CARL VON OSSIETZKY UNIVERSITY OF OLDENBURG

MASTER'S THESIS

Benchmarking EdgeAI platforms using Arduino Nano 33 BLE

Autor:
Rohitashva S. JHALA

Supervisor:
Prof. Dr. Martin FRÄNZLE
M.Sc. Jan CORDES

Master's Thesis as a part to obtain Master's degree of Science

in collaboration with

DLR – Deutsches Zentrum für Luft- und Raumfahrt
Department of Computer Science

June 13, 2024

Declaration of Authorship

I, Rohitashva S. JHALA, hereby certify that I am doing this work independently and did not use any other sources and aids than those specified. I also certify that I understand the general principles of scientific work and publication as described in the guidelines for good scientific practice of the Carl von Ossietzky University of Oldenburg.

Signed:

Date:

CARL VON OSSIETZKY UNIVERSITY OF OLDENBURG

Abstract

Faculty II - Computer Science, Economics and Law
Department of Computer Science

Master of Science

Benchmarking EdgeAI platforms using Arduino Nano 33 BLE

by Rohitashva S. JHALA

An edge device is an endpoint of a network usually comprising some sensors attached to a microcontroller. These are the devices that collect/gather real-time data from their surroundings and send it further in the communication pipeline. Now to process this data locally rather than sending it to the cloud or a powerful machine for computing is where edge AI comes into play. In essence, running machine learning models locally on these resource-constrained devices by processing the incoming data is edge AI. In this work, we aim to explore the evaluation of various edge AI frameworks/platforms along the whole development process starting from the data preprocessing, all the way to ML model deployment.

Now, there are multiple platforms that claim to achieve this functionality. Some by providing a web-based application while others require an installation on a local computer. Some platforms train the ML models over the cloud while others on the local PC. Theoretically, ML models with the same configuration trained over the same data could give similar results but how that compares on a spectrum of frameworks is what we want to explore. The primary goal is to determine which platform or framework delivers the best balance between computational efficiency and practical utility.

To facilitate this, we benchmarked several edge AI frameworks across five critical metrics: power consumption, memory usage, inference time, model accuracy, and user-friendly interface. The benchmarks include assessments of power (during both idle and active states), memory requirements, the time taken for each platform to perform a standard inference task, and accuracy measurements for a predefined human motion-sense dataset. Additionally, usability was assessed based on the ease of integration, configuration, and modification provided by the platforms, considering a developer's perspective.

Acknowledgements

First of all, I would like to thank my supervisors Prof. Dr. Martin Fränzle and M.Sc. Jan Cordes. Thank you for showing me the way and supporting me with a lot of commitment during my work. I learned more during my work at DLR than ever before, I would like to thank the entire project group! I would also like to thank Dr. Domenik Helms for the many tips at the start of the Master's Thesis. The biggest thanks go to my parents. Thank you for making this study possible and for your motivating support during my entire course! Without your support, this work would not have been possible!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In an increasingly interconnected world, the ability to process information at the source, namely the device itself, has become a critical challenge in technology. The concept of Edge AI is to decentralize data processing from cloud-based infrastructures and high--performance computers to the very source, where the data is actually generated. This shift is not without its challenges, particularly when it comes to deploying Deep Learning (DL) algorithms like neural networks on devices with significant constraints in terms of computational resources, memory, and power [1]. Traditional architectures and methods, designed with the assumption of abundant resources, cannot function under these limitations. They often require either a compromise in the accuracy of the algorithms or a reliance on remote servers, introducing latency and privacy concerns [2].

The potential to overcome these hurdles lies in the optimization of neural network algorithms and adapting them to fit within the system requirements of microcontroller devices. This not only maintains the integrity and efficiency of data processing but also extends the scope of Artificial Intelligence (AI) applications to areas previously considered limited. From wearable health monitors, [3] to smart sensors in remote locations and farming [4], the implications for daily life are vast and varied.

This exploration began with a specific, practical problem: the development of a sensor module for railway tracks, capable of classifying vibrations triggered by various phenomena such as different kinds of trains, fallen trees, animals, and vehicle crossings. The objective was to enhance rail safety by providing immediate, on-site analysis of potential hazards without the need for constant, high-bandwidth connectivity to a centralized data centre. However, the application faced challenges due to the limited availability of datasets in the industry. Although the dataset used in this thesis is focused on the detection of human activities, the findings would serve as a strong reference for the project above to bring the rail industry forward. This realization initiated a deeper investigation into how neural networks can be effectively implemented on resource-constrained devices, setting the stage for the current work.

By starting from a practical application and extending into a general inquiry, this research seeks to address a fundamental question: How can we bring the power of neural

networks to the resource-constrained micro-controller devices, and what could this mean for our evolving understanding of the paradigms and possibilities within Edge AI and its effect in our daily lives? This question encapsulates the essence of the challenges, the potential for innovation, and the real-world impact that defines the field of Edge AI.

1.2 Structure of the thesis

The structure of this work is divided into the following five chapters. In the first chapter, the basic chapter, there is an introduction to the topics in this work and the required technologies. In the second chapter, the concept of this work with an overview of the proposed frameworks, metrics, and dataset is presented. In the third chapter, this concept is then applied to comparing frameworks. The fourth chapter deals with the evaluation of the resulting data of the frameworks over a set of metrics like accuracy, memory, current consumption (in Ampere), inference time, and usability among others, and discusses possible applications of the results. The last chapter gives a conclusion about the work, the results achieved, and how the research question of this work was solved.

1.3 Basics

This section includes the introduction and overview of the EdgeAI frameworks, the deep learning libraries, and their functionalities that are necessary to understand the current state of research and preliminary research done in this thesis.

1.3.1 Keras - A Deep Learning API

Keras is an open-source deep learning API (Application Programming Interface or API is an interface for two or more software programs to communicate with each other with set definitions and protocols) used to implement neural networks in Python. It provides a high-level, user-friendly interface while running on the TensorFlow backend responsible for low-level operations. It offers a simple architecture for building models by: defining the model, adding their layers and, configuring the learning process. Whereas TensorFlow executes the processes defined by Keras such as constructing computational graphs, implementation of algorithms and loss functions, training it over the data, evaluating errors and, predicting over the new data [5].

In this work, a Convolutional Neural Network (CNN) is implemented using Keras to identify patterns in time-series Inertial Measurement Unit (IMU) data. This network, trained on Google Colab's CPU runtime in Python programming language, serves as a baseline reference for comparison against various Edge AI frameworks, some of which utilize cloud-based training (like NanoEdge AI studio, Imagimob Studio, and Edge Impulse).

1.3.2 TensorFlow

Developed by Google, TensorFlow is an open-source software library designed for machine learning, deep learning purposes and other statistical predictive analytics workload [6]. It makes the implementation of machine learning models easier by providing a common interface for acquiring data, training models and fine-tuning the results. It does so by providing abstraction which is, grouping and wrapping pieces of code that represent higher-order functionality, so that developers can focus on the overall logic of the application. TensorFlow offers high-level abstractions for ease of use, while also providing powerful low-level optimizations that are difficult to achieve manually, allowing for efficient execution by taking advantage of specific hardware capabilities, such as GPUs (graphics processing units) and TPUs (tensor processing units) [7]. Although TensorFlow has a Python front-end, it is built on C++ and Java giving quick training and deployment times [6]. Supporting a wide range of programming languages like Python, Java, C, C#, Go, and Swift makes using the framework very straightforward. The name TensorFlow is derived from two words, Tensor and Flow. Tensors are n-dimensional vectors or matrices that represent all forms of data and Flow is the series of calculations/procedures performed to generate the output. Hence, the name TensorFlow.

1.3.3 Scikit-learn

Scikit, originated from the name SciPy Toolkit (a Free open-source Python library for Scientific Computing) is a free machine learning software library for Python programming language [8]. Offering a wide selection of algorithms for classification, regression, clustering, support-vector machines, k-means, random forests and many more, makes it easier for developers and data scientists to build machine learning models without getting deeply involved in the algorithmic complexities. By importing the library into a Python script, one can utilize all the data processing methods and algorithms defined in it. Scikit-learn algorithms and data processing methods are implemented in Python, utilizing Cython for performance-critical code segments to ensure efficiency [8]. Behind the scenes, it heavily relies on NumPy for high-performance matrix operations and array handling and integrates with SciPy for additional scientific computing capabilities. In addition, It integrates well with other Python libraries such as NumPy and SciPy for mathematical operations and Matplotlib for plotting, providing a smooth integrated workflow. The development of scikit-learn is community-driven and it boasts extensive documentation with community support.

1.3.4 Google Colab

Google Colab is a cloud-hosted service from Google providing Jupyter Notebook system services [9]. There are multiple versions of Colab, starting from a free version to paid individual or enterprise versions. The free version provides access to a handful of GPUs

and TPUs, along with 12 GB of RAM and 68 GB of disk space. Colab, short for Colaboratory, allows users to write and execute Python code without any configuration or setup. Google uses one account for all its products, and if you have an existing Google account, you can use the same credentials for logging into Colab without registration. Conversely, you can register with Google to access the Colab service. The Colab notebooks created are saved to your Google Drive account. Google Colab is built on top of Jupyter Notebook, an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text [10]. It integrates seamlessly with Google Drive, enabling collaborative work and easy sharing of projects. The infrastructure is powered by Google's cloud services, providing scalable and robust computing resources. The primary purpose of Google Colab is to provide an accessible and efficient platform for data analysis, machine learning, and research. It democratizes access to powerful computational resources, which can be particularly beneficial for students, educators, and researchers who may not have access to high-end hardware. Colab also supports popular machine learning libraries such as TensorFlow, Keras, PyTorch, and OpenCV, making it a versatile tool for various data science and AI tasks. Google Colab offers several key features [11]:

- Interactive Environment: Users can run code cells interactively, making it easy to test and debug code.
- Collaborative Tools: Multiple users can work on the same notebook simultaneously, similar to Google Docs, with integrated comments and version control.
- Pre-installed Libraries: A wide range of Python libraries are pre-installed, reducing the setup time for new projects.
- GPU and TPU Support: Access to GPUs and TPUs accelerates the training of machine learning models, significantly reducing the time required for computation-intensive tasks.
- Free Public Data Collection: It provides access to a public data collection library that hosts a wide range of datasets for computer science disciplines [12].
- Example Notebooks: Colab features examples and open-source projects to help users get started and become familiar with its architecture.

Google Colab is a tool for anyone involved in data science and machine learning. It removes the barriers of limited local computational resources and offers a collaborative, cloud-based environment that is both user-friendly and powerful [13].

1.4 Related Work

Osman et al. [14] conducted a comparative study of two popular frameworks: TensorFlow Lite Micro (TFLM) and STMicroelectronics' CUBE AI. The latter, also known as STM32Cube.AI, is a toolchain that allows the conversion of pre-trained neural network models into optimized C code that can run on STM32 microcontrollers. This capability is particularly advantageous for deploying AI on devices where memory and power are limited. Their benchmarking focused on different TinyML applications to determine the most suitable use cases for each framework. They concluded that CUBE AI is better suited for memory-limited and power-intensive TinyML applications due to its efficient model optimization and integration capabilities. In contrast, TFLM offers wide availability and extensive support for a variety of devices. Osman et al. highlighted the need for further comparisons and benchmarking of various other frameworks to better assess their suitability with respect to specific applications, as mentioned in their future work section.

Similarly, The article by Pratim Ray [15] examines the field of TinyML, outlining its current landscape and critical components. Ray discusses the existing toolset that includes a variety of development boards and software frameworks that are important in deploying machine learning algorithms on microcontrollers. While Osman's paper focused on comparing two frameworks, this article by Ray aims to highlight the potential and limitations of TinyML. The paper highlights key requirements in the TinyML paradigm, such as advances in ultra-low power computing and the availability of high-efficiency neural network models for constrained devices. It provides a broad overview of state-of-the-art frameworks like TensorFlow Lite Micro, CUBE AI, and MicroTVM (Tensor Virtual Machines), which facilitate the implementation of ML models on tiny devices. Moreover, the article elaborates on various use cases ranging from predictive maintenance and asset tracking to voice activation and anomaly detection, all within IoT ecosystems. Despite the progress, Ray identifies significant challenges such as the lack of standardized benchmarks, the need for model optimization, and the difficulties in handling diverse data types and sources. He calls for a collaborative effort in the scientific community to develop new benchmarks with novel datasets, which would help in testing these frameworks more rigorously and understanding their application-specific performances better. This call to action underscores the necessity for ongoing research and innovation in the TinyML field to overcome existing hurdles and unlock its full potential.

Chapter 2

Concept

2.1 Overview

This section outlines the foundational concepts and methodology underpinning the research. A framework, in the context of Edge AI, refers to a structured platform or environment that supports the development, training, deployment, and in some cases optimization of artificial intelligence models on edge devices. These devices, such as microcontrollers, have limited computational resources compared to traditional cloud servers. Edge AI frameworks provide essential tools, and interfaces that facilitate the integration of AI capabilities directly onto these devices, enabling real-time data processing and decision-making close to the source of data generation. The process begins with data processing where the sliding window technique, explained in the section 2.5.1, is employed to transform the raw IMU data into a series of windows. Each window represents a single sample on which the neural network will be trained. By segmenting data into overlapping windows, we can capture the temporal dependencies which are crucial in sequential data. Prior to creating sliding windows, each file containing the data is first divided into test and train sets and then windows are generated as illustrated in the figure 2.1. This procedure is followed for every CSV file to ensure the integrity of the data and their respective labels. This methodology also ensures that there is no overlapping of the data between test and training sets and hence the windowed data generated is independent for both training and testing sets. Once the windows are generated, the training data is then input into a neural network which learns the features of the signal and then validates it over the test set. Metrics of interest like F1 Score, Accuracy, and Memory are measured before the deployment and once deployed on the target device, the same metrics are again measured to see any difference in the values of the metrics. Power consumption of the device when Idle versus when running the model is also gauged along with Inference time. By following this workflow for every framework, a comparison between the efficiency and usability of the frameworks can be drawn and the possible applications or use-case scenarios can be discussed.

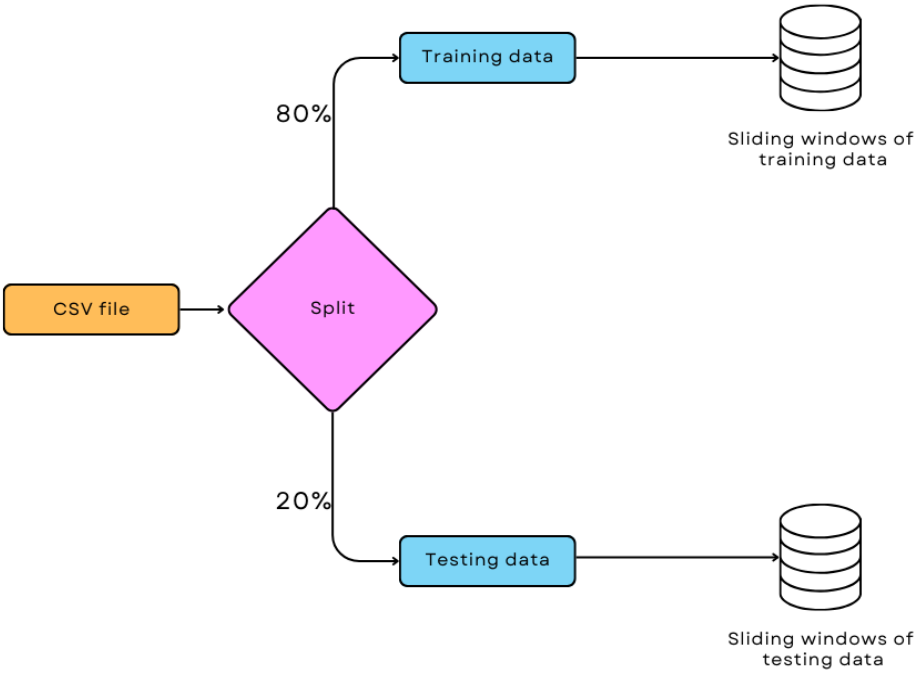

FIGURE 2.1: Generating sliding windows for test and train data

2.2 Frameworks

2.2.1 Edge Impulse

Edge Impulse is a browser-based development platform for building, deploying, and scaling embedded machine learning applications. It provides an interface to design machine learning models on real-world data and deploy them onto various target hardware devices. It has a cloud-based infrastructure where the user can upload/collect data, label it, analyse it, and then design the models for specific use cases. The CSV (comma-separated values) Wizard allows the user to configure the format in which the dataset has to be read when uploading which is particularly useful for large datasets to define their labels, analyse the data, configure timestamps (in the case of time-series data) and split the samples into test and training sets. It also supports a wide range of other type of files mentioned here [16]. Next is the Impulse Design (Figure 2.2), a process that creates customized models comprising input, processing, and learning blocks to address specific project requirements such as movement classification or object detection. The Input block defines the type and characteristics of data your model will use, like time series data or images and uses UMAP (Uniform Manifold Approximation and Projection), a dimensionality reduction algorithm to visualize high dimensional data into three-dimensional space. Then, Processing blocks are where raw data is transformed into features suitable for machine learning, often through digital signal processing techniques. Finally, the Learning block involves neural networks that are trained on these features to perform tasks like classification or object detection. To train, Edge Impulse uses different frameworks for different machine learning models like, for neural networks, it typically uses TensorFlow and Keras, for object detection models it uses TensorFlow with Google's Object Detection API, and for classic non-neural machine learning algorithms mainly uses sklearn. The users can also upload and deploy their pre-trained machine learning models (in TensorFlow SavedModel, ONNX, or TensorFlow Lite format) directly into Edge Impulse projects where they can quantize, configure, and process them for final deployment. On the other side, the EON (Embedded Online Neural) Tuner, one of the features of edge impulse, can automatically find the best model by analysing the input data and trying a series of processing blocks and neural networks under the constraints of the hardware device, out of which the user can select the one with the best possible fit. When deploying the model, Edge Impulse offers model optimization options like Quantized (int8) and Edge Optimized Neural compilers to reduce both flash and RAM (Random Access Memory) utilization while saving the inference times. It offers a range of deployment support: Deploying as a customizable library, as a pre-built firmware, as a docker container, as a custom deployment block (a paid feature) or on your phone/computer or Linux systems. The use of an SDK (Software Development Kit) and APIs (Application Programming Interfaces) abstracts the complexities and translates the model's requirements into commands that can be easily understood by different hardware architectures, thus offering a wide range

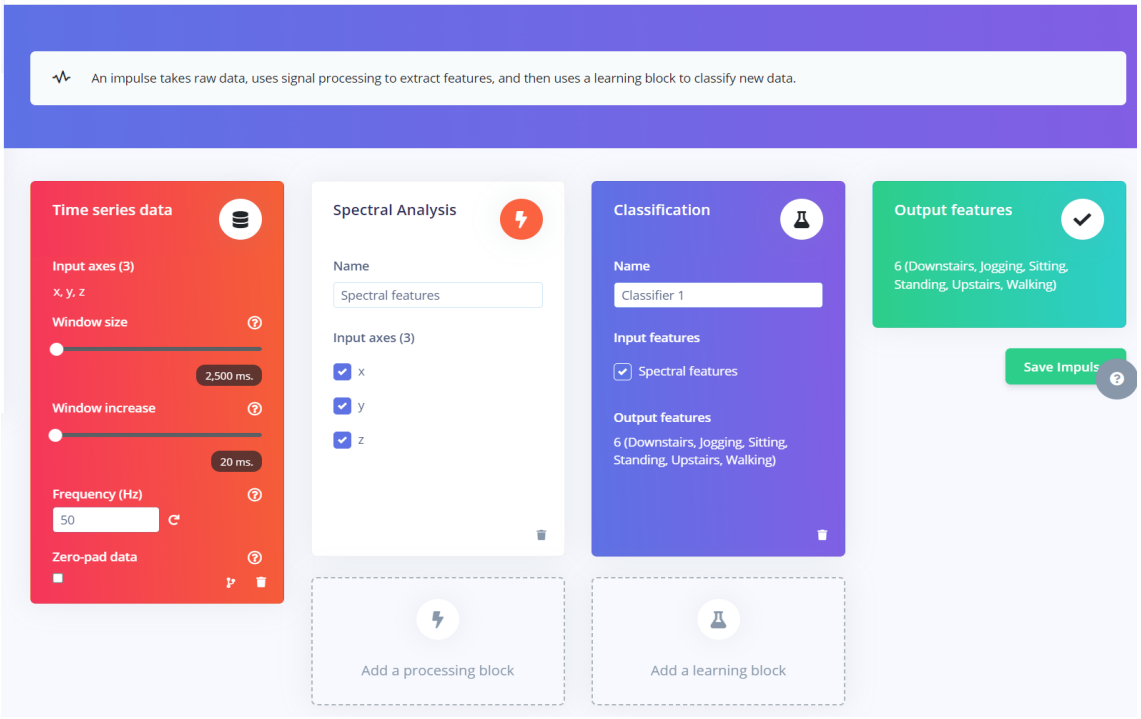

FIGURE 2.2: Impulse Design

of support for microcontroller units (MCU), central processing units (CPU), graphical processing units (GPU), and some AI accelerators & industrial devices.

