Final Report

BRIDGE: Bridge Real-Time Intelligent Detection & Geospatial Evaluation

Giovanni Moncibaez, Joseph Lennon, John Nguyen, Nicholas Chlumsky

Department of Engineering, Florida Gulf Coast University

CEN 4935 - Senior Software Engineering Project II

Instructors: Dr. Fernando Gonzalez, Michael Osheroff

Mentor/Sponsor: Dr. Ali Ozdagli

Sensor Node **4**
How to set up Sensor Node for Programming 4
Download Arduino IDE 4
Download Libs 4
Connect ESP to Device 6
Success 6
Conclusion 6
Raspberry Pi **7**
How to set up Raspberry Pi 7
Setup 7
View on pc 7
NOTES: 8
ChirpStack LoRaWAN Network Application 8
NOTES: 9
Extra Considerations: 9
Node-Red 9
Backend **11**
AWS EC2 Setup & Installation 11
InfluxDB Installation & Setup 11
Tensor Flow **13**
How to set up Tensorflow 13
1. Environment 13
2. Install Dependencies 13
3. Upload Data 13
4. Load & Concatenate CSVs 13
5. Windowing (Data Augmentation) 14
6. Train/Validation Split 14
7. Visual Sanity Check 14
8. Build the Encoder 14
9. Build the Decoder 14
10. Assemble & Compile Autoencoder 14
11. Training 15
12. Quantize & Export to INT8 TFLite 15
13. Convert the .tflite into a C-array (autoencoder_model.h) 15
14. Include & Declare TFLite-ESP32 Globals 15
15. Loading & Initializing the TFLite Model 15
16. Quantize → Invoke → Dequantize 16
17. Putting It All Together in loop() 16

Sensor Node

Sensor Node Refers to the box that's going on the structure (Bridge), it is composed of a water resistant box, Solar Panel, Battery, ESP32, Accelerometer, GPS, LoRa, etc. The box layout or its contents might change during your time working with the project. Nevertheless, in this section you will learn how to set everything you need to start working on it.

How to set up Sensor Node for Programming

- Download Arduino IDE
- Download Libs
- Connect ESP to Device
- Success

Download Arduino IDE

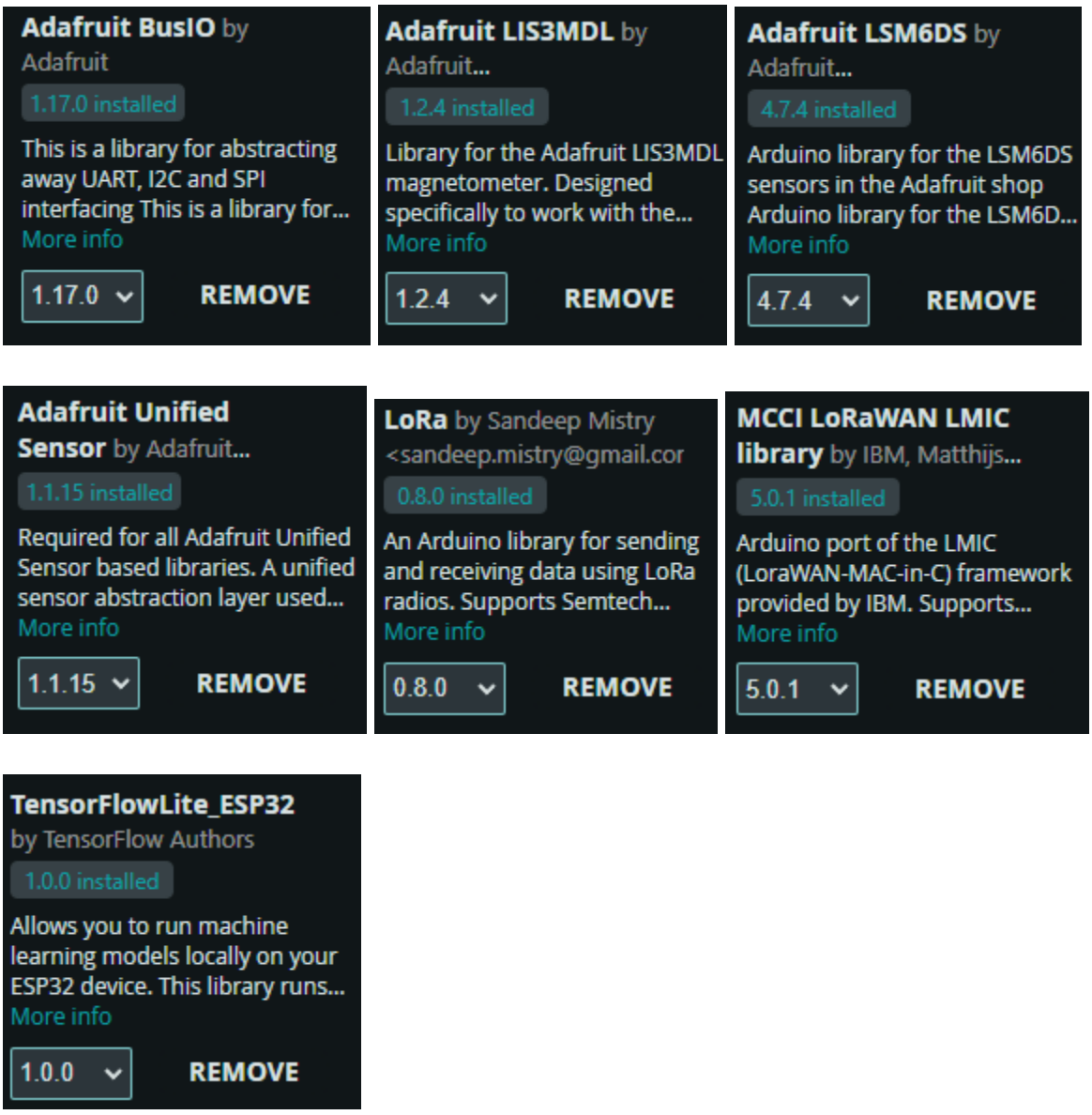
1. Go to https://www.arduino.cc/en/software/
2. Download the msi or zip (whichever you prefer)
3. Run

