, W. C.Wang, and S. Han, "Tiny machine learning: Progress and futures," *arXiv*:2403.19076v2 [cs.LG], Jun. 2016. [Online]. Available: https://arxiv.org/abs/2403.19076
[2] R. Mathur, "A detailed intro to neural networks," Aug. 2023. [Online]. Available: https://rikinmathur.substack.com/p/a-detailed-intro-to-neural-networks
[3] S. Heydari, Q. H. Mahmoud, "Tiny machine learning and on-device inference: A survey of applications, challenges, and future directions," May. 2025. [Online]. Available: https://www.mdpi.com/1424-8220/25/10/3191
[4] D. Wilcher, "Designs News December 25 webinar code," GitHub repository, Dec. 2025. [Online]. Available: https://github.com/DWilcher/DesignNews-WebinarCode/blob/main/December_25_Webinar_Code.zip

CEC Continuing Education Center

DesignNews

Thank You

Sponsored by

DigiKey

informa markets