2.2.2 TensorFlow Lite for Microcontrollers

TensorFlow Lite (TFLite) Micro is an open-source library designed to run machine learning models on microcontrollers with limited memory and processing power. It is an extension of TensorFlow Lite which is a mobile library that converts the TensorFlow models into efficient smaller-size models for deployment on resource-constraint devices as shown in the figure 2.3 [17]. TFLite micro is written in C++17 and requires a 32-bit platform. It supports Arm Cortex-M series and ESP32 architecture and can be embedded inside any C++ 17 workflow. Within the list of supported devices, the library currently supports limited operations [18] and cannot be trained on edge devices. For deployment on Arduino boards, the trained TensorFlow network first has to be converted into a .tflite model with a TensorFlow Lite converter which not only reduces the size but also enables the use of TensorFlow Lite operations defined in the operations resolver header file [18] and then using Arduino IDE can be deployed to the boards.

With an existing extensive community of TensorFlow, the TFLite for Microcontrollers Experiments collection [19] boasts a compilation of work done by developers to combine Arduino and TensorFlow for open-source free projects.

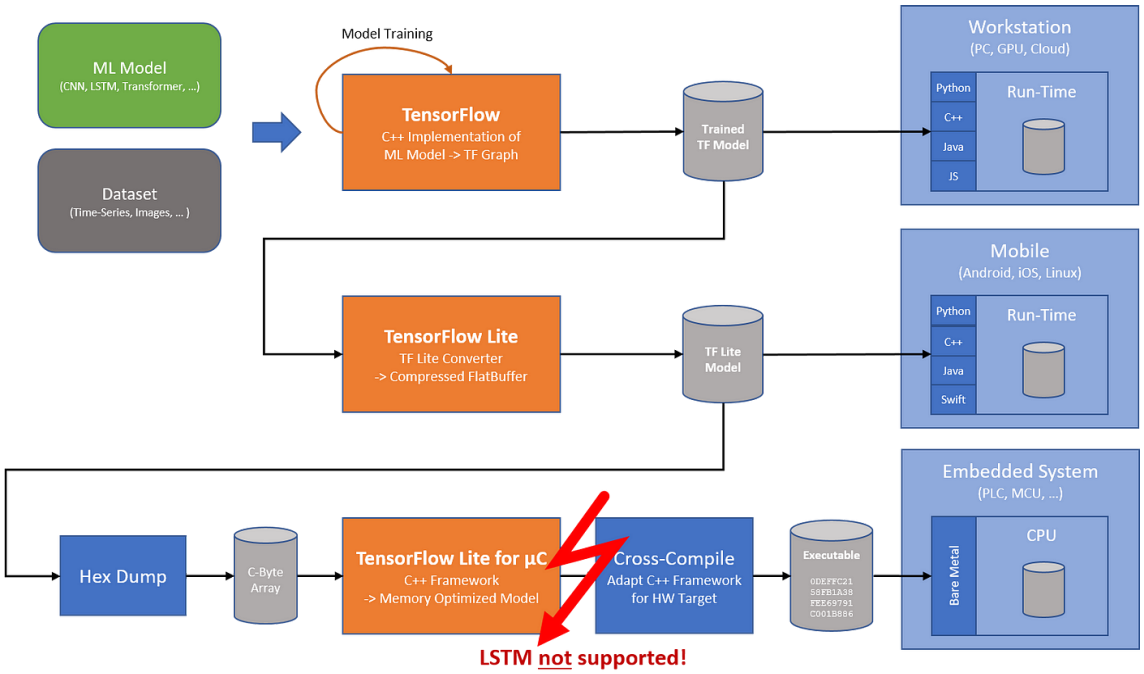

FIGURE 2.3: TensorFlow Lite workflow

2.2.3 MathWorks

MathWorks is a leading commercial developer of mathematical computing software Matlab and Simulink [20]. Matlab is a programming platform designed to analyze data, develop algorithms and create models & applications. It offers a wide range of applications like deep learning and machine learning, signal processing and communications, image and video processing, control systems, test and measurement, computational finance, and computational biology [21]. Simulink is an environment with a model-based design, where a system is designed using multiple blocks (from Simulink Library), each representing a task or a set of tasks. Then followed by simulations it can be deployed on the selected hardware. Because of the model-based architecture, the user does not necessarily need programming skills to design, simulate, and deploy the models. In the case of Edge AI, with Matlab and Simulink the user can:

- Generate optimized C/C++ code from trained models
- Include pre trained TFLite models, and
- Use hyperparameter tuning, quantization, and network pruning for AI models for inference on edge devices.

Matlab coder on the other hand is an extension of the functionalities of Matlab by converting the algorithms developed in Matlab into executable C/C++ for hardware. Similarly, the Simulink coder also converts the Simulink model blocks into executable C/C++ code

for hardware implementation and optimization [22]. In terms of hardware support, It enables working with a variety of Microcontrollers (MCUs), Digital Signal Processors (DSPs), Field-Programmable Gate Arrays (FPGAs), System on Chips (SOCs), Application-Specific Integrated Circuits (ASICs), Embedded Linux Systems and Real-Time Operating Systems (RTOS) [21].

2.2.4 Edge-ML

Edge-ML is another open-source and browser-based toolchain for machine learning on microcontrollers. It currently supports two boards: Nicla Sense ME and Open Earable v3 for deployment of the machine learning models directly from the framework. Currently, The framework is still in the ongoing development phase and thus supports the collection, management, and labelling of the dataset as fully available features. Meanwhile, the Training and Validation features are in their Beta version and the Deployment feature is in the Alpha version offering limited capabilities. It supports data collection from CSV, Node.js, Arduino boards, Android (Java) and Python.

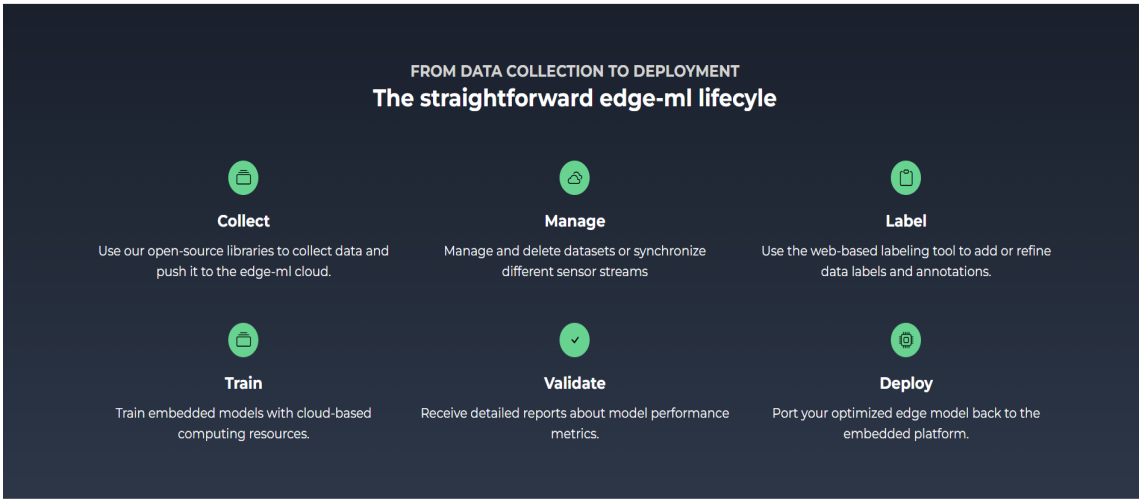

FIGURE 2.4: EdgeML workflow

Currently, It only supports boards over BLE (Bluetooth Low Energy) so the hardware has to be equipped with a BLE sensor to establish communication. In the case of Arduino boards, It supports data upload from Arduino Nano 33BLE Sense and Nicla Sense ME by providing an EdgeML-Arduino library. This library can be downloaded from the library manager in Arduino IDE and thereafter can be used to connect the Arduino boards to the framework via Web Bluetooth. Additionally, It also supports other ESP32 SoC microcontroller that comes equipped with BLE and Wifi out of the box. In terms of functionalities, Edge-ML offers a series of functions [23] that can be used with the Arduino IDE for better integration. Users also have an option to connect other boards and sensors to Edge-ML by utilizing the specific functions/classes mentioned [23] to implement Custom Board Configuration and upload the data. Another Alpha feature is Edge-ML Auto, which

performs a neural architecture search for the best possible neural network fit to the dataset based on the feature extracted. In terms of model deployment, it outputs Python and C files which then can be integrated within the hardware for deploying the functionalities.

2.2.5 NanoEdge AI Studio

NanoEdge AI Studio is a free software developed by the global semiconductor company STMicroelectronics for Windows and Linux platforms. The Studio takes the input data and generates a static library (.a) file as an output containing the model [24]. This library then must be linked to some C code programmed and compiled by the user to leverage its functionalities on the edge device. Once compiled on the hardware, it gives the ability to bring Machine Learning capabilities in the form of easily implementable functions for learning signal patterns, detecting anomalies, classifying signals, or extrapolating data. The NanoEdge AI Library developed by Cartesiam is a precompiled file that provides the building blocks to implement the AI capabilities into a C code. Based on the provided input parameters, the studio generates an optimized NanoEdge AI Library tailored to your application. This library can be pre-trained within the studio or deployed in an untrained state, where it begins learning from data once embedded in the microcontroller. This Library is comprised of three main blocks (Figure 2.5): A signal pre-processing algorithm to extract features from the input data, a machine learning model, and hyperparametrization for the ML model.

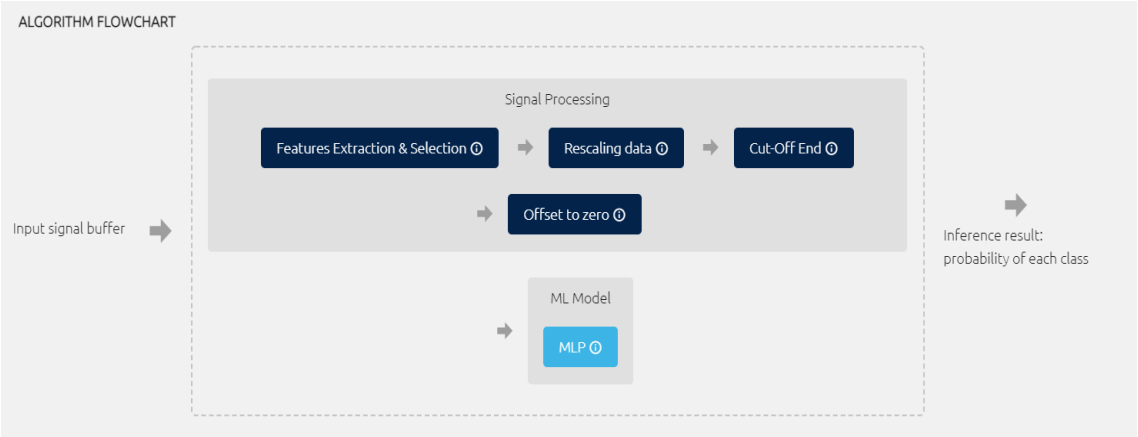

FIGURE 2.5: NanoEdge AI Studio - Library Workflow

Some tools of the studio also include a Data Logger generator, Data Manipulation and, Sampling Finder. Data Logger is used to create a configuration code that easily collects data from the development boards. Data Manipulation helps to refine the shape of the dataset easily. Sampling Finder is used to estimate the sampling rate or signal length of the dataset to be used in a project. In terms of hardware support, the studio allows compilation on a variety of STM32 development boards (140+), microcontrollers (550+) and Arduino boards (18+). The input signals can be imported using three sources: as a .txt

or .csv format, from a live datalogger via a serial port USB, or from a datalog (.dat) file. The studio supports input signals like Anomaly Detection, 1-class Classification, n-class Classification, and Extrapolation. In the Benchmark process, the Studio automatically searches for the best possible NanoEdge AI library according to the configuration of the hardware and the input signals. The Studio automatically:

- divides all the imported signals into random subsets (same data, cut in different ways),
- uses these smaller random datasets to train, cross-validate, and test one single candidate library many times,
- takes the worst results obtained from step #2 to rank this candidate library, then moves on to the next one,
- repeats the whole process until convergence (when no better candidate library can be found).

As for the deployment, the compiled library containing the AI model can be downloaded along with an optional "Hello World" code (as an example for guidance) when programming the corresponding C code to be flashed onto the hardware board.

2.2.6 Imagimob Studio

Imagimob Studio is a free development platform for Edge AI applications providing full access to the development, evaluation and testing of Edge AI models. The training of the models is done over the cloud with a usage limit of 3000 compute minutes/month of AI training. However, production and commercial development requires a fee on request. The studio uses a graph-based interface as shown in figure 2.6 [25] called Graph UX (User Experience) to visualize end-to-end machine learning workflow from building to the evaluation of the models as graphs. It is designed to provide clear understanding by representing workflow as basic building blocks and ease of use by dragging and dropping machine learning components on canvas. For importing the data, the data must be prepared in the format supported by Imagimob Studio, while supporting audio (.wav), video (.mp4), and time series (.csv, .data) data. Real-time data can also be collected using Graph UX and IMAGIMOB capture. Graph UX is used to capture real-time acoustic data using serial capture (Figure 2.7) or a local microphone and requires flashing the Imagimob data streaming firmware on the development board [26] while IMAGIMOB capture captures the real-time data from any sensor/development board over a serial connection [27].

Another feature of the Studio is ML-assisted labelling. It is a process in which the user initially labels the subset of the dataset manually and then trains the machine learning model over it, after which the model can be used to train the remaining dataset. For

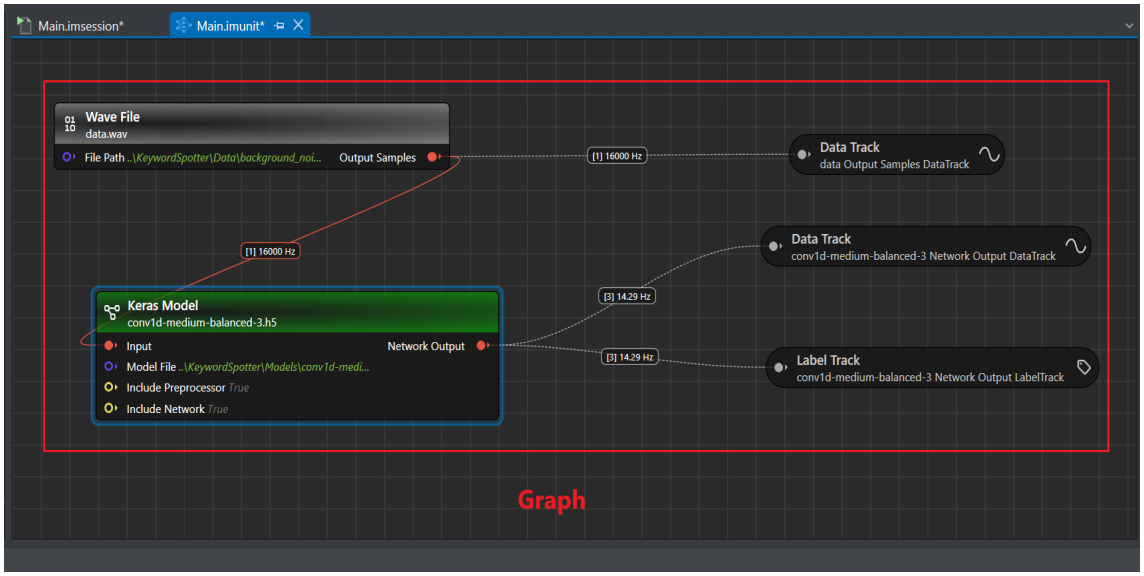

FIGURE 2.6: Imagimob Studio - Graph UX

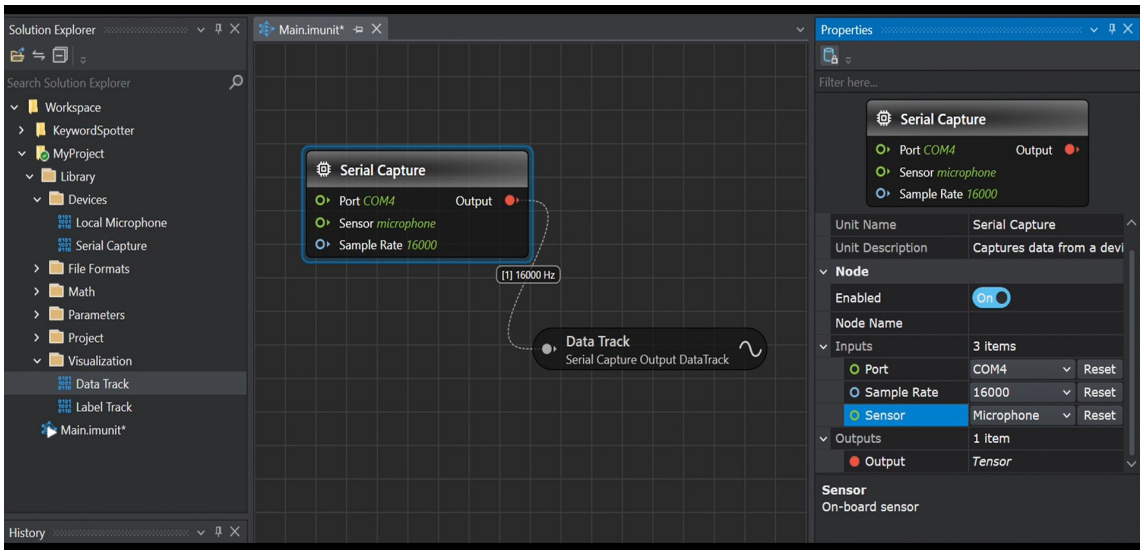

FIGURE 2.7: Capturing real-time microphone data using serial capture

training a model on the whole dataset, Imagimob Studio uses an auto machine learning approach for configuring a neural network architecture and sending it to the Imagimob cloud for training. Once trained, the user can use the code generation feature to convert the model into optimized C-code [28] and Edge API for interacting with the model for deployment on the edge device. Within the Edge API, the model is first initialized. Subsequently, input data is pushed using a predefined function within the source code [29]. Upon feeding the data into the model, a designated predefined output function receives the resulting values. The specific data types and formats for input and output depend upon the neural network architecture, necessitating careful handling in accordance with the model's expectations. Following output generation, it can undergo further post-processing for the intended application.