Download Libs

1. Get this ESP32 Board Manager (THIS VERSION OR WILL BREAK)
2. Download these Libs
3. If there are any problems then use the version we have in the photos
4. Note: Tensorflow lib is cooked and you might need to make adjustments, check github to see if we uploaded our edited version of the TensorFlow Library.

5. Here is the board manager

6. Here are the libs



Connect ESP to Device

1. Connect ESP with type C cable
2. Once connected open arduino ide
3. Select boards
4. Make sure it says adafruit esp32 v2 feather

Success

1. Now that you follow all of these steps, you should be able to..
    a. Compile our code (20 mins long)
        i. My github: https://github.com/GioMonci/Tflite-Accel-LoRa-Pi
        ii. ViperLab: https://github.com/VIPERFGCU/WISE-Framework
    b. Edit and improve our code (maybe rewrite it all)

Conclusion

Following all of these steps should get you up and running on the sensor node aspect of the project. If you encounter any problems, it's probably because the board manager or the libs are updated and they broke something in tensorflow lib. If that's the case you can revert to the versions in the photos above, or delve into the beautiful libraries and change code. (Lower ESP32 version of the board manager before you do anything if you are having trouble. Fixed it for me. WORKS ON MY MACHINE.

Raspberry Pi

The other major part of the project is the Raspberry Pi, it acts as the middle man between the sensor node and the backend. The Pi is physically installed with a RAK2245 Pi Hat which allows for multichannel LoRa transmissions.

How to set up Raspberry Pi

The Raspberry Pi runs on Chirpstack's custom OS made for their software. The Full version of the OS is installed for this iteration of the project.

Setup

If starting from scratch and there is no OS on the SD card (You will need a working PC):

1. Download either the base or full version of the custom ChirpStackOS (Full was used for this project).
2. Download BelenaEtcher.
3. Flash the OS on to the SD card
4. Install SD card onto the Raspberry Pi and Power

View on pc

The Raspberry Pi should have a WAP named ChirpStack-XXXXXX to access the Raspberry Pi

1. Join the ChirpStack WAP (Password: ChirpStackAP)
2. Go to 192.168.0.1

3. Login with root and no password
    a. On the first login, you are prompted to make a password
4. You then can go to the ChirpStack LoRaWAN Network application (should be 192.168.0.1:8080)

NOTES:

- ChirpStack Web application can back up your current configuration of ChirpStack. I recommend this when editing settings. Save/Backup Often.
- FGCU campus Wi-fi can be restrictive, so joining the wi-fi may cause issues and when it tries to connect to the internet via wifi, it may cause the Raspberry Pi to change its IP access point in which you won't be able to access the 192.168.0.1 IP.
    - If you're knowledgeable enough, you can edit the files inside the Raspberry Pi to change the wifi access point back, but we just reflashed the SD card.