2.2.7 uTensor

MicroTensor or uTensor is an open-source framework for running machine learning models on resource-constrained edge devices. It is extremely light-weight and is built on TensorFlow and optimized for Arm architecture platforms. It converts a pre-trained TensorFlow model into readable and easy-to-customize C++ source files. The framework consists of a runtime library and an offline tool which converts the models and produces a .cpp and .hpp file as shown in the workflow figure 2.8 [30]. The uTensor runtime is

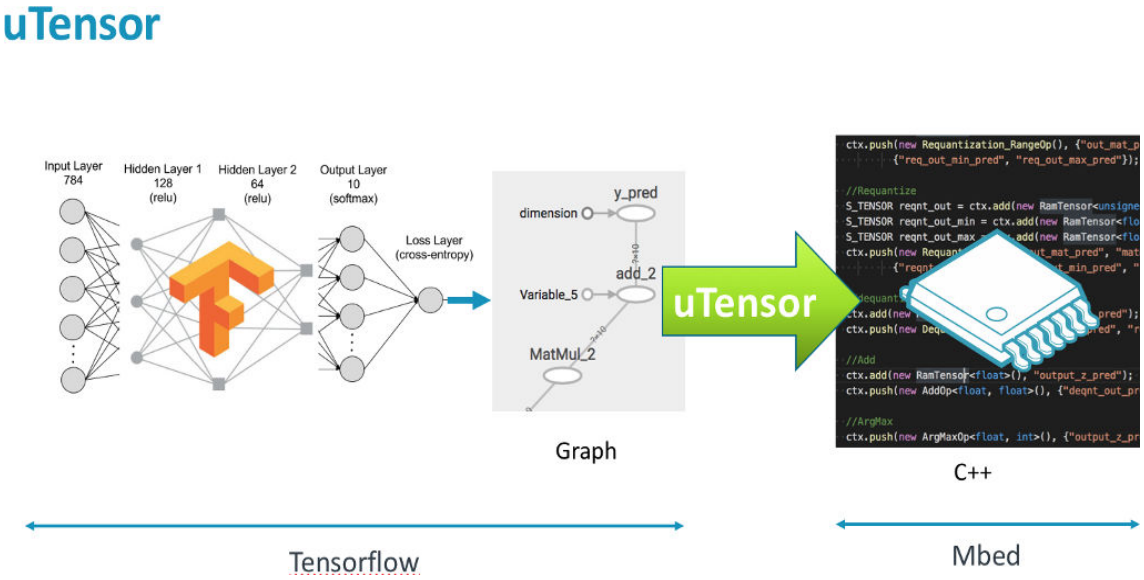

FIGURE 2.8: uTensor Workflow

comprised of two other components, uTensor Core and uTensor Library. The uTensor Core manages the execution of operations, memory management, and task scheduling and consists of data structures and interfaces required during runtime. The uTensor Library consists of default Implementations of error handlers, operators, memory managers, and tensors which can be modified and customized by the user as per applications [31].

The re-architecture of uTensor now enhances the system safety, clarity, debuggability, and extensibility of the framework. Dedicated memory regions for tensor data and metadata, ensure predictable memory usage enabling safe remote model updates. Mismatched inputs/outputs, incorrect sizes, and impossible memory access errors are now checked before runtime. This enhances the overall developer experience by making the program easier to update and modify over time while adapting to specific user needs and preferences [30]. The step-by-step the installation procedure of the uTensor code generation library with its overall backend architecture information can be found here [32].

2.2.8 Arm CMSIS-NN

The Arm CMSIS-NN library is a collection of open-source efficient neural network kernels developed for optimum performance of neural network operations with the least memory trace on Arm Cortex-M processors. It was developed by Arm, the company known for creating the Arm architecture used in embedded systems and mobile devices. Arm designed the CMSIS-NN as part of the Cortex Microcontroller Software Interface Standard (CMSIS) suite to specifically enhance and optimize neural network performance on Cortex-M processors [33]. These optimizations are tailored to take full advantage of the hardware features of Cortex-M processors, such as SIMD (Single Instruction Multiple Data) instructions and efficient use of the limited memory bandwidth. CMSIS-NN also supports int8 and int16 quantization of machine learning models from the TensorFlow Lite Micro library. The CMSIS-NN is comprised of core neural network functions that perform basic neural network layer operations like convolutions, softmax, pooling, and activation and helper functions which assist in data reformatting, parameter checking, and other utilities that facilitate the implementation and execution of neural networks [34]. It utilizes fixed-point arithmetic instead of floating-point, which is more resource-efficient for MCUs that lack a floating-point unit (FPU). CMSIS-NN utilizes CMake for building the library. During the build process, specifying the target CPU is mandatory. Additionally, users have the option to customize the build by selecting from a range of optimization levels and adjusting compiler settings, which can significantly influence performance [33].

2.2.9 ELL Microsoft

Microsoft's Embedded Learning Library (ELL) is an open-source library designed by Microsoft for developing and building intelligent devices and AI powered gadgets. Like CMSIS-NN, it targets resource-constrained devices but offers a broader scope of use across various hardware architectures like small single-board computers, such as Raspberry Pi, Arduino, and micro:bit [35]. This library is an ongoing development work and could include some changes in the API in the future. The library and its tools are written in C++ with an optional interface in Python. ELL acts like a cross-compiler where the compiler itself runs on a personal computer in which it is installed while generating the

machine code for the microcontrollers. The library also provides a range of tutorials for image classification and audio classification in Python and C++ languages. Along with a setup guide, most of the tutorials provided are focused on building and deploying models on Raspberry Pi board [36]. Getting started with ELL requires cloning the ELL's git repository. Additionally, prerequisites for building ELL require a C++ compiler, a CMake build system with version 3.15 or newer and some packages like LLVM, SWIG (Simplified Wrapper and Interface Generator), OpenBLAS, and Doxygen. ELL can also be used in Python 3.6 version or newer. A step-by-step procedure to download all the dependencies and related packages (for Windows system) to run the library can be found here [37]. ELL supports models trained with popular frameworks like TensorFlow, PyTorch, and CNTK, making it versatile and adaptable to various development needs. ELL library includes compiler and optimizer tools that convert and optimize machine learning models for specific target architectures. Model Wrappers facilitate model integration into existing applications by wrapping the optimized models with additional code necessary for deployment. Using the advanced installation, the user can also override the processor architecture and can manually build a specific version of OpenBLAS architecture (Basic Linear Algebra Subprograms library) as per requirements [38].

2.3 Metrics

In this section, we evaluate various Edge AI frameworks by assessing their performance across five critical metrics: power consumption, inference time, model accuracy, memory consumption, and usability. These metrics are crucial for determining the feasibility and efficiency of deploying AI models on low-power devices like the Arduino Nano 33 BLE. Power consumption is a vital metric for edge devices, particularly those operating in power-constrained environments. To measure the power consumption of each framework once deployed on the Arduino Nano 33 BLE, we use a digital multimeter to record the current observed when the model is making predictions (active) and when the device is idle. This is done by connecting the multimeter in series between the 3-volt power supply and the Arduino board. The positive polarity of the source passes through the multimeter and connects to the 3.3 volts input pin of the board while the negative polarity of the source is connected to the GND (ground) pin of the board. This is shown in the figure 2.9. Inference time refers to the duration the model takes to output a prediction after receiving input. This metric is critical for applications requiring real-time processing. To measure inference time, we time the execution period from the moment an input is provided to the model until an output is received. This is done by using the Profiler library from the library manager of Arduino IDE. The objects defined within the library measure the time taken by the functions or any part of the code enclosed within the curly braces where the object is called [39]. Accuracy measures the model's performance in correctly predicting the labels compared to the true labels. While this metric is typically assessed

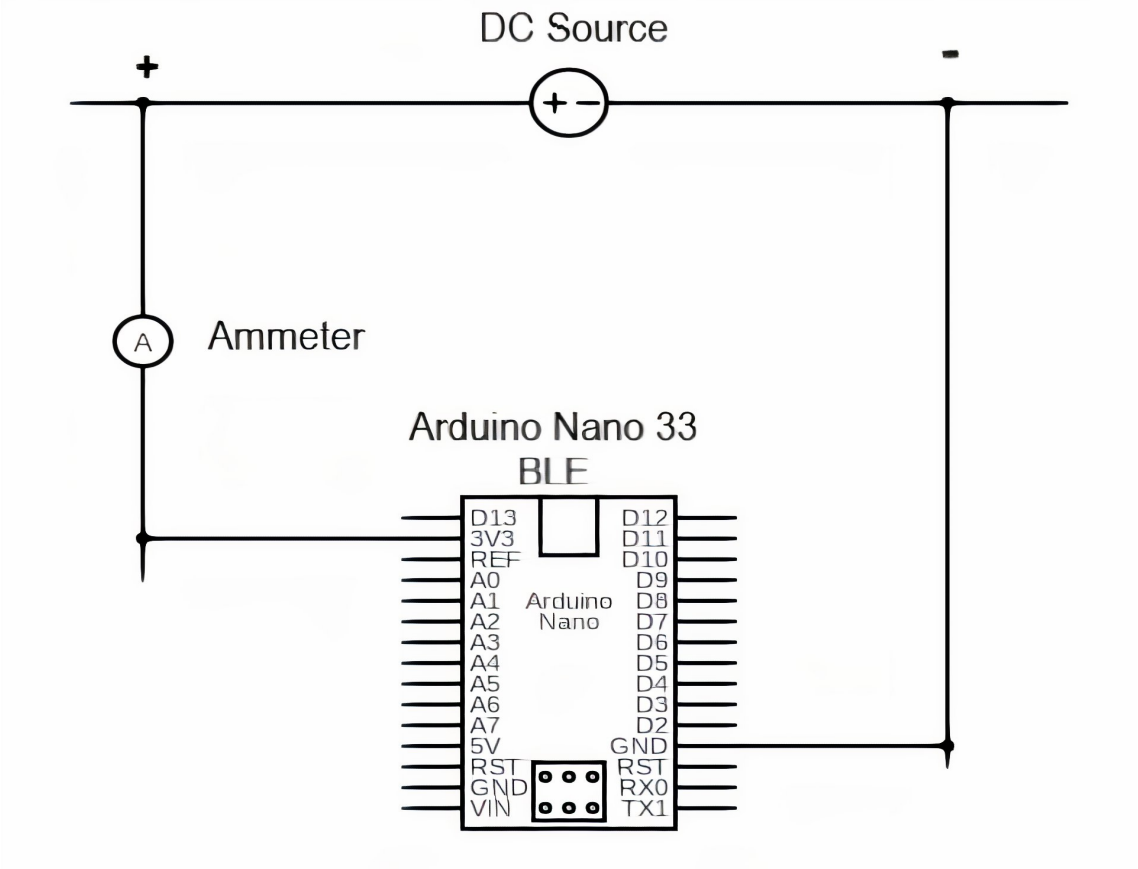

FIGURE 2.9: Circuit diagram to measure the current consumption of Arduino Nano 33 BLE

during the model's training phase on a standard dataset, for the purposes of this report, we consider the accuracy results provided through our validation using a separate test dataset. Memory consumption is critical for ensuring that the deployed model does not exceed the hardware's capacity. For the Arduino Nano 33 BLE, we monitor both the static memory (used for storing the program itself) and dynamic memory (used for runtime operations). The Arduino IDE provides this information during the sketch upload process, reporting the amount of flash memory and the RAM used.

The efficiency of a framework extends beyond its ability to generate good quantitative results. It also includes the effectiveness and ease with which these results are achieved. As these frameworks require significant human interaction, the design and user-friendliness of the framework play a crucial role in successfully building, training, and deploying AI models on our target device. In our evaluation, we also want to assess the usability of each framework based on a set of established usability principles. These principles will help us understand how intuitive and accessible it is for users, from setup and configuring to troubleshooting errors. Specific aspects to be examined include:

- **User Control and Freedom**[40] - ability to configure the training process (epoch, batch size, learning rate) to achieve optimal performance, ability to configure the neural network architecture (layers/operations supported) to support wide range of application needs, ability to configure data file structure into the framework's required format (for CSV files), ability to customize the resolution of the framework application.
- **Clarity**[40] - the ability to have clear text describing the functionality of a feature.
- **Help and Documentation**[41] - diversity in tutorials/examples of datasets (image, video, audio, IMU etc.) and hardware boards (Arduino Nano 33 BLE, Raspberry Pi etc), clear and precise documentation for performing inference, scale and average response time in community forum.
- **Compatibility**[42] - hardware device compatibility.

By systematically measuring these metrics, we can provide a comprehensive evaluation of each Edge AI framework's suitability for deployment on the Arduino Nano 33 BLE, guiding optimal framework selection based on the specific needs and constraints of the deployment environment.

2.4 Dataset

Initially, our project was designated for rail applications, specifically for classifying rail track vibrations. However, we encountered significant challenges in sourcing appropriate datasets. The closest dataset that we could find was from a joint research project by the German Centre for Rail Traffic Research at the Federal Railway Authority (DZSF),

Digitale Schiene Deutschland / DB Netz AG, and FusionSystems GmbH [43]. The dataset consisted of sequences of multi-sensor data like colour camera, infrared camera, lidar, radar, localization, and for our particular interest IMU. These sensors were placed at 21 different railway tracks in Hamburg, Germany. However, the IMU data recorded at each location had only about 9-12 data points with each data recording in a separate CSV file [43]. Despite extensive searches, no suitable data on rail track vibrations were available that met our research needs. Consequently, we opted for the MotionSense dataset, which, although originally intended for human motion classification, offers similar characteristics in terms of accelerometer data. It was readily available and easy to understand, making it a practical alternative. Despite its primary focus on human motion, it shares several attributes with the type of data we expected to analyze for rail track vibrations, such as patterns and frequency ranges relevant to our classification goals. This similarity has allowed us to proceed with our analysis while adapting the methodologies initially developed for rail track vibration classification. The MotionSense Dataset [44] is recorded from the sensors of an Apple iPhone 6s. It's a time-series data generated by an accelerometer sensor of the iPhone recorded at 50Hz frequency in gravitational acceleration (G's) unit of measurement [45]. Using the SensingKit (A Multi-Platform Mobile Sensing Framework), data is collected from the CoreMotion framework (responsible for giving values from the sensors of the iPhone) of the iOS. This data was recorded from 24 participants from a range of genders, ages, heights, and weight values performing 6 different activities: Walking, Sitting, Jogging, Standing, Upstairs, and Downstairs. A total of 15 trials were performed within the same environment keeping uniform conditions over all trials. For each trial, the Crowdsense app (a tool based on an open-source library SensingKit) from the app store was used while the iPhone was kept in the trouser's front pocket of the participants while performing each activity.

The data is logged into multiple CSV files where each file is one sequence of data representing an activity from a single participant. The first column represents the sequence number and the next three columns represent raw acceleration values along X-axis, Y-axis, and Z-axis as shown in the figure 2.10. All of these CSV files from one activity are grouped into a folder representing the label of the respective files.

From the whole dataset, 10% of the data was completely separated from the training and testing processes of the framework to ensure that the frameworks are biased on the popular datasets available on the internet.

2.5 Data Preprocessing

2.5.1 Sliding Window Technique

Sliding Window is a computational technique commonly used in signal processing and time series analysis, which involves creating subsets of a larger dataset to analyze or

	A	B	C	D	E
1		x	y	z	
2	0	0.56908	1.45737	-0.67998	
3	1	-0.04892	1.88261	-0.35477	
4	2	-0.15527	2.1059	-0.17993	
5	3	0.06329	1.5078	-0.05658	
6	4	-0.03566	0.93608	-0.04901	
7	5	0.03391	0.55469	-0.27225	
8	6	0.2915	0.58188	-0.21861	
9	7	-0.01697	0.84241	-0.41289	
10	8	-0.00645	0.64606	-0.38568	
11	9	-0.02936	0.36589	-0.21533	

FIGURE 2.10: Structure of the data file

process sequential data points within a fixed-size segment or 'window.' This window length corresponds to the time segment over which you want to analyze the data and the stride or step size determines how many points the window moves over for each step—it can be just one point (a stride of 1) for high overlap or more for less overlap. At each step, the data within the window is extracted and can be treated as a single sample for analysis or feature extraction as demonstrated in the figure 2.11 [46].

In our case, accelerometer data is time-series signals that capture the dynamic movements of the human body along three spatial axes (x, y, z) at 50 Hz, that is, one data point every 20 ms. When trying to classify human activity, such as walking, jogging, sitting etc., it's beneficial to look at segments of data over a period of time rather than individual data points.

As discussed in the article by Wang Gaojing et al. [47], window length has a significant effect on human motion recognition. In the comprehensive study conducted by Wang G concluded that the window length of 2.5 to 3.5 s provides the best tradeoff between performance and latency. In our case, this translates to 125 to 175 data points in a single window. Each window can be used to extract meaningful features that describe the activity within that period. These features could be statistical measures like the mean, variance, and peak values or more complex time and frequency domain features. By using overlapping windows (with strides smaller than the window length), the technique ensures continuity between samples, which can lead to smoother and more accurate activity classification [48]. For real-time applications, sliding windows allow the system to continually update the activity classification as new data is received, enabling ongoing

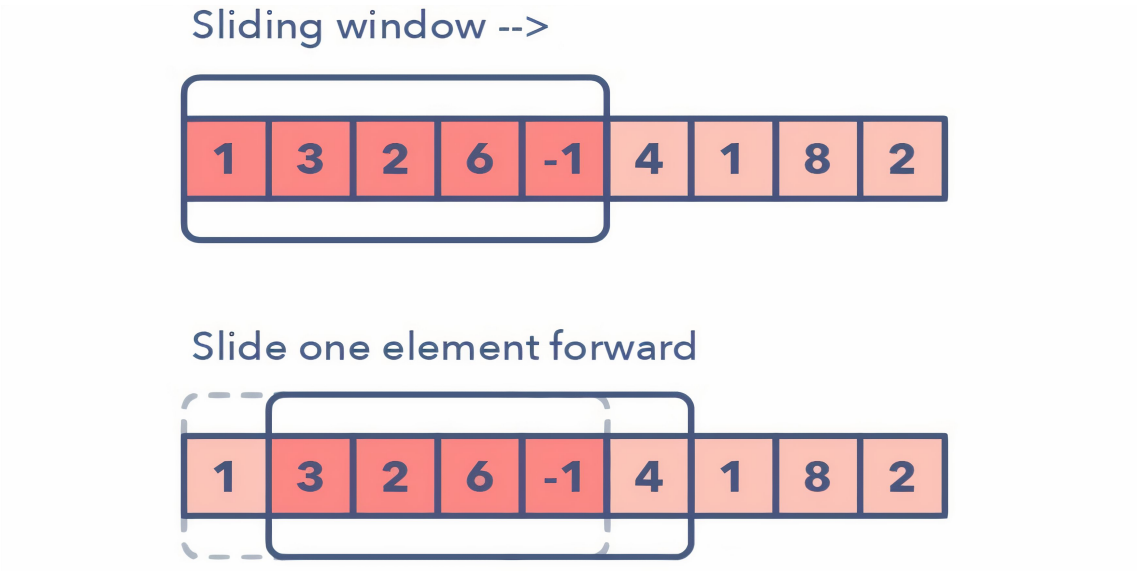

FIGURE 2.11: Sliding Window Technique

monitoring.

2.6 Hardware Device

In terms of the edge device for testing and deployment of the machine learning models, we are using the Arduino Nano 33 BLE board. It is versatile hardware with an onboard 9-Axis LSM9DS1 IMU sensor module and a Low Energy 5.0 Bluetooth module for connectivity. The IMU sensor features a 3D accelerometer, gyroscope and magnetometer allowing the detection of vibration or any movement while the 2.4 GHz Bluetooth module can be used to transmit the data between other devices with BLE capabilities [49]. The integrated IMU simplifies the hardware setup without any need to connect extra peripherals and enables to run the AI applications requiring vibration analysis like motion detection, gesture recognition etc seamlessly. The main processor powering the board is an Arm Cortex-M4F running at 64MHz with 1 MB Flash and 256 kB RAM. In terms of peripheral connectivity, it offers UART, I2C, and SPI communication with the board's I/O voltage of 3.3 volts.

Running on an open-source operating system Arm Mbed OS specifically designed for Cortex-M boards, it is power efficient and provides an abstraction layer for developers to write Mbed-enabled C/C++ applications [50]. Arduino IDE, Arduino CLI, and Arduino Web Editor are some of the software that allows programming the board along with some 3rd party applications like Platform IO IDE which is built on top of Microsoft Visual Studio Code providing a toolset for C/C++ development. The extensive Arduino community supports a wide array of libraries, tools, and comprehensive documentation. This ecosystem facilitates a smoother development process and also simplifies troubleshooting, swiftly resolving any issues that may arise.

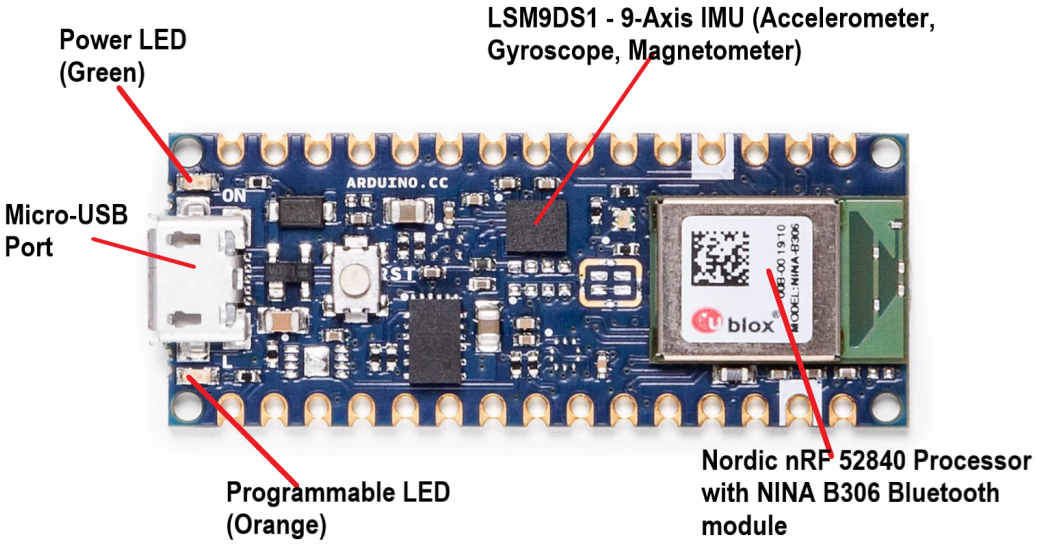

FIGURE 2.12: Arduino Nano 33 BLE hardware overview

Chapter 3

Implementation

3.1 Overview

This section outlines the technical implementation of the frameworks and a systematic approach to ensure precise replicable results. Starting with the data upload, due to differing requirements across frameworks regarding the format of the CSV dataset, a preliminary phase of data preprocessing was necessary prior to model construction. In some frameworks, a digital signal processing (DSP) block to generate and analyze features from IMU signals was required and could not be skipped. These DSP features are then used along with the neural networks to classify the data. However, not all frameworks include a DSP block.

When it comes to building neural networks, the frameworks vary in terms of supported layers and operations. Consequently, we sometimes had to change the network designs to make sure they worked with each framework, but we tried to keep these changes minimal and stick to a standard design we set up in Google Colab with Keras and TensorFlow.

Finally, the deployment methodologies for the trained network also differ among the frameworks. Some, like Edge Impulse, come as a ready-to-use package library, while others generated C++ source code that we then add to an Arduino project. This code then must follow specific guidelines to interface the model on the hardware board. Of the nine frameworks initially proposed for evaluation, four (Edge Impulse, NanoEdge AI Studio, Imagimob Studio, TFLM) successfully generated the required results for assessment. Three frameworks (Mathworks, uTensor, Edge-ML) encountered implementation challenges, which are detailed in their respective sections. Additionally, the evaluation of two frameworks (ELL Microsoft, Arm CMSIS-NN) was anticipated to be beyond the scope of this thesis due to time constraints, as identified at the project's inception.

3.2 Baseline architecture of Neural Network

In this section, we outline the process of creating a baseline neural network architecture that can be adopted by other frameworks. This baseline model serves as a foundational template, ensuring consistency and comparability across different EdgeAI frameworks. The development process began with data preparation and culminated in the training and

evaluation of a simple Convolutional Neural Network (CNN) using Google Colab and Keras.

The initial step involved preparing the raw dataset, which was stored in Google Drive. By uploading the dataset to Google Drive, we facilitated easy access and manipulation within Google Colab. Using the Colab environment, we first imported the Google Drive to retrieve the dataset which comprised of CSV files containing recorded accelerometer datasets for further processing.

Once the data was imported, the next task was to generate sliding windows from the CSV data. Sliding windows are essential for transforming time-series data into a format suitable for training neural networks. In this work, we generated windows of 2.5 seconds, corresponding to 125 data points, given the data's 50Hz recording frequency. The sliding step was set to 20 milliseconds, equivalent to one data point, ensuring a comprehensive capture of temporal patterns in the dataset. This process was applied to the entire dataset, creating windows for all labels and concatenating the results into the Xtrain and Ytrain arrays for the training set as demonstrated in appendix A.

For the test dataset, we employed a different strategy to ensure robust evaluation. We randomly selected 20% of the CSV files from each label and placed them into a separate folder. Sliding windows were generated for these files as well, resulting in the Xtest and Ytest arrays. During the window generation, we also created corresponding labels for both the training and validation sets, stored in Ytrain and Ytest respectively. To enhance the robustness of the training process, both the training and testing datasets were shuffled before being fed into the neural network.

With the data preparation complete, the focus shifted to designing the neural network. Using Keras, we built a simple CNN model as shown in appendix A, incorporating essential layers to effectively capture and learn from the data patterns. The model architecture included Conv2D layers for convolution operations, MaxPooling2D layers for down-sampling, and two dense layers for classification. This structure provided a balanced combination of feature extraction and decision-making capabilities.

The model was compiled using the Adam optimizer, known for its efficiency in training deep-learning models [51]. We employed sparse categorical crossentropy as the loss function, suitable for multi-class classification tasks [52]. After compiling the model, we proceeded to train it using the prepared training dataset. The model was trained for four epochs with a batch size of 32, to balance the training time and model performance. The training process involved iteratively updating the model parameters to minimize the loss and improve accuracy.

Following the training phase, the model was evaluated using the Xtest and Ytest datasets. This evaluation provided insights into the model's performance and its ability to generalize to unseen data. The creation and training of this simple CNN model established a baseline architecture that other frameworks can follow. This approach ensures a standardized methodology for evaluating and comparing the performance of various

EdgeAI frameworks on the Arduino Nano 33 BLE board.

3.3 Frameworks

3.3.1 MathWorks

Matlab Simulink Online version

MathWorks provides free licenses for students at the university, which can be accessed using a university email to register. This includes access to the online versions of Matlab and Simulink, which are particularly useful for those with limited device performance power. We initially opted to use these online versions for our project. The first step involved uploading the dataset to Matlab Drive; initially, we selected five files from each subfolder of the main dataset to familiarize ourselves with the functionality and interface of Matlab Online.

After uploading the data, the next task was to import it into a Matlab live script file. We utilized the built-in function 'importfile()' to transfer the data from Matlab Drive to the script file. This function was configured to exclude headers, with the first column containing the serial numbers of the data points. Subsequently, we processed the data to create sliding windows of length 150 and a stride of 1. All generated windows were stored in a 3-dimensional array sized ['total number of windows' x 150 x 3]. This array was then transformed into a cell array, with each cell representing a single sample window of [150x3 double] type for simplicity. Additionally, we created a corresponding cell array for labels, with each cell representing a single label. Since the labels were string values, they were first converted into categorical values before being stored in the cell array.

Regarding the neural network structure, the configurations were identical to those of the neural network trained in Google Colab in Python, except for two layers: 'sequenceInputLayer' and 'classificationLayer', which define the input and output layers of the neural network, respectively. Based on Matlab documentation [53], we chose to use 'trainnet' instead of 'trainNetwork' for training the deep neural network. 'trainnet' is recommended because it supports deep learning network objects, allows easy specification of loss functions, offers better visualization, and is faster than 'trainNetwork'. However, we removed the 'classificationLayer' as 'trainnet' does not support an output layer.

After completing the training, the next step was to install the Arduino support packages for Matlab and Simulink. Unfortunately, these Add-Ons are not supported by Matlab Online, necessitating the installation of the software locally on a PC.

Matlab Simulink On-device installed version

During the installation of the MATLAB application, a custom location was chosen instead of the default (typically the C: drive) to keep the installation organized. Simulink was installed concurrently with MATLAB by selecting it from the additional software options

provided during the MATLAB installation process. Once the installation was completed successfully, the Arduino support packages for MATLAB and Simulink were added separately.

Although the overall installation was completed without issues, some problems arose during the setup process. For the Arduino support package in MATLAB, after configuring the correct board settings and libraries for communication via USB, the libraries were successfully uploaded to the Arduino Nano 33 BLE board. However, the initial attempt to test the connection failed. After manually switching the Nano board into Bootloader mode and reconnecting it to the PC several times, a successful connection was finally established. Fortunately, this issue did not recur; subsequent device tests were successful, and the board automatically entered Bootloader mode as facilitated by MATLAB.

In the case of Simulink, a similar issue occurred during the connection test, despite the successful installation of the Arduino support package. The connection test between the board and the PC failed repeatedly over USB. Following advice from a MATLAB staff member on the MATLAB Answers forum, I attempted to update the Bootloader driver from the Arduino support package installation folder. However, this did not resolve the issue, and the connection problems persisted. This ongoing issue was subsequently raised in the MATLAB community forum, in hopes of finding a solution. Recently, a staff member responded, advising me to contact and register a case with MATLAB's technical support team for further assistance. Consequently, a support case was registered with MathWorks under case number: 06996279, as illustrated in figure 3.1. The MathWorks team responded by instructing me to apply specific patch files from the MATLAB command window before proceeding with the Simulink test connection setup. Once applied, these patches resolved the error, successfully establishing the connection between Simulink and the Arduino Nano 33 BLE. The patch files adjusted the board configuration and settings and are typically used to update or rectify issues in existing code without requiring a complete reinstallation of the software. Upon inquiry about the cause of the connection error, the support staff identified a bug in their existing code, as highlighted in figure 3.1. This discovery suggests that the issues encountered during the latter stages of development between Simulink and the Nano 33 BLE might stem from software bugs rather than the methodology itself. However, due to time constraints, this hypothesis could not be conclusively verified.

After successfully installing the support packages, the neural network was trained on a larger dataset with the same configurations used in Matlab Online. To establish communication with the Arduino Nano 33 BLE board, two objects were created: one for connecting to the Arduino, and another for interfacing with the onboard 9-axis IMU sensor LSM9DS1 using the I2C library. The bus address was set to 1, facilitating successful communication and the display of accelerometer values. Despite the LSM9DS1 object supporting a 'Sampleperread' argument, multiple acceleration values could not be displayed in a single read due to the limitations of the 'readAcceleration' function, which can only

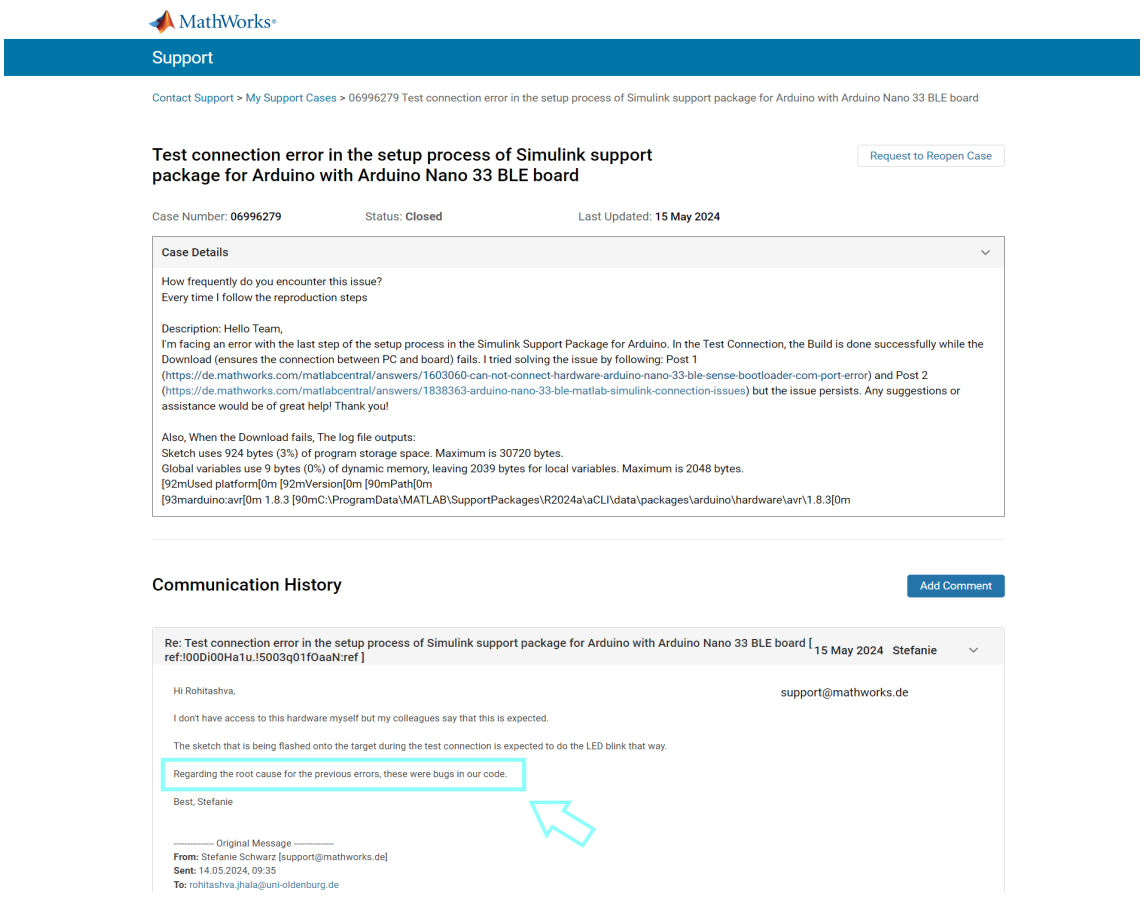

FIGURE 3.1: Conversation with Mathworks technical support team for case: 06996279

process one sample at a time [54]. To read multiple values, the function had to be invoked repeatedly within a loop.

When working with Simulink to test the connection, a simple LSM9DS1 sensor block from the Arduino library of Simulink was added and connected with the Scope for visualization. Upon building the project, path variable errors were reported. To solve this the file system path (where the configuration files are located for Arduino Avr platform to work with Matlab) had to be added in the PATH of environment variables. Once solved another error was reported when building the project, wherein the gcc compiler is not located in the path specified by the Matlab as shown in the figure 3.2. With no

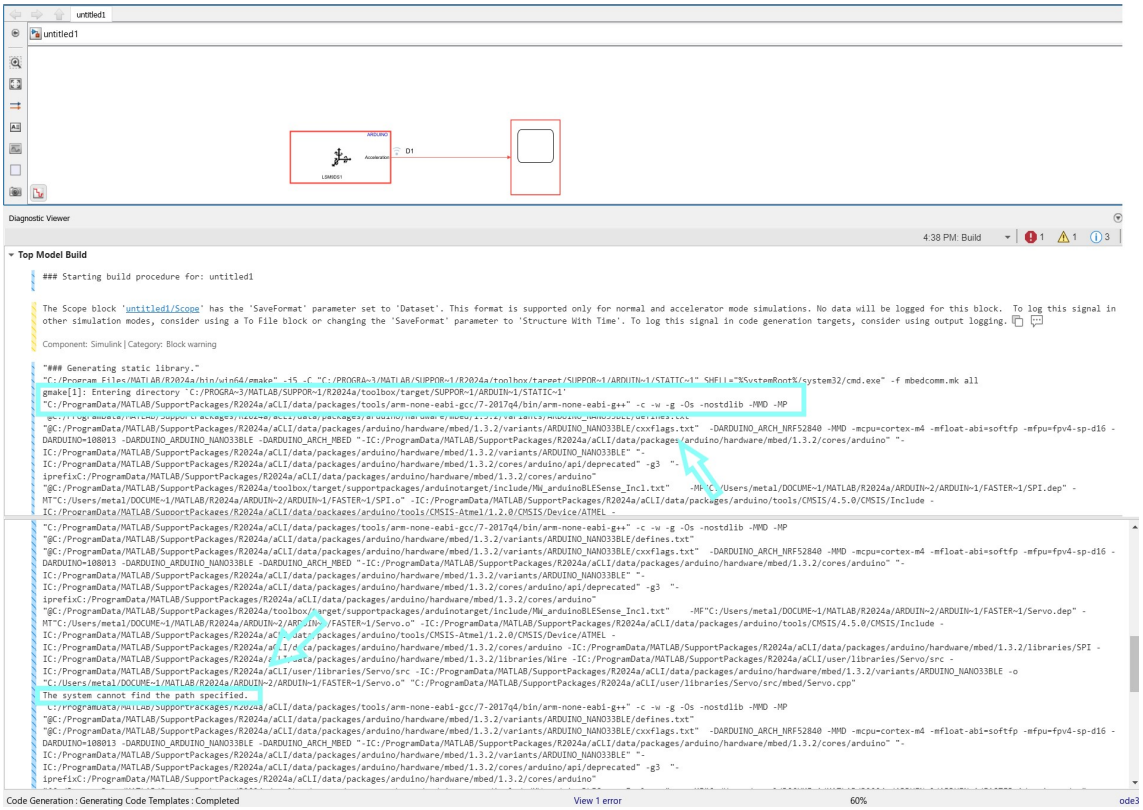

FIGURE 3.2: Simulink error for unable to find the path of gcc compiler

active solution to be found after multiple tries, this error was posted on the MathWorks community forum. One reply was received with a possible hint to the problem being improper installation and to bypass the problem, the necessary support package file had to be replaced from the one attached in the answer. The new file is for a standard Arduino installation with version 1.8.10 while the old one seems to point to a MathWorks-specific version of Arduino software, version 0.33.0, compatible with R2024a. Both files contain configuration data for what appears to be a software installation or update process, specifically related to the Arduino software for the Windows 64-bit platform. The new file supposedly facilitated the communication with an Arduino server within the file located in the support package directory of Matlab. After making the new changes suggested in

the response on the forum, the error persisted and no further solution could be found. To crosscheck the error, we build again the same project but with the hardware specification of Arduino Mega 2560 and the build finished successfully without any errors. In order to bypass the problem, instead of sensor blocks from the library we defined our own Matlab function block performing the same functionalities. With the Matlab function block, we tried to bring the same functionality tested in Matlab to Simulink by declaring Arduino and LSM9DS1 sensor objects and using the 'readAcceleration' function to read the values. These values were then fed into the Scope for visualization. In this very case upon building the project, an error for Matlab code generation not supported for Nano 33 BLE was reported as shown in figure 3.3. When using an in-built example of Pitch and Roll from Simulink, the same error was reported for the Matlab function block in the example. In an in-built Matlab example to identify shapes using machine learning on Nano 33 BLE Sense, No deployment on the board was exercised. Instead, the board was used to read the IMU values and create a dataset for training the ML model in Matlab. This error was also then posted to the MathWorks community forum for further insights.