ChirpStack LoRaWAN Network Application.

1. The default login is:
    - Username: admin
    - Password: admin
2. In this page, go to Device-Profile and create a device profile.
3. Then you go to the application tab and use the newly created device profile as the device profile.

4. Within the application tab, you can create a new device.
5. Generate the device keys for the sensor node using Over The Air Activation (OTAA)
    - Keep in mind Big Endian (MSB) and Little endian (LSB)
    - LMIC library requires the DevEUI and AppEUI to be in LSB but the AppKey can be in MSB. (realistically as long as it successfully connects to the Raspberry Pi gateway, it should be fine as long as you understand what is required)
6. In this device, you can view events and the LoRaWAN frames tab, which will keep track of sensor nodes using those keys.

For deeper understanding of ChirpStack, refer to the documentation provided.

https://www.chirpstack.io/docs/

NOTES:

- There are a lot of integrations that ChirpStack has, we used Node-Red for HTTPs but there is a direct option for HTTPs and InfluxDB integration.

Extra Considerations:

- Instead of installing the Full Version onto the Raspberry Pi, the base version will just have the LoRa component and you would be able to access the Network Server on something like an external host or an AWS instance.

Node-Red

Node-Red uses the MQTT protocol to be able to send data from one place to another. Node-Red was used as a means to send data from ChirpStack/Raspberry Pi to the backend (InfluxDB) via an

API. Node-Red, in the WISE iteration, was intended to be used as a means to keep track of which sensors were active and to ensure that sensors were active. Node-Red can be used for a wide application outside of these two applications. For deeper details, refer to the ChirpStack documentation: https://www.chirpstack.io/docs/guides/node-red-integration.html

1. In the flow tab, create an MQTT In node using
    - Server: localhost
    - Port: 1883
    - Topic: application/+/device/+/event/+
    - QoS: 0
    - Output: auto-detect.
2. Create a function node:
    - When the MQTT node receives data, it will take ChirpStack's whole payload. So the intent of this function node was to just replace the msg object as just the payload/data.
3. HTTP node:
    - Method: POST
    - Paste the API key to the endpoint URL.
4. (Optional) Debug Node
    - Debug nodes can be used to debug, you can place them at the end of any node in node red to see the output of the node.

Backend

The backend is all hosted on an Amazon EC2 instance. All active services include a custom Node.JS API to ingest data from the Raspberry Pi, InfluxDB to store the data, and Grafana to graph and visualize the data.

AWS EC2 Setup & Installation

- Set up an Amazon AWS account.
- Register for the free tier EC2 instance.
- Install Ubuntu 20.04 or newer (22.04, 23.04, 24.04, …)

InfluxDB Installation & Setup

- Download and Install InfluxDB by following the instructions at https://docs.influxdata.com/influxdb/v2/install
- Once downloaded, navigate to the Web UI created by InfluxDB (typically located at **{MachineIP}:8086**) to secure it. On the Web UI, you can create an organization (**ORG**) and a data bucket (**BUCKET**).
- After creating an org and bucket, you can navigate to **Load Data > API Tokens** and create an API token (**API_TOKEN**) which will be used for the backend API and Grafana visualizations.

Grafana Installation & Setup

- Download and Install Grafana by following the instructions at https://grafana.com/docs/grafana/latest/setup-grafana/installation/debian/

- Once downloaded, navigate to the Web UI to create an account and secure the Grafana dashboard.
- In Grafana, navigate to the data sources tab to add the InfluxDB bucket as a data source. (**Data Sources > Add New Data Source > InfluxDB**). Set name to anything i.e.: **“BRIDGE_INFLUX”**. Set **Query Language** to **InfluxQL**. Enable Basic Auth. Set HTTP URL to **“http://{MACHINE_IP}:8086/”.** Insert under InfluxDB details the **ORG** for Organization, **API_TOKEN** for token, and **BUCKET** for the bucket/default bucket. Then click **Save and Test** to add the data source.
- After adding the data source, you can create a new or import our existing Grafana dashboard which will be used to visualize new data.