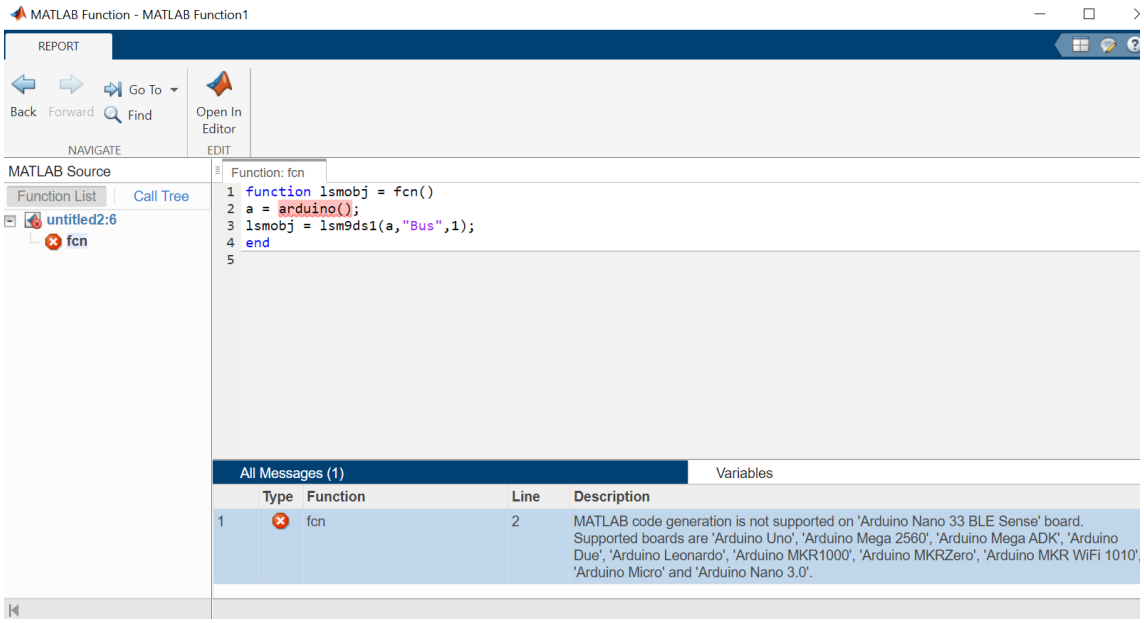

FIGURE 3.3: Simulink codegen error when using function blocks with Arduino Nano 33 BLE

With multiple errors in Simulink and very little insight about them on the internet, It was a roadblock for any further development. So we decided to uninstall the Matlab software and all the related packages from the PC following the guidellines [55], even deleting the PATH environment variables. Thereafter, following the suggestion from the MathWorks forum in one of our responses that the possible cause of errors could be the improper installation. We disabled Windows Security completely, and reinstalled the Matlab and all the support packages at the default system suggested location (in program data in C drive), using the home network to avoid any potential firewall issues

from eduroam (University's Wifi). Despite these measures, the installation completed successfully, but in the setup process of the Arduino support package for Simulink, we encountered the very same error of failed connection. The support package could not establish a connection between the board and the PC in the connection test. Running the Simulink project to read acceleration values again resulted in errors, indicating that the Matlab function block was not supported for the Nano 33 BLE—a limitation confirmed upon reviewing the documentation. The Matlab function block feature is only supported for some boards and Nano 33 BLE is not listed there. As for the other Simulink project with LSM9DS1 sensor block, the same error of the system being unable to find the gcc compiler's specified path occurred. In this error case, as we exercised before according to a response on the MathWorks forum, we changed the support package file with the one attached in the response and restarted the Matlab and the PC. The error still persisted and we replied the same with details on the MathWorks forum. Then we tried changing the location of the gcc compiler by explicitly mentioning the actual location in the Matlab command window but it didn't work. It still searched for the compiler at the originral location. Finally, we manually copy paste the folder from the actual location to the location where the Matlab was searching and it worked. There were no more gcc compiler errors. The project was built and deployed on the Nano 33 BLE board successfully but it didn't give any readings when checked with the visualization scope as shown in the figure 3.4. Following no errors with the deployment of this project, we this time tried building the Matlab function block project for reading acceleration values but it gave the same error for not being supported for the Nano 33 BLE board. The Arduino support package for Simulink didn't specify any communication options with Nano 33 BLE over Bluetooth or Wifi unlike Matlab where the Arduino support package had the option to setup the board via Bluetooth or Wifi and the connection was also established successfully upon testing.

MathWorks suggested that the capabilities of EdgeAI can be leveraged using Matlab and Simulink together but, in our case with the Arduino Nano 33 BLE board, it didn't work. Another solution that could work but with a different approach than the Simulink was by using the Matlab Coder extension which basically converts the Matlab code to an equivalent C/C++ source code for deployment on embedded hardware. It can also be used to generate static or dynamic libraries. In combination with Embedded Coder, it provides hardware-specific optimizations, code traceability, and code verifications before generating the embedded code [56]. In addition to Matlab Coder and Embedded Coder extensions, we also added a Matlab Coder interface for deep learning libraries as it was mandatory for our use case. Once installed, a Matlab function as a .m file was defined which takes the input data and uses the trained network to predict the corresponding output classification. The trained network actually contains the model trained over the whole training dataset and saved as a .mat file for use in this function. In the Matlab coder, once you specify the entry-point functions and the input types, it checks for the run-time issues by generating an MEX function from the entry-point functions and invoking it to

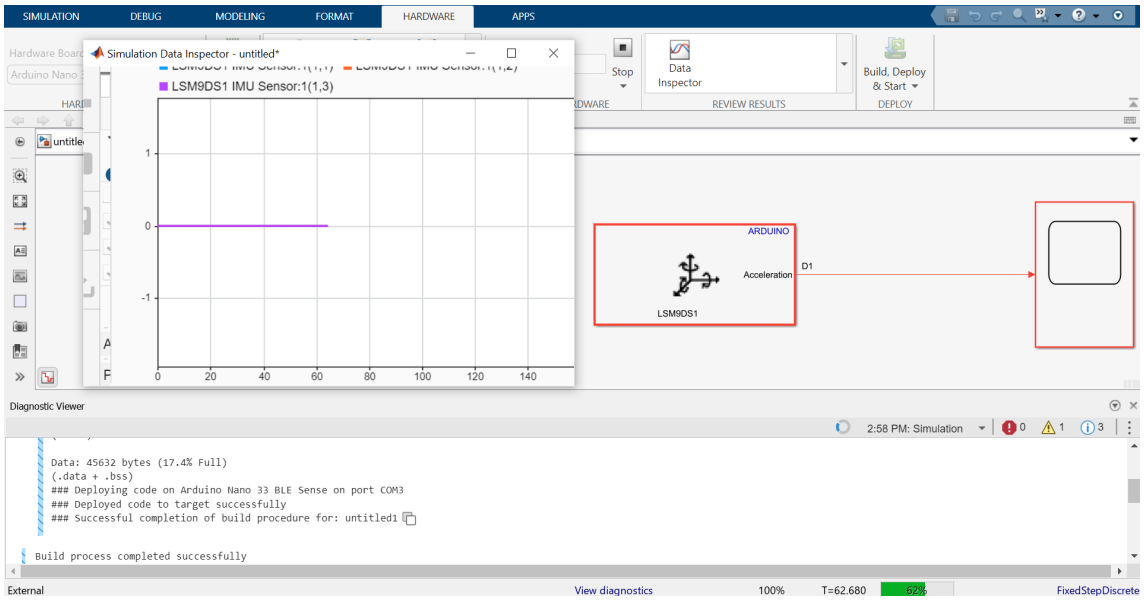

FIGURE 3.4: Scope showing zero acceleration values for the LSM9DS1 IMU sensor

report any issues that might be hard to diagnose in the C code. Once it passes the test, the library or the source code can be built by specifying the hardware board, the device and the toolchain to use. For the toolchain, we chose the option to automatically locate an installed toolchain and use it for building the generated code. As for the device and the hardware board, we tried multiple options.

One with hardware board as ARM Cortex-M4 (which is the processor in the Nano 33 BLE) in which case it automatically selects device type as ARM Cortex-M and device vendor as ARM Compatible. Additionally, we also had to add an Embedded Coder Support Package for ARM Coretx-M Processors for this use case. In this particular option, we encountered the error *"Failed to apply the 'addedToConfig' method to the Hardware property: In 'targetsdk:functions:ThirdPartyMissing', parameter 1 must be a real scalar,"*. It suggests a problem with configuring or using third-party support or tools within MATLAB, particularly related to code generation or hardware configuration.

Second, with hardware board not specified, which gives multiple options for device vendors and types to choose from. We selected device vendor as 'ARM Compatible' and the device type is 'ARM Cortex-M'. We again run the code generation and save the package as a zip file. This zip package contains all the necessary header and source files important for building the project in C. After importing this package in the Arduino libraries, with Arduino IDE I create a project and include the files enclosed within the package. By using the LSM9DS1 sensor to read real-time acceleration data, and feeding it to the network predict function defined in the library, we create a project that is now ready to work with the Nano 33 BLE board. When compiling it gives the error *"omp.h file not found"*. After a brief search on internet, we circumvented the error by correctly configuring a parameter. Following this, while uploading the model on the hardware,

we encountered another error as shown in figure 3.5. This is a memory overflow error becasue the model generated through code gen has a larger memory footprint than what is available on the harware itself. when compiling the program. Following this, we

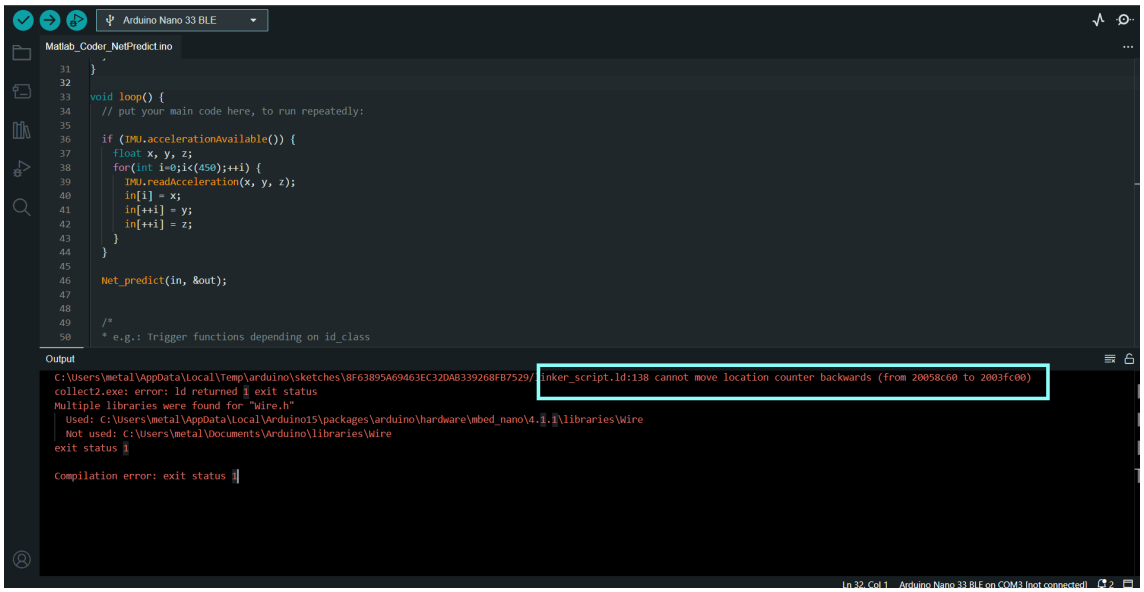

FIGURE 3.5: Memory overflow error for model generated using Matlab Coder

installed the Deep Network Quantizer package in Matlab to try quantizing the network. For this we had to build a new neural network from ground up as the application does not support importing an architecture already defined in Matlab. This approach resulted in another error: Deep Network Quantizer does not support quantization for CPU execution environments as shown in figure 3.6. Thereafter, we tried other approach to quantize the network within Matlab before generating the source files from Matlab Coder. Here we found that code generation for quantized network is not supported as shown in figure 3.7. Here at this point, we decided to stop the development process with MathWorks due to time restrictions.

3.3.2 uTensor

uTensor or MicroTensor is an open-source machine learning interface library essentially build to convert a TensorFlow model into a C++ source file for embedded deployment. The primary objective was to deploy a TensorFlow Lite (TFLite) model on an Arduino Nano 33 BLE using uTensor. The initial step was to convert the TFLite model into C++ code using uTensor tools.

Setting up the environment involved creating a Python virtual environment to isolate dependencies and then installing the utensor-cgen tool, which is responsible for generating the C++ files. An older version of TensorFlow, specifically version 1.15, was also required due to compatibility with uTensor as shown by the error in the figure 3.8. The challenge

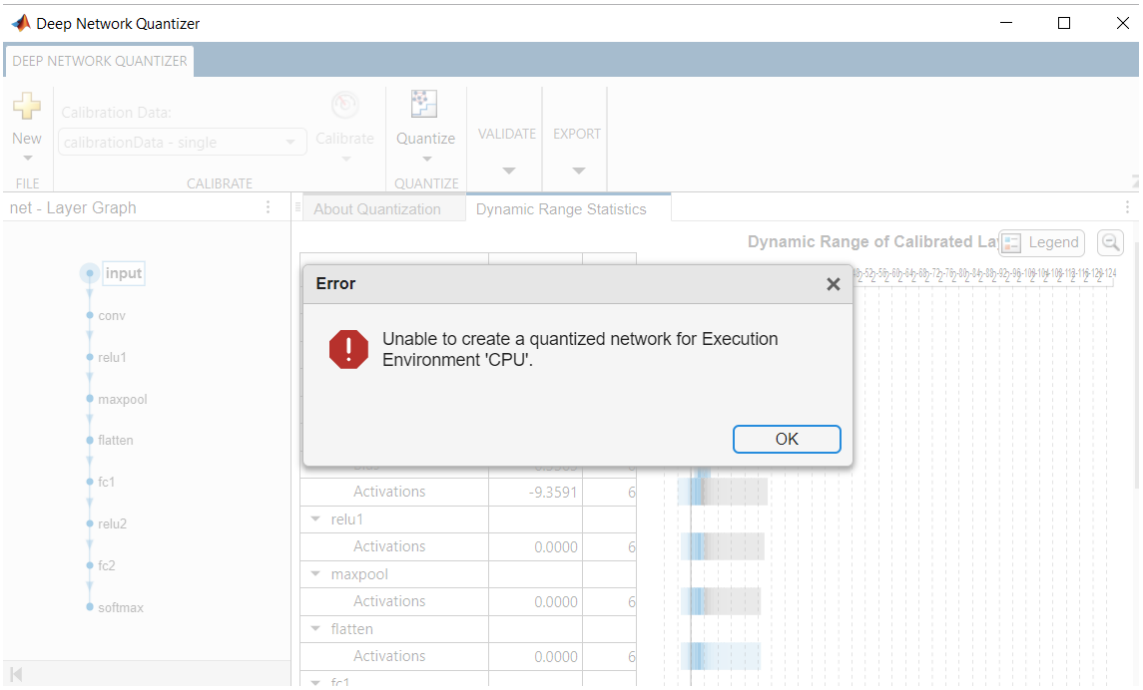

FIGURE 3.6: Quantization error for CPU environments from Deep Network Quantizer package

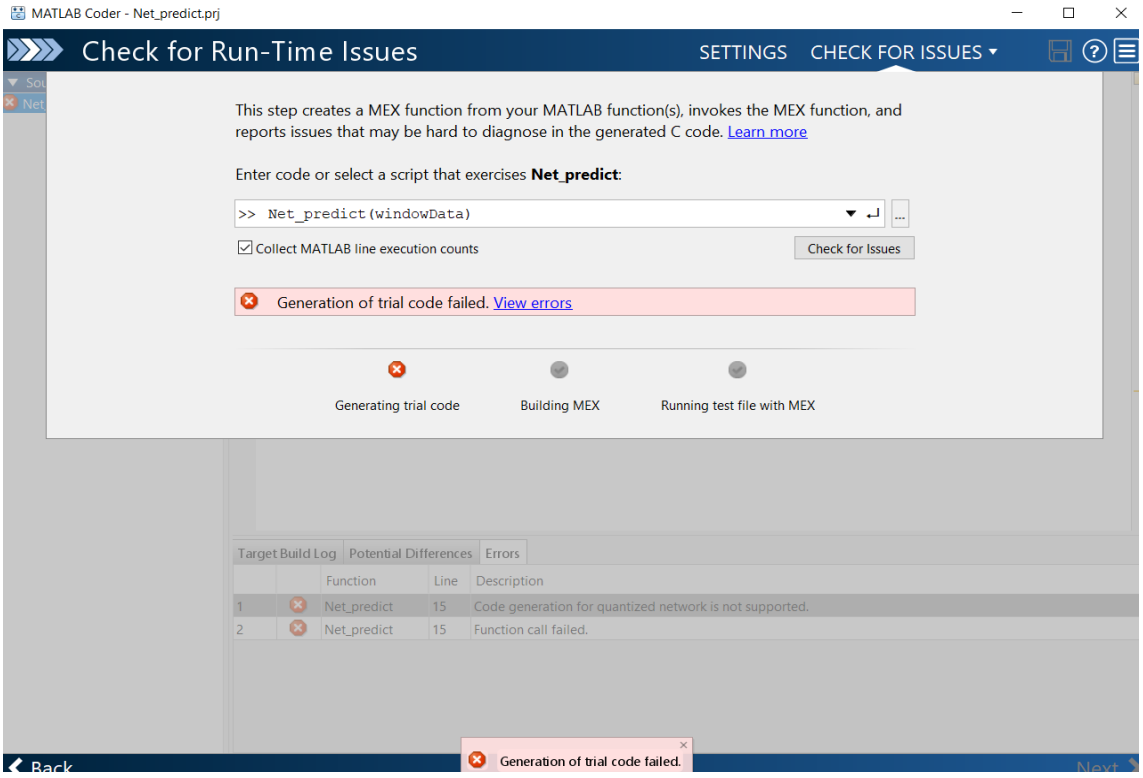

FIGURE 3.7: Quantized network not supported for code generation in Matlab

began with locating a version of TensorFlow that would work within the constraints of uTensor. Installing TensorFlow 1.15 was tricky because it isn't compatible with Python versions beyond 3.7. Thus, we had to downgrade Python to version 3.7.

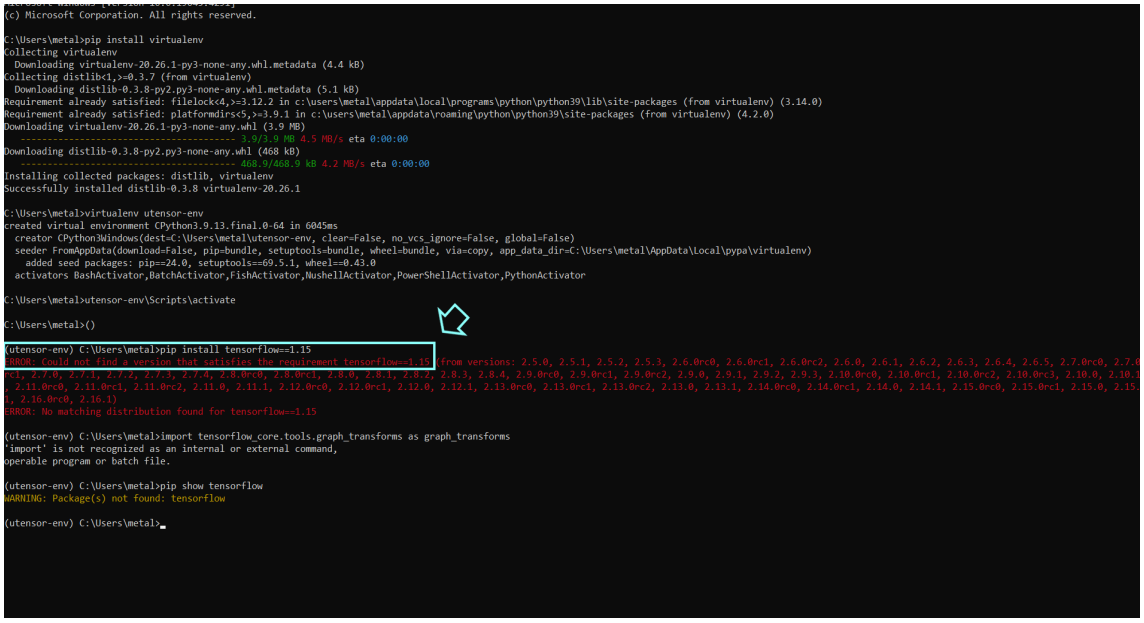

FIGURE 3.8: Error showing TensorFlow version 1.15 is required for utensor code generation

Once the environment was prepared, the next step was converting the TFLite model. However, the uTensor tools threw an error, indicating that .tflite files are not directly supported. Instead, uTensor expects models in the Protocol Buffers (PB) format. This required converting the TFLite model to a frozen GraphDef .pb file. A Python script was developed to handle this conversion, but errors arose due to mismatches in TensorFlow versions and attributes like explicit_paddings not being supported. The solution was to ensure the model architecture did not include unsupported attributes and to explicitly use SAME or VALID padding in convolution layers.

The export script was tested in Google Colab to freeze the model into a .pb file. However, TensorFlow 2.x generated errors like "NodeDef mentions attr 'explicit_paddings' not in Op." as shown in the figure 3.9. The workaround was to ensure compatible versions of ONNX and ONNX-TF were installed. Specific versions of ONNX (1.6.0) and ONNX-TF (1.2.1) were required, along with downgrading Protobuf to version 3.20.3. Despite these changes, the errors persisted, so the model's convolution layers were adjusted to use only SAME or VALID padding.

After successfully generating the frozen .pb model, the next challenge was to convert this GraphDef model into the uTensor format using utensor-cli. This tool required the output node name to be specified precisely, which in our case was the last layer of the model (dense_1). The conversion process led to additional errors, such as "ValueError:

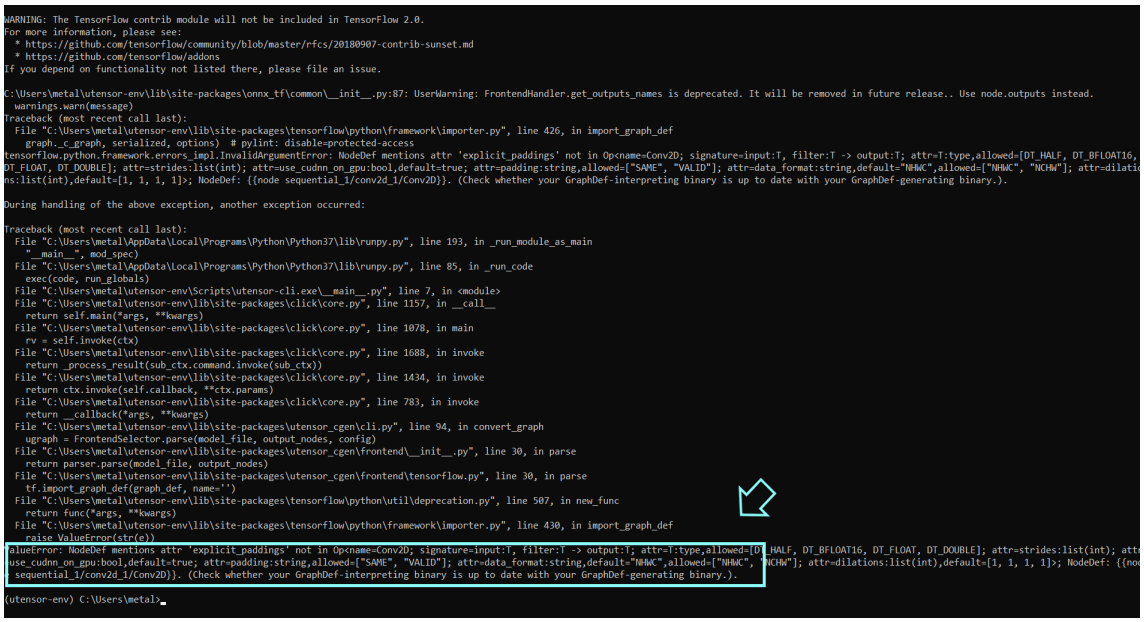

FIGURE 3.9: Error using utensor code generation to convert the tflite model into C++ source files

NodeDef mentions attr 'explicit_paddings' not in Op." These errors were tackled by ensuring the input model file was explicitly compatible with uTensor.

In summary, implementing EdgeAI with uTensor involved multiple challenges, ranging from version incompatibilities and unsupported attributes to tool limitations and conversion errors. The lack of support for fully quantized models and errors like "unknown model file ext found: .tflite" required several workarounds. Unfortunately, because of the timing restrictions in a thesis project, being limited, we decided to move to another framework. Despite the difficulties, this Implementation serves as a guide for practitioners to navigate the toolchain effectively and highlights the importance of understanding the limitations of both the model architecture and the uTensor tools.

3.3.3 Edge-ML

Edge-ml is an open-source cloud-based platform that requires an online account registration. Once the account has been registered, you can log in with the credentials. Upon logging in you receive a whole page pop-up message with terms mentioning that the edge-ml is a beta deployment and that all the account-related information may get deleted at any time (figure 3.10).

Once you accept the terms (being the only option), you enter into the dashboard where general options like dataset, labellings, models, and settings are provided along with the links to their GitHub and documentation webpage. For uploading the dataset, edge-ml has its own required format [57] for reading the CSV files, which in our case involved performing some data preprocessing on the raw dataset by using the Pandas library. After meeting the data format requirements, the labelling of the data is performed automatically

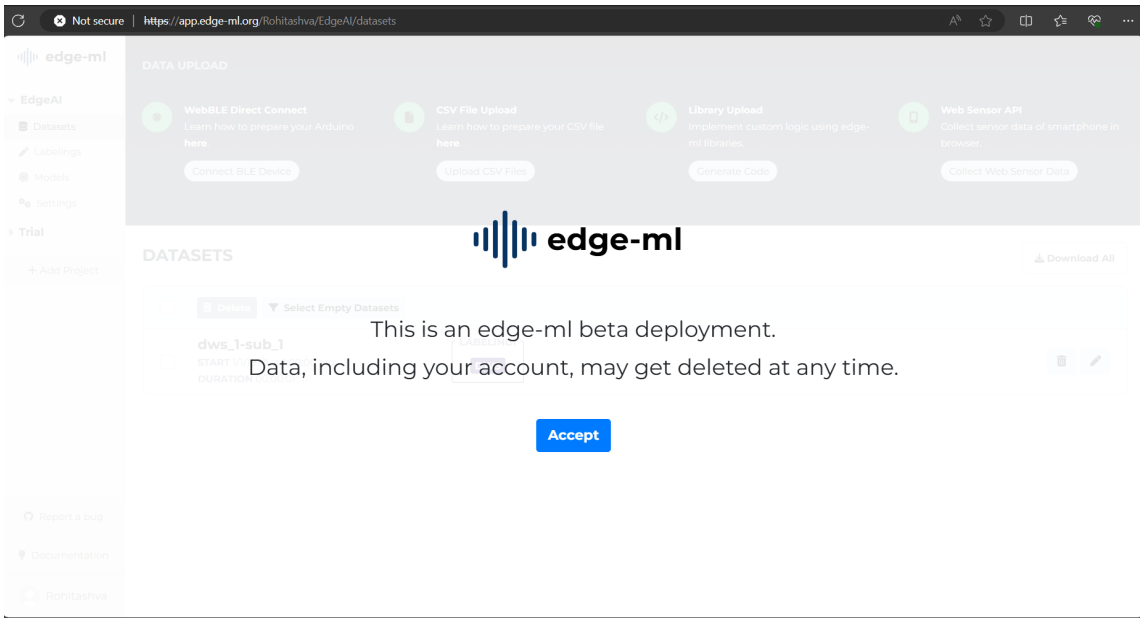

FIGURE 3.10: A Beta deployment pop-up from Edge-ml

according to the labels specified in the CSV file. Users also have the option to change the labels or assign new labels after uploading the data by first, adding the label name in the labelling set and then editing the data from the datasets section. Thereafter, in the model's section, the framework redirects to another webpage, this time "beta edge-ml" as shown in the figure 3.11.

The beta edge-ml website version again requires a new account for login and does not recognize the credentials from the previous edge-ml website where we performed the data upload and labelling. So, now we again have to upload the data (with the same format requirements) to proceed with designing the neural network. In terms of the dashboard and functionalities, the beta version is identical to the regular version of edge-ml. Hence, no additional steps were required once the data was uploaded. In the model section, to train a model, there are two available options. One with a manual classification pipeline and the other with an AutoML classification pipeline. Proceeding with the manual classification pipeline is the only option as the AutoML pipeline mentioned no support for C++ platforms. In the manual classification, once the training dataset and labels are selected, the next step is to specify sliding window parameters to split the training dataset into smaller segments. With a window size of 2.5 seconds and a sliding step of 20 milliseconds, the next step is feature extraction with the only available method a Simple Feature Extractor. No further information or description on the kind of feature extraction technique is mentioned. After selecting the Simple Feature Extractor, the next step offers Normalizer. Out of the two available options: MinMaxNormalizer and ZNormalizer, MinMaxNormalizer is the only option supporting the C++ platform. It normalises the data between the [0,1] range and hence we proceeded with it. The next step is selecting the type of classifier for the classification of the data. The list of classifiers includes Decision

FIGURE 3.11: Beta version of Edge-ml

Trees, Random Forest, Dense Neural Network and Convolution Neural Network. Out of the mentioned, a decision tree is the only classifier supporting the C++ platform as mentioned by the framework. As the basis for comparison between the frameworks is convolutional neural networks, we selected CNN as a classifier. The next step is Evaluator in which either the TestTrainSplit or KFold strategy was to be used. However, no information was provided on whether the split is done before generating the sliding windows or after as the latter one would cause a data leak (as explained in the section 2.1). Selecting TestTrainSplit with an 80-20 split for training and testing, we proceeded with training the model. As shown in the figure 3.12, the model was trained successfully.

As shown in the figure 3.12, the framework shows the available options to either download or deploy the model. Upon clicking the download option, a pop-up window appears showing the language options: C++ and Python, to choose from and then download the model (figure 3.13). Unfortunately, the model could not be downloaded with the C++ language option even though the option was available as shown in the figure 3.13.

Going ahead with the deploy option, it showed a pop-up for generating the firmware for the supported devices as shown in the figure 3.14. Within the list of supported devices, we selected Arduino Nano33 Rev1 and configured the device with IMU acceleration along the x, y and z axes for the features. Now when trying to download the firmware for the device, It did not download and upon inspecting we found out that the download button did not have any link attached. Another option could be to directly deploy the model onto the hardware device, but the features are still in the development phase as mentioned on their GitHub page [23]. Although we did manage to connect the board with the framework via BLE but the features were only limited to recording/generating the

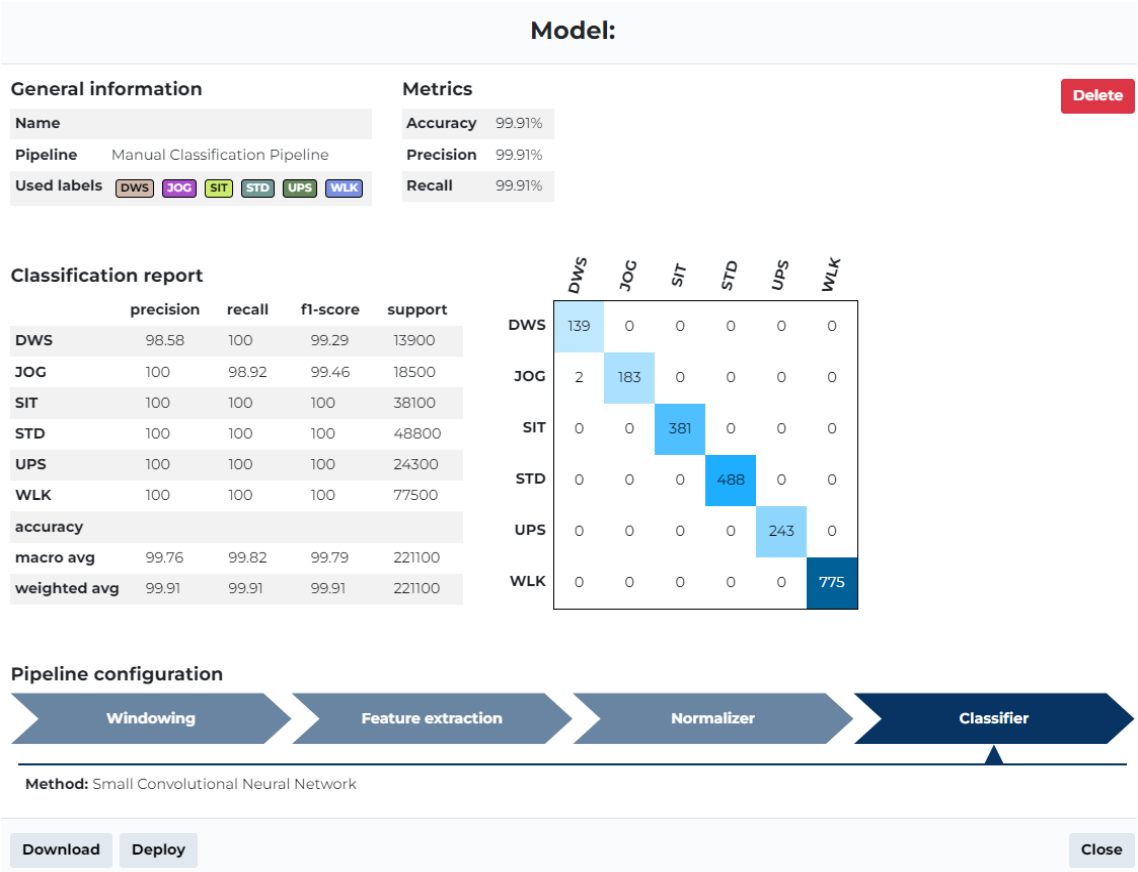

FIGURE 3.12: A convolutional neural network trained with Edge-ml on the MotionSense dataset

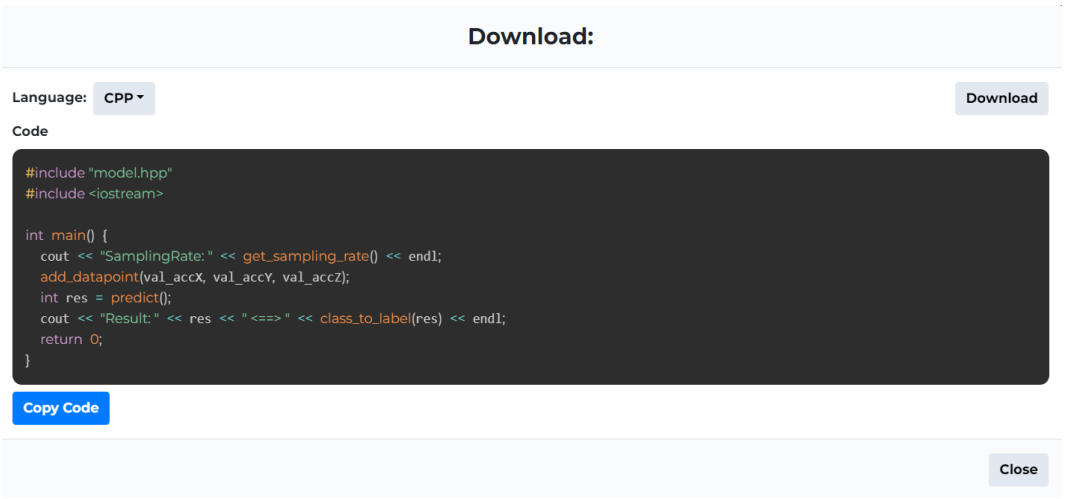

FIGURE 3.13: Download window for the trained convolutional neural network with Edge-ml

dataset from the on-board sensors.

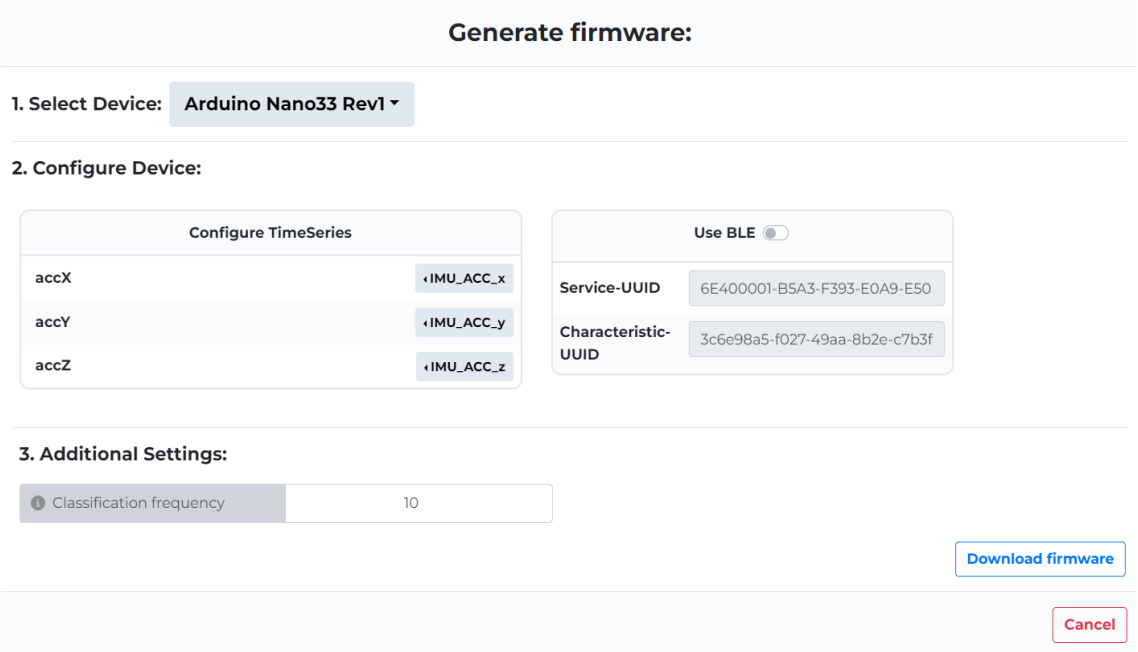

FIGURE 3.14: Window showing feature to Generate firmware for the trained convolutional neural network for Arduino Nano33 Rev1 in Edge-ml

Overall, Edge-ML is still in its early stages of development. Although they claim the support of edge devices like Arduino Nano 33 BLE [58], the integration of the device along the whole development process is not yet supported. The dataset requires to be in a very specific format with timestamps which might suggest the framework is limited to time-series data. The framework does not provide any integrated tool to preprocess the data into their required format. Furthermore, the list of available features working as of now is very limited and with very abstract information about them, usually one-liner. With a primary focus on the edge AI development [58], the available options for the type of neural network for training for a C++ platform are limited to one, decision tree. Meanwhile, with every other classifier option neither the model nor the firmware could be downloaded for deployment. Additionally, neither of any available classifiers could be configured in terms of layers, activation functions, epoch, learning rate, batch size etc. Furthermore, when training the neural network the webpage crashed several times with errors like: "The requested URL could not be retrieved" and "This site can't be reached" and remained inactive for several days. Due to parts of the framework currently being in their beta and alpha phase of development [58], the framework is unstable and can't be used for scientific analysis.

3.3.4 TensorFlow Lite for Microcontrollers

After establishing the baseline Convolutional Neural Network (CNN) architecture and training the model as detailed in section 3.2, we proceeded to implement TensorFlow Lite

(TFLite) for Microcontrollers. Continuing our workflow in Google Colab, following the training of our baseline CNN model, the next step was to convert the TensorFlow (TF) model into the TFLite format. This conversion process is crucial for optimizing the model to run efficiently on microcontrollers. Before the conversion, it was essential to quantize the model reducing the model size and improving inference speed by converting the weights and activations from floating-point to integer values. or our implementation, we used full integer quantization, which involves converting all the weights and activations to 8-bit integers. To achieve full integer quantization, we first defined a representative dataset, which is a small sample of the training data used to calibrate the quantization process. This representative dataset ensures that the range of values in the training data is well-represented during quantization. In Google Colab, we loaded this dataset and applied the quantization operations, converting the model's weights and activations to integer format. Once the quantization was complete, we proceeded to convert the quantized model into the TFLite format using TensorFlow's TFLite Converter, a tool designed to transform TensorFlow models into an optimized .tflite version. The detailed steps and code for the conversion process are documented in the section A.

Once the model was generated, the easiest way to use a model on the microcontroller was to convert it into a C array [59] and compile it into a program using the Arduino IDE. For deploying it on the Arduino Nano 33 BLE, we first installed the TensorFlow Lite Micro Library for Arduino and used the inbuilt hello world example as a basis for development [60]. To facilitate the integration of the TensorFlow Lite model on an Arduino Nano 33 BLE board, we additionally included the LSM9DS1 sensor library. Within the loop function, data from the IMU was continuously read and then quantized and stored in the input tensor. The Invoke method of the interpreter is called to run the inference. The output from the model is then dequantized back to floating-point values, and the results are printed on the serial monitor. The loop continues to run, processing new IMU data and performing inferences indefinitely. The source files containing the model and the relevant files can be accessed here [61].