Backend API Installation & Setup

- The backend Node.JS API can be set up by installing any service that can run a Node.JS application. One example would be using the PM2 package, and setting it up to run a local copy of the API on the machine. View https://pm2.io/. Within the Node application, you will configure the same InfluxDB parameters to send data to InfluxDB. (There are countless simple examples on different ways to set this up, so I will not go into great detail on how to do so).

Tensor Flow

TensorFlow Lite is leveraged in this project to deploy a compact 1D-convolutional autoencoder—trained offline in Python—to the ESP32. The autoencoder learns to denoise raw vibration waveforms (100 Hz accelerometer samples at 1 Hz, 5 cm amplitude) and to compute a reconstruction‑error–based anomaly metric (e.g. RMSE).

How to set up Tensorflow

1. Environment
    - **Google Colab** (recommended) or any local machine with Python 3.7+.
    - In Colab: *Runtime → Change runtime type → GPU* for faster training.
2. Install Dependencies
    - **NumPy/Pandas** for data handling
    - **TensorFlow/Keras** for model building & training
    - **Matplotlib** for visualization
3. Upload Data
    - In Colab's file browser, upload 100hz1.csv, 100hz2.csv, 100hz3.csv.
    - Or ensure they live in your working directory.
4. Load & Concatenate CSVs
    - Read three shake-table files
    - Stack them into one long DataFrame

- Extract the 3-axis accelerometer columns into a NumPy array of shape (N, 3)

5. Windowing (Data Augmentation)
    - Slide a 50-sample window across the data, moving by 10 samples each time
    - Results in overlapping windows to boost training data
6. Train/Validation Split
    - First 80% windows for training, last 20% for validation

7. Visual Sanity Check
    - Plot one raw window (50 samples) on all three axes
8. Build the Encoder
    - Conv1D layers act as learnable FIR filters over time
    - Strides reduce the temporal dimension, forcing compression
    - Final Dense layer maps the 13×8 tensor into a 16-dim “code”

9. Build the Decoder
    - Conv1DTranspose upsamples the temporal dimension
    - Cropping1D fixes off-by-one stride artifacts
    - Last Conv1D linearly maps back to 3 channels
10. Assemble & Compile Autoencoder
    - End-to-end model: window in → denoised window out
    - MSE loss trains it to reproduce its own input

11. Training

- Runs 40 epochs, tracks training and validation loss

12. Quantize & Export to INT8 TFLite

- Produces a fully-quantized model ready for deployment on ESP32

13. Convert the .tflite into a C-array (autoencoder_model.h)

- **From your shell** (on Linux/macOS or in WSL on Windows), run:

  xxd -i autoencoder_model_3D_INT8.tflite > autoencoder_model.h

14. Include & Declare TFLite-ESP32 Globals

- **micro_error_reporter / error_reporter**

  Captures and prints any TFLite errors.

- **model**

  Points to the compiled .tflite data that we linked in via autoencoder_model.h.

- **interpreter**

  Runs ops from that model inside the pre-allocated tensor_arena.

  tensor_in / tensor_out

  After allocation, these point to the single input and output tensor buffers.

15. Loading & Initializing the TFLite Model

- **GetModel(...)** wraps the binary .tflite into a Model object.

- **AllOpsResolver** pulls in every operator implementation the model might need (Conv1D, ReLU, etc.).
- **AllocateTensors()** carves out all the input, output, and scratch buffers from our 16 KB arena.
- Finally, **input(0)** and **output(0)** give you the addresses of the I/O buffers for inference.

16. Quantize → Invoke → Dequantize

- We read the FP32→INT8 scale and zero-point baked into the TFLite model.
- Each raw float is scaled, shifted, clamped, and cast into the int8 input buffer.
- Invoke() runs the network end-to-end.
- We reverse the process on the outputs, recovering a denoised float sequence.

17. Putting It All Together in loop()

- Fill raw_x[], raw_y[], raw_z[] at 100 Hz.
- Once you have 100 samples:
    - Zero the index
    - Denoise each axis with doInference()
    - Log raw vs. denoised first sample
    - Transmit the cleaned sample over LoRa