3.3.5 Edge Impulse

In our work with Edge Impulse, we began by uploading the raw dataset, utilizing the CSV wizard to read the files while excluding unnecessary data such as serial numbers. This preliminary step ensured that our dataset was clean and ready for further processing. The dataset was then randomly divided into test and training sets, providing a foundation for model training and validation. Following the dataset preparation, we configured the Impulse design to generate sliding windows, selecting a window size of 2500 ms and a window increase of 20 ms. This configuration allowed us to segment the data into manageable and consistent chunks for analysis. In the processing block, we opted for the raw data option, choosing to process the data without any additional preprocessing steps

following the guidelines from baseline architecture. This decision was based on the nature of our data and the requirements of our application.

In the learning block, we selected a classification algorithm designed to learn patterns from the data, which is particularly useful for movement classification tasks. Once the impulse design was complete, we generated features from the raw data and identified the importance of each feature, providing insights into which aspects of the data were most relevant for classification. The next phase involved designing the neural network architecture and specifying the training options as displayed in the section A. We maintained consistency with our baseline architecture to ensure comparability and reliability of results. Once the model was trained, we tested it on the test dataset to evaluate its performance and validate its accuracy.

In the final step, we built the trained model into an Arduino library, which was then downloaded for deployment. This integration was straightforward, allowing us to incorporate the library within the Arduino IDE seamlessly. The ease of use facilitated the deployment process, enabling us to perform inference directly on the Arduino hardware. This end-to-end workflow with Edge Impulse highlighted the platform's capabilities in handling data preprocessing, model training, and deployment efficiently. The Arduino sketch and the necessary files included in the library for the model generated can be found here [61].

3.3.6 NanoEdge AI Studio

For our work with NanoEdge AI Studio, the initial step involved converting the raw MotionSense dataset into the format required by the NanoEdge AI framework as shown by the code in the section A. This preprocessing ensured compatibility and smooth integration with the framework's system. Once the dataset was prepared, we uploaded 80% of it to the application, reserving the remaining 20% as the test dataset. Unlike some other frameworks, NanoEdge AI does not provide a built-in feature to split the dataset, so we performed this step manually to ensure proper training and testing. Before uploading the dataset, we created a project in NanoEdge AI Studio, selecting the target board and sensor type. For our project, we chose the Arduino Nano 33 BLE and the 3-axis accelerometer sensor. With these selections made, we proceeded to the signals section where the preprocessed dataset files were uploaded. After the upload was complete, we moved on to the Benchmark section. In the Benchmark section, we ran the benchmark process, during which NanoEdge AI Studio processed the data using an extensive array of libraries available on its servers. This process is crucial as it evaluates and selects the best-performing library based on criteria such as accuracy and memory consumption. Once the benchmarking was finished, the framework provided the optimal library for our specific application. Out of the available libraries we chose MLP (multi layer perceptron) to stick to our baseline architecture. Following the benchmark, we proceeded to the validation step. Here, we uploaded our reserved test dataset to assess the performance

of the selected library. This validation ensured that the library not only performed well during benchmarking but also maintained its accuracy and efficiency with new, unseen data. In the final step, we built the selected library and integrated it with the Arduino IDE using the provided Hello World example. The access to the sketch and the other necessary files required to run the inference can be found here [61].

3.3.7 Imagimob Studio

For our work with Imagimob Studio, the initial step was to create a new project within the platform. Following the project setup, we processed the raw dataset to ensure it met the specific data format requirements of Imagimob Studio as demonstrated by the code snippet in the section A. This preprocessing step was essential for smooth integration and accurate analysis within the framework. Once the entire dataset was uploaded, we redistributed it into validation, test, and training sets, ensuring a balanced and comprehensive dataset for model development. Next, we moved to the preprocessor section where we included a sliding window layer with the shape [125, 3] and a stride of 3. This configuration allowed us to generate appropriately sized input data windows for the neural network, enhancing the model's ability to learn temporal patterns from the sensor data.

In the training section, we began by creating an empty model. We then added layers to this model based on the baseline architecture defined in our previous work with Google Colab. By following the same network architecture and training configurations, we ensured consistency across different frameworks. The training process in Imagimob Studio is handled by cloud servers, providing robust computational resources and speeding up the training process. Once the training was completed, Imagimob Studio displayed a comprehensive view of test and validation accuracy scores, as well as the F1 scores for each label. These metrics were crucial for evaluating the model's performance and ensuring it met our accuracy requirements.

Following the successful training and evaluation, we utilized the code generation feature of Imagimob Studio to convert the trained model into C/C++ source files. Using the brief documentation provided by the Imagimob Studio Edge API, we integrated the generated source files into our Arduino sketch. Additionally including the LSM9DS1 sensor library data from the IMU was continuously read and the fed into the enqueue function responsible feeding the data into the model. Using the dequeue function we obtained the output and displayed the movement observed by the IMU signal on the Nano 33 BLE board. This final step allowed us to deploy and test the model on the Arduino Nano 33 BLE. The source file containing the model and the arduino sketch file performing inference are attached here [61].

Chapter 4

Evaluation

4.1 Quantitative Results

	Edge-Impulse (Enterprise)	TFLite for Microcon-trollers	NanoEdge AI Studio	Imagimob Studio
Accuracy	88.58%	97.35%	94.80%	96.52%
Memory Usage-RAM (KB)	50.41	70.82	48.40	96.32
Memory Usage-Flash (KB)	156.15	359.20	128.21	845.85
Inference Time (ms)	21	20	36	56
Current Consumption (mA)	7.95	7.98	8.03	8.01

TABLE 4.1: Resulting metrics from the implementation of the frameworks.

In this section, we present and analyze the results from four EdgeAI frameworks: Edge Impulse, TensorFlow Lite for Microcontrollers, NanoEdge AI Studio, and Imagimob Studio across four key metrics: accuracy, memory footprint (RAM and flash), inference time, and current consumption as shown in the table 4.1.

- **Accuracy**: The accuracy of the frameworks varied, with TFLM achieving the highest accuracy at 97.35%, followed by Imagimob Studio at 96.52%, NanoEdge AI at 94.80%, and lastly Edge Impulse at 88.58%. While higher accuracy generally indicates better performance, it is crucial to balance this with other factors such as memory usage, power consumption, and inference time.
- **Inference Time**: The inference time is a critical parameter for real-time applications. TFLM demonstrated the fastest inference time at 20 ms, a fraction faster than Edge Impulse at 21 ms, followed by NanoEdge AI at 36 ms and finally Imagimob Studio at 56 ms. Faster inference times enhance the responsiveness of the application.
- **Current Consumption**: Current consumption is a key factor when using battery-operated or solar-powered devices. In this regard, all the frameworks indicated nearly identical power consumption figures ranging from 7.95 mA to 8.03 mA. The board's power consumption can be reduced even further by removing the LED and unused peripherals [62].

- **Memory Footprint**: Arduino Nano 33 BLE has 1 MB of flash memory out of which 960 KB of flash is available, and 256 KB of RAM is available for development. Imagimob Studio has the largest flash usage at 845.85 KB, followed by TFLM at 359.20 KB, Edge Impulse at 156.15 KB, and the least by NanoEdge AI at 128.21 KB. As for the RAM usage, NanoEdge AI here too utilizes the least RAM at 48.40 KB, followed by Edge Impulse at 50.41 KB, TFLM at 70.82 KB, and Imagimob Studio with the most usage at 96.32 KB.

4.2 Experimental results of Usability

In this section, we examine the usability of the four EdgeAI frameworks: Edge Impulse, TensorFlow Lite for Microcontrollers, NanoEdge AI Studio, and Imagimob Studio based on the principles described in the section 2.3.

4.2.1 TensorFlow Lite for Microcontrollers

- **User Control and Freedom**: TFLM uses Python as the interface language and supports various IDEs like Visual Studio Code, PyCharm, and Jupyter Notebook. As a result, it offers complete control over the training process and neural network architecture. The framework allows for the manipulation of raw data into the required format using the third-party Pandas library. The resolution depends on the choice of IDE although providing significant flexibility, can be varied based on users.

- **Clarity**: The clarity of features in TFLM is largely dependent on the chosen IDE. While the flexibility of multiple IDE options can be beneficial, it may lead to a subjective experience that varies from user to user.

- **Help and Documentation**: TFLM provides extensive tutorials and projects, ranging from image classification and speech recognition to person detection. However, these tutorials often focus on a limited set of boards, such as the Raspberry Pi and Arduino Nano 33 BLE Sense. It provides documentation for performing inference via various platforms such as Linux, MacOS and Windows. The large TensorFlow community offers numerous additional examples and tutorials online, ensuring quick support and troubleshooting resources, although the response time of community support may vary.

- **Compatibility**: TFLM officially supports a wide range of microcontrollers with Arm Cortex-M and ESP32 architectures, ensuring broad hardware compatibility.

4.2.2 Edge Impulse

- **User Control and Freedom**: Edge Impulse also provides extensive control over the training process and the neural network architecture. However, to leverage

it, the user needs to switch to Keras expert mode in neural network settings. The framework also allows converting the raw data into the required format via the CSV Wizard feature. Being a web-based application, the resolution depends on the browser in use.

- **Clarity**: Edge Impulse offers a clear description of the features and options provided within the application. It offers step-by-step navigation through the application interface.
- **Help and Documentation**: It offers a wide range of end-to-end tutorials, from anomaly detection to image and object classification. These tutorials are varied across a number of boards from Arduino, Texas instruments and Nordic semiconductors. It provides documentation for inference on a wide range of MCUs across multiple options. The response time as observed from my post about the training-validation data split on the community forum of Edge Impulse is within two to three days.
- **Compatibility**: It provides a wide range of hardware compatibility ranging from an extensive list of microcontrollers to CPUs and GPUs.

4.2.3 NanoEdge AI Studio

- **User Control and Freedom**: NanoEdge AI Studio does not provide control over the training process and neither neural network architecture. It tests from a precompiled set of libraries on its servers. It offers partial support for converting raw data into the required format. The Data manipulation feature can perform simple tasks like removing rows and columns. While complex tasks like generating sliding windows from the dataset are not supported. The framework also does not allow any change in adjusting the size of the application according to the screen.
- **Clarity**: It offers limited clarity about the features and options within the application. Particularly in the benchmark section where no description of the library is provided other than the accuracy and memory usage.
- **Help and Documentation**: It offers only a handful of five to six tutorials on the current sensing classifier, multi-state vibration classifier, multi-frequency classifier and anomaly detection. These examples are practised mainly on STM32 microcontrollers. Documentation is provided for compiling and generating the ML library, however, no information is provided on performing inference. The community forum seems to be extensive although no response was observed from my post about the deployment difficulties.
- **Compatibility**: The framework is compatible with a wide range of development boards, MCUs, and Arduino Boards.

4.2.4 Imagimob Studio

- **User Control and Freedom**: Imagimob Studio provides control to manipulate the training process and neural network architecture. Although, It does not provide a tool to convert the dataset into its required format. The resolution of the application is configurable and can resized according to preferences.
- **Clarity**: It offers limited clarity about the features and the options available within the application. In the case of data upload and management, the procedure to select the data and label files separately is not clearly described.
- **Help and Documentation**: The framework offers only two tutorials or examples, namely human-activity detection and siren detection. The tutorials are practised on only one development kit, the Infineon PSoC™6 development kit. A brief documentation is provided on performing inference. The community forum has very limited activity and no response was observed on my error about deployment.
- **Compatibility**: In terms of board compatibility, it supports any development kit or board for model evaluation.

Chapter 5

Conclusion

In this work, we wanted to examine and compare the performances of nine EdgeAI frameworks: Edge Impulse, TensorFlow Lite for Microcontrollers, Mathworks, Edge ML, ELL Microsoft, Arm CMSIS-NN, uTensor, NanoEdge AI Studio, and Imagimob Studio across five metrics: accuracy, inference time, memory consumption, current consumption, and usability on the Arduino Nano 33 BLE board. Out of the nine mentioned frameworks, two encountered persistent errors, uTesnor and Mathworks, which could not be resolved within the given time constraints. Given the availability of other frameworks that required our attention, it was not feasible to allocate further time and so we moved on to the other remaining frameworks. One other framework, Edge-ML could not produce desired results. It is still in its early stages of development and as of now is unstable in model training and deployment. Two more frameworks, Arm CMSIS-NN and ELL Microsoft were beyond the scope of the project's timeframe as anticipated in the beginning of the thesis.

The four remaining frameworks were evaluated for bringing the neural networks on resource-constrained microcontroller devices. Out of them, TFLM performed the best in terms of accuracy and inference time. While NanoEdge AI Studio showed the minimum overall memory usage, flash and RAM combined. In terms of current consumption, all of the frameworks showed nearly identical figures of approximately 8 mA, suggesting a constant active state architecture of the microcontroller. From the usability perspective, TFLM excels in user control & freedom, and help & documentation principles. While Edge Impulse excels at clarity, compatibility, and help & documentation. As different frameworks excel at different metrics, the selection of the EdgeAI framework for a specific application should be guided by the priority of the metrics for that application. In the case of railway safety applications, the metrics of priority are inference time, accuracy, and power consumption. Based on our results, for this specific use case, the TFLM could be the best choice as an EdgeAI framework for development.

Chapter 6

Future Work

The aim of this thesis was to evaluate the relative differences between various EdgeAI frameworks in terms of accuracy, inference time, memory consumption, current consumption, and usability. Based on our findings, several avenues for future work can be identified.

Firstly, the neural network architectures used in this evaluation can be further refined and adapted to better suit specific application requirements. Experimenting with different configurations can optimize the models, and can improve the accuracy. Additionally, power consumption can be further minimized by implementing sleep and deep sleep modes, as well as removing unused peripherals and hardware components. This will help in making the frameworks more energy-efficient. While this study focused on the Arduino Nano 33 BLE board, future benchmarking could be conducted on a variety of hardware platforms to provide a more comprehensive evaluation of the frameworks. This will help identify platform-specific optimizations and ensure broader applicability. Expanding the range of evaluation metrics could be another factor; incorporating additional metrics such as F1 scores for each label and performing live classification tests can provide deeper insights into the model's performance before and after deployment. By addressing these areas, we can significantly improve the utility and efficiency of EdgeAI frameworks, ensuring they meet the diverse needs of different applications and hardware environments.

Appendix A

Appendix

```
import numpy as np
import pandas as pd
import os
import ast

labels_predefined = {'WLK':0, 'JOG':1, 'SIT':2, 'UPS':3, 'DWS':4, 'STD':5}

#function for creating and storing sliding windows
def rolling_agg_list(src_frame, cols, window):
  frame = src_frame[cols]
  d = {v: list for v in cols}
  values = [frame.iloc[i:i + window].agg(d)for i in range(len(frame) - window
      + 1)]
  return pd.DataFrame(values, columns=cols)


# Function to load and preprocess data from a single CSV file
def load_and_preprocess_data(df):

  # Assuming each 'x', 'y', 'z' column has lists of varying lengths
  x_lengths = df['x'].apply(len)
  y_lengths = df['y'].apply(len)
  z_lengths = df['z'].apply(len)

  # Find the maximum length among 'x', 'y', and 'z' arrays
  max_length = max(x_lengths.max(), y_lengths.max(), z_lengths.max())

  # Pad arrays to the maximum length
  df['x'] = df['x'].apply(lambda x: np.pad(x, (0, max_length - len(x))))
  df['y'] = df['y'].apply(lambda y: np.pad(y, (0, max_length - len(y))))
  df['z'] = df['z'].apply(lambda z: np.pad(z, (0, max_length - len(z))))

  # Combine 'x', 'y', and 'z' arrays into a 3D array
  xt = np.array(df['x'].tolist())
```

```
  yt = np.array(df['y'].tolist())
  zt = np.array(df['z'].tolist())
  stacked_data = np.stack((xt, yt, zt), axis=2)

  accelerometer_data = np.array([df['x'].tolist(), df['y'].tolist(),
      df['z'].tolist()]).T

  df['label_numeric'] = df['label'].apply(lambda x:labels_predefined[x])
  labels = df['label_numeric'].values

  return stacked_data, labels

# Directory containing CSV files
data_directory = '/content/drive/MyDrive/B_Accelerometer_data/Raw/zTest/wlk'

# List to store data from all files


# Load and preprocess data from each CSV file
for index, filename in enumerate(os.listdir(data_directory)):
  if filename.endswith(".csv"):
    file_path = os.path.join(data_directory, filename)
    df = pd.read_csv(file_path)
    # x = len(df)*0.80
    X_ttemp = df#[: round(x)]
    # X_vtemp = df[round(x) :]

    window_train = rolling_agg_list(X_ttemp, ['x','y','z'], 125)
    window_train.insert(3, 'label', 'WLK')
    # window_val = rolling_agg_list(X_vtemp, ['x','y','z'], 125)
    # window_val.insert(3, 'label', 'DWS')

    accelerometer_data_train, labels_train =
        load_and_preprocess_data(window_train)
    # accelerometer_data_val, labels_val = load_and_preprocess_data(window_val)

    print(accelerometer_data_train.shape, labels_train.shape)
    # print(accelerometer_data_val.shape, labels_val.shape)
    if index==0:
      all_data_train = accelerometer_data_train
      all_labels_train = labels_train
      # all_data_val = accelerometer_data_val
      # all_labels_val = labels_val
    else:
      all_data_train = np.concatenate((all_data_train,
          accelerometer_data_train), axis = 0)
```

```
      all_labels_train = np.concatenate((all_labels_train, labels_train))
      # all_data_val = np.concatenate((all_data_val, accelerometer_data_val),
          axis = 0)
      # all_labels_val = np.concatenate((all_labels_val, labels_val))

    print('Aat iteration {} the shape of temporary TRAINING xy is {} and {}
        and final TRAINING shape is {}'.format(index,
        accelerometer_data_train.shape, labels_train.shape,
        all_data_train.shape, all_labels_train.shape))
    # print('Aat iteration {} the shape of temporary VALIDATION xy is {} and
        {} and final VALIDATION shape is {}'.format(index,
        accelerometer_data_val.shape, labels_val.shape, all_data_val.shape,
        all_labels_val.shape))
```

LISTING A.1: Generating sliding windows in Google Colab

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
    Reshape

model = Sequential()
# Adjusting the input shape to work with Conv2D
model.add(Reshape((125, 3, 1), input_shape=(125, 3))) # Adding a
    single-channel dimension
model.add(Conv2D(16, kernel_size=(3, 3), activation='relu', padding='SAME',
    data_format='channels_last'))
model.add(MaxPooling2D(pool_size=(3, 3)))
model.add(Flatten())
#model.add(Reshape((-1, )))
model.add(Dense(32, activation='relu'))
model.add(Dense(6, activation='softmax')) # Assume 6 is the number of your
    classes

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])
model.summary()

model.fit(X_train, y_train, epochs=4, batch_size=32, validation_data=(X_val,
    y_val))
```

LISTING A.2: CNN architecture using keras in Google Colab

```
# Define a function that generates representative input data
def representative_dataset():
  for i in range(100): # Generate 100 samples
    data = X_val[i] # Extract the data
```

```
    data = data.astype(np.float32).reshape((1, 125, 3, 1)) # Reshape and
        type-cast
    yield [data]

import numpy as np
import tensorflow as tf

# Set up the converter for full-integer quantization
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
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 the model
tflite_int8_model = converter.convert()

# Save the converted model
with open('tflite_int8_model(FullDataset).tflite', 'wb') as f:
  f.write(tflite_int8_model)
```

LISTING A.3: Generating TFLite model with Full Integer quantization

```
import os
import pandas as pd

# Path to the directory containing your CSV files
input_directory = '/content/drive/MyDrive/B_Accelerometer_data/Raw/wlk_15'
output_directory = '/content/drive/MyDrive/Imagimob data
    preprocessing/Data/WLK'

# Create output directory if it does not exist
os.makedirs(output_directory, exist_ok=True)

# Function to convert serial numbers to time
def convert_serial_to_time(input_csv, output_csv, interval_ms=20):
  # Read the CSV into a DataFrame
  df = pd.read_csv(input_csv)

  # Calculate the interval in seconds
  interval_s = interval_ms / 1000.0

  # Generate time column based on the number of rows and interval
  time_column = [i * interval_s for i in range(len(df))]
```

```
    # Replace the first column with the time column
    df.iloc[:, 0] = time_column
    df.columns = ['Time (s)', 'Accelerometer X', 'Accelerometer Y',
        'Accelerometer Z']

    # Write the modified DataFrame back to a new CSV file
    df.to_csv(output_csv, index=False)

# Process all CSV files in the directory
for filename in os.listdir(input_directory):
    if filename.endswith('.csv'):
        input_csv = os.path.join(input_directory, filename)
        output_csv = os.path.join(output_directory, filename)
        convert_serial_to_time(input_csv, output_csv)

print("Conversion completed.")

# Path to the directory containing your modified CSV files
input_directory = '/content/drive/MyDrive/Imagimob data preprocessing/Data/WLK'
label_directory = '/content/drive/MyDrive/Imagimob data
    preprocessing/Label/WLK'
label_extension = '.label'

# Create label directory if it does not exist
os.makedirs(label_directory, exist_ok=True)

# Function to create a label file corresponding to the data file
def create_label_file(data_csv, label_file, label_name, interval_ms=20):
    # Read the CSV into a DataFrame
    df = pd.read_csv(data_csv)

    # Calculate the total recording time in seconds
    interval_s = interval_ms / 1000.0
    total_time = (len(df) - 1) * interval_s # Exclude the header row

    # Set the starting time, length, label, confidence, and comment
    start_time = 0.0
    length = total_time
    confidence = 1.0
    comment = "No Comment"

    # Create the label DataFrame
    label_data = {
        'Time': [f'{start_time:.3f}'],
        'Length': [f'{length:.3f}'],
        'Label': [label_name],
```

```
    'Confidence': [f'{confidence:.1f}'],
    'Comment': [comment]
  }
  label_df = pd.DataFrame(label_data)

  # Write the label DataFrame to a .label file
  label_df.to_csv(label_file, index=False, encoding='utf-8')

# Process all CSV files in the directory
for filename in os.listdir(input_directory):
  if filename.endswith('.csv'):
    data_csv = os.path.join(input_directory, filename)
    label_file = os.path.join(label_directory, os.path.splitext(filename)[0] +
        label_extension)

    # You can customize this label name as per your requirements
    label_name = 'Walking'

    create_label_file(data_csv, label_file, label_name)

print("Label creation completed.")
```

LISTING A.4: Data preprocessing to meet data format requirements of Imagimob Studio

```
import os
import pandas as pd
import numpy as np

# Define the path to the folder containing the CSV files
input_folder_path =
    '/content/drive/MyDrive/B_Accelerometer_data/Raw/zTest/wlk' # change to
    your folder path
output_folder_path = '/content/drive/MyDrive/B_Accelerometer_data/Nano EdgeAI
    Studio/ztest/wlk' # change to your output folder path

# Step 2: Load and process each CSV file
for filename in os.listdir(input_folder_path):
  if filename.endswith('.csv'):
    # Load the CSV file
    input_file_path = os.path.join(input_folder_path, filename)
    df = pd.read_csv(input_file_path)

    # Step 3: Remove the first column (assumed to be the serial number)
    df = df.iloc[:, 1:].reset_index(drop=True)
```

```
    # Ensure the DataFrame has the expected number of columns
    if df.shape[1] != 3:
      print(f"Unexpected number of columns in file {filename}. Skipping.")
      continue

    processed_data = []

    # Step 4: Generate sliding windows of 125 data points with stride 1
    for i in range(len(df) - 124): # The last index for a full window
      window = df.iloc[i:i+125].values.flatten()
      if len(window) == 375: # Ensure the window is of the correct length
        processed_data.append(window)

    # Convert the list of processed data into a DataFrame
    processed_df = pd.DataFrame(processed_data)

    # Save the processed data to a new CSV file with the same name as the
        original
    output_file_path = os.path.join(output_folder_path, filename)
    processed_df.to_csv(output_file_path, index=False, header=False)

from random import shuffle
# Define the path to the folder containing the windowed CSV files
windowed_folder_path = '/content/drive/MyDrive/B_Accelerometer_data/Nano
    EdgeAI Studio/ztest/wlk' # change to your folder path
output_file_path = '/content/drive/MyDrive/B_Accelerometer_data/Nano EdgeAI
    Studio/ztest/WLK_fullWdata.csv' # change to your desired output path

# Step 2: Read and concatenate all windowed CSV files
all_windowed_data = []

for filename in os.listdir(windowed_folder_path):
  if filename.endswith('.csv'):
    # Load the windowed CSV file
    windowed_file_path = os.path.join(windowed_folder_path, filename)
    df = pd.read_csv(windowed_file_path, header=None)

    # Append the DataFrame to the list
    all_windowed_data.append(df)

# Step 3: Concatenate all DataFrames
concatenated_df = pd.concat(all_windowed_data, ignore_index=True)

# Step 4: Shuffle the rows of the concatenated DataFrame
shuffled_df = concatenated_df.sample(frac=1).reset_index(drop=True)
```

```
# concatenated_df = dummy[idx]

# Step 4: Save the concatenated DataFrame to a single CSV file
concatenated_df.to_csv(output_file_path, index=False, header=False)
```

LISTING A.5: Data preprocessing to meet data format requirements of NanoEdge AI Studio

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, InputLayer, Dropout, Conv1D,
    Conv2D, Flatten, Reshape, MaxPooling1D, MaxPooling2D, AveragePooling2D,
    BatchNormalization, Permute, ReLU, Softmax
from tensorflow.keras.optimizers.legacy import Adam

EPOCHS = args.epochs or 4
LEARNING_RATE = args.learning_rate or 0.001
# If True, non-deterministic functions (e.g. shuffling batches) are not used.
# This is False by default.
ENSURE_DETERMINISM = args.ensure_determinism
# this controls the batch size, or you can manipulate the tf.data.Dataset
    objects yourself
BATCH_SIZE = args.batch_size or 32
if not ENSURE_DETERMINISM:
  train_dataset = train_dataset.shuffle(buffer_size=BATCH_SIZE*4)
train_dataset=train_dataset.batch(BATCH_SIZE, drop_remainder=False)
validation_dataset = validation_dataset.batch(BATCH_SIZE, drop_remainder=False)

# model architecture
model = Sequential()
model.add(Reshape((125, 3, 1), input_shape=(None, 375)))
model.add(Conv2D(16, kernel_size=3,
    kernel_constraint=tf.keras.constraints.MaxNorm(1), padding='same',
    activation='relu'))
model.add(MaxPooling2D(pool_size=3, strides=3, padding='same'))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(classes, name='y_pred', activation='softmax'))

# this controls the learning rate
opt = Adam(learning_rate=LEARNING_RATE, beta_1=0.9, beta_2=0.999)
callbacks.append(BatchLoggerCallback(BATCH_SIZE, train_sample_count,
    epochs=EPOCHS, ensure_determinism=ENSURE_DETERMINISM))

# train the neural network
```

```
model.compile(loss='categorical_crossentropy', optimizer=opt,
    metrics=['accuracy'])
model.fit(train_dataset, epochs=EPOCHS, validation_data=validation_dataset,
    verbose=2, callbacks=callbacks)

# Use this flag to disable per-channel quantization for a model.
# This can reduce RAM usage for convolutional models, but may have
# an impact on accuracy.
disable_per_channel_quantization = False
```

LISTING A.6: CNN classifier architecture of Edge Impulse

Bibliography

[1] R. Singh and S. S. Gill, "Edge ai: A survey," *Internet of Things and Cyber-Physical Systems*, vol. 3, pp. 71–92, 2023, ISSN: 2667-3452. DOI: https://doi.org/10.1016/j.iotcps.2023.02.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2667345223000196.

[2] M. M. H. Shuvo, S. Islam, J. Cheng, and B. Morshed, "Efficient acceleration of deep learning inference on resource-constrained edge devices: A review," *Proceedings of the IEEE*, vol. 111, pp. 1–50, Dec. 2022. DOI: 10.1109/JPROC.2022.3226481.

[3] S. Shajari, K. Kuruvinashetti, A. Komeili, and U. Sundararaj, "The emergence of ai-based wearable sensors for digital health technology: A review," *Sensors*, vol. 23, no. 23, 2023, ISSN: 1424-8220. DOI: 10.3390/s23239498. [Online]. Available: https://www.mdpi.com/1424-8220/23/23/9498.

[4] P. E. David, P. R. Chelliah, and P. Anandhakumar, "Chapter nine - reshaping agriculture using intelligent edge computing," in *Applying Computational Intelligence for Social Good*, ser. Advances in Computers, P. E. David and P. Anandhakumar, Eds., vol. 132, Elsevier, 2024, pp. 167–204. DOI: https://doi.org/10.1016/bs.adcom.2023.08.007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0065245823000736.

[5] K. Team, *Keras documentation: About Keras 3 — keras.io*, https://keras.io/about/, [Accessed 20-05-2024].

[6] *Arxiv.org*, https://arxiv.org/pdf/1605.08695, [Accessed 20-05-2024].

[7] *XLA - TensorFlow, compiled — developers.googleblog.com*, https://developers.googleblog.com/en/xla-tensorflow-compiled/, [Accessed 17-05-2024].

[8] *Jmlr.org*, https://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf, [Accessed 20-05-2024].

[9] *Colab.google — colab.google*, https://colab.google/, [Accessed 28-05-2024].

[10] *Research technology service | University of St Andrews — st-andrews.ac.uk*, https://www.st-andrews.ac.uk/research/support/research-technology-service/interactive-computing-environment-jupyter/70332, [Accessed 28-05-2024].

[11] Melanie, *Google Colab: The power of the cloud for machine learning — datascientest.com*, https://datascientest.com/en/google-colab-the-power-of-the-cloud-for-machine-learning, [Accessed 28-05-2024].

[12] *Datasets – Google Research — research.google*, https://research.google/resources/datasets/, [Accessed 28-05-2024].

[13] *Ijset.in*, https://www.ijset.in/wp-content/uploads/IJSET_V12_issue1_587.pdf, [Accessed 28-05-2024].

[14] A. Osman, U. Abid, L. Gemma, M. Perotto, and D. Brunelli, "Tinyml platforms benchmarking," in *Applications in Electronics Pervading Industry, Environment and Society*, S. Saponara and A. De Gloria, Eds., Cham: Springer International Publishing, 2022, pp. 139–148, ISBN: 978-3-030-95498-7.

[15] P. P. Ray, "A review on tinyml: State-of-the-art and prospects," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 4, pp. 1595–1623, 2022, ISSN: 1319-1578. DOI: https://doi.org/10.1016/j.jksuci.2021.11.019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1319157821003335.

[16] *Data acquisition | Edge Impulse Documentation — edge-impulse.gitbook.io*, https://edge-impulse.gitbook.io/docs/edge-impulse-studio/data-acquisition, [Accessed 10-04-2024].

[17] C. Siegl, *TensorFlow Lite for Microcontrollers adds Support for Efficient LSTM Implementation — christoph-siegl*, https://medium.com/@christoph-siegl/tensorflow-lite-for-microcontrollers-adds-support-for-efficient-lstm-implementation-25a5f7baa4f6, [Accessed 19-05-2024].

[18] T. Team, *Tensorflow lite for microcontrollers*, https://github.com/tensorflow/tflite-micro/blob/main/tensorflow/lite/micro/micro_mutable_op_resolver.h, 2023.

[19] *TensorFlow Lite for Microcontrollers - Experiments with Google — experiments.withgoogle.com*, https://experiments.withgoogle.com/collection/tfliteformicrocontrollers, [Accessed 11-04-2024].

[20] *Mathworks.com*, https://www.mathworks.com/content/dam/mathworks/fact-sheet/2023-company-factsheet-8-5x11-8282v23.pdf, [Accessed 19-05-2024].

[21] *What Is MATLAB? — de.mathworks.com*, https://de.mathworks.com/discovery/what-is-matlab.html, [Accessed 11-04-2024].

[22] *Simulink Coder Documentation - MathWorks Deutschland — de.mathworks.com*, https://de.mathworks.com/help/rtw/, [Accessed 14-04-2024].

[23] EdgeML Team, *EdgeML-Arduino: An open-source toolchain for deploying machine learning models on microcontrollers*, https://github.com/edge-ml/EdgeML-Arduino, Accessed: 2024-04-12, 2024.

[24] *AI:NanoEdge AI Studio - stm32mcu — wiki.st.com*, https://wiki.st.com/stm32mcu/wiki/AI:NanoEdge_AI_Studio, [Accessed 14-04-2024].

[25] *Imagimob Studio documentation — developer.imagimob.com*, https://developer.imagimob.com/getting-started/graph-ux-concept, [Accessed 20-05-2024].

[26] Imagimob, *Collect data using graph ux*, https://developer.imagimob.com/data-collection/collect-data-using-graph-ux, Accessed: 2024-04-12, 2024.

[27] *Bitbucket — bitbucket.org*, https://bitbucket.org/imagimob/captureserver/src/master/, [Accessed 14-04-2024].

[28] *Imagimob Studio documentation — developer.imagimob.com*, https://developer.imagimob.com/edge-optimization/edge-deployment, [Accessed 14-04-2024].

[29] *Imagimob Studio documentation — developer.imagimob.com*, https://developer.imagimob.com/edge-optimization/edge-api, [Accessed 20-05-2024].

[30] *GitHub - uTensor/uTensor: TinyML AI inference library — github.com*, https://github.com/uTensor/uTensor?tab=readme-ov-file, [Accessed 21-05-2024].

[31] *UTensor/src/uTensor/README.md at master · uTensor/uTensor — github.com*, https://github.com/uTensor/uTensor/blob/master/src/uTensor/README.md, [Accessed 21-05-2024].

[32] *Utensorcgen: C/C++ code generator for uTensor 2014; utensorcgen documentation — utensor-cgen.readthedocs.io*, https://utensor-cgen.readthedocs.io/en/latest/, [Accessed 21-05-2024].

[33] *GitHub - ARM-software/CMSIS-NN: CMSIS-NN Library — github.com*, https://github.com/ARM-software/CMSIS-NN, [Accessed 24-05-2024].

[34] *CMSIS-NN/Tests/UnitTest/README.md at main · ARM-software/CMSIS-NN — github.com*, https://github.com/ARM-software/CMSIS-NN/blob/main/Tests/UnitTest/README.md, [Accessed 24-05-2024].

[35] *The Embedded Learning Library - Embedded Learning Library (ELL) — microsoft.github.io*, https://microsoft.github.io/ELL/, [Accessed 24-05-2024].

[36] *ELL Tutorials - Embedded Learning Library (ELL) — microsoft.github.io*, https://microsoft.github.io/ELL/tutorials/, [Accessed 24-05-2024].

[37] *ELL/INSTALL-Windows.md at master · microsoft/ELL — github.com*, https://github.com/microsoft/ELL/blob/master/INSTALL-Windows.md, [Accessed 24-05-2024].

[38] *ELL/INSTALL-Advanced.md at master · microsoft/ELL — github.com*, https://github.com/microsoft/ELL/blob/master/INSTALL-Advanced.md, [Accessed 24-05-2024].

[39] *GitHub - ripred/Profiler: Easily profile your Arduino functions to see how much time they take. The output can be disabled and enabled at runtime. Very lightweight. — github.com*, https://github.com/ripred/Profiler, [Accessed 31-05-2024].

[40] *Usability Principles — improvement.stanford.edu*, https://improvement.stanford.edu/resources/usability-principles, [Accessed 25-05-2024].

[41] *Help and Documentation (Usability Heuristic 10) — nngroup.com*, https://www.nngroup.com/articles/help-and-documentation/, [Accessed 25-05-2024].

[42] *ISO 25010 — iso25000.com*, https://iso25000.com/index.php/en/iso-25000-standards/iso-25010, [Accessed 25-05-2024].

[43] R. Tilly, P. Neumaier, K. Schwalbe, *et al.*, *Open sensor data for rail 2023*, de, 2023. DOI: 10.57806/9MV146R0. [Online]. Available: https://data.fid-move.de/dataset/3d7e7406-639f-49f6-bbca-caac511b4032.

[44] *GitHub - mmalekzadeh/motion-sense: MotionSense Dataset for Human Activity and Attribute Recognition (time-series data generated by smartphone's sensors: Accelerometer and gyroscope) (PMC Journal) (IoTDI'19) — github.com*, https://github.com/mmalekzadeh/motion-sense, [Accessed 14-04-2024].

[45] *Getting raw accelerometer events | Apple Developer Documentation — developer.apple.com*, https://developer.apple.com/documentation/coremotion/getting_raw_accelerometer_events, [Accessed 14-04-2024].

[46] *Sliding Window and Two Pointer Technique — linkedin.com*, https://www.linkedin.com/pulse/sliding-window-two-pointer-technique-saiful-islam-rasel, [Accessed 14-04-2024].

[47] *Impact of Sliding Window Length in Indoor Human Motion Modes and Pose Pattern Recognition Based on Smartphone Sensors — ncbi.nlm.nih.gov*, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6021910/, [Accessed 14-04-2024].

[48] *Overview of temporal action detection based on deep learning - Artificial Intelligence Review — link.springer.com*, https://link.springer.com/article/10.1007/s10462-023-10650-w, [Accessed 20-05-2024].

[49] *Arduino nano 33 ble docs*, https://docs.arduino.cc/hardware/nano-33-ble/, [Accessed 21-05-2024].

[50] *Introduction - Introduction to Mbed OS 6 | Mbed OS 6 Documentation — os.mbed.com*, https://os.mbed.com/docs/mbed-os/v6.16/introduction/index.html, [Accessed 22-05-2024].

[51] N. Vishwakarma, *What is Adam Optimizer? — analyticsvidhya.com*, https://www.analyticsvidhya.com/blog/2023/09/what-is-adam-optimizer/, [Accessed 01-06-2024].

[52] S. Chand, *Choosing between Cross Entropy and Sparse Cross Entropy — The Only Guide you Need! — shireenchand*, https://medium.com/@shireenchand/choosing-between-cross-entropy-and-sparse-cross-entropy-the-only-guide-you-need-abea92c84662, [Accessed 01-06-2024].

[53] *(Not recommended) Train neural network - MATLAB trainNetwork - MathWorks Deutschland — de.mathworks.com*, https://de.mathworks.com/help/deeplearning/ref/trainnetwork.html, [Accessed 25-05-2024].

[54] *Read one acceleration data sample from LSM9DS1 sensor - MATLAB readAcceleration - MathWorks Deutschland — de.mathworks.com*, https://de.mathworks.com/help/matlab/supportpkg/lsm9ds1.readacceleration.html, [Accessed 25-05-2024].

[55] *How do I perform a clean installation of MATLAB? — de.mathworks.com*, https://de.mathworks.com/matlabcentral/answers/131519-how-do-i-perform-a-clean-installation-of-matlab, [Accessed 27-04-2024].

[56] *MATLAB Coder — mathworks.com*, https://www.mathworks.com/products/matlab-coder.html, [Accessed 28-04-2024].

[57] *Home — github.com*, https://github.com/edge-ml/edge-ml/wiki/, [Accessed 23-05-2024].

[58] *Edge-ml: E2e machine learning on the edge — edge-ml.org*, https://edge-ml.org/, [Accessed 23-05-2024].

[59] *Build and convert models | TensorFlow Lite — tensorflow.org*, https://www.tensorflow.org/lite/microcontrollers/build_convert, [Accessed 01-06-2024].

[60] *GitHub - tensorflow/tflite-micro-arduino-examples — github.com*, https://github.com/tensorflow/tflite-micro-arduino-examples, [Accessed 01-06-2024].

[61] *Benchmarking EdgeAI frameworks using Nano 33 BLE - Google Drive — drive.google.com*, https://drive.google.com/drive/folders/1TpNfgIISODVHl5J83mI8PriDaEzISCP7?usp=sharing, [Accessed 01-06-2024].

[62] *Nano33 ble - how should I go about reducing current draw — forum.arduino.cc*, https://forum.arduino.cc/t/nano33-ble-how-should-i-go-about-reducing-current-draw/637270, [Accessed 01-06-2024].