TinyAED: Real Time Fall Detection Alert System through Audio Classification using IoT & Tiny ML

Michael Peres
200362146
Dr Stephen Poslad
MSc Artificial Intelligence

***Abstract*— Provides an in-expensive, privacy orientated exploration into the feasibility of fall detection using audio classification on TinyML devices. It aims to provide an extensive view of two different machine learning techniques, CNN, and RNNs which run locally without the need for the cloud. As well as providing a custom dataset created by the author of over 30 minutes of fall and non-fall audio. The solution provided, allows for alerting care gives about falls via Serial, without the need for Wi-Fi access points. The papers' author achieved accuracies for both models' recognition accuracy between 93-95%.**

***Keywords*— TinyML, Alert System, Audio Classification, Edge AI, IoT, TensorFlow Lite Micro, Audio Augmentation, Fall Detection, RNNs, LSTM, CNN, Transformers, Tiny ViT**

I. Introduction

Human falls account for 684,000 deaths annually (World Health Organization, 2021)**,** with the majority of individuals within 60 years of age, it should be evident that the elderly are at most risk of this unfortunate event, and such alert systems should be in place to prevent the absence of aid when this event occurs automatically. This paper delves into a solution based on IoT devices, TinyML and audio classification.

In recent years, the use of tiny devices such as Internet of Things (IoT) and microcontrollers has increasingly been adopted, due to its cost-effectiveness, energy reductions and privacy-oriented nature. Compared to the hub and spoke architecture, current trends incentives transition of data processing closer to the data source via inexpensive IoT device rather than bulky datacentres (Singh et al., 2020). This has brought around advantages such as reduced latency and lower strain on networks, and ever more increasingly, a methodology for improved privacy of raw data by removing the need for upload to the cloud, (Wang, Ellul and Azzopardi, 2020). In addition, an increased availability for IoT devices expected to reach 30 billion by 2030, (Vailshery, 2023) this showcases the need for methods in alleviating the traffic flow of sensory data to the cloud and exploiting such smart devices to perform in useful processing of sensitive data in an ever growing privacy conscious society. The pivot to edge devices providing local processing facilities also aids in reducing the high energy consumption of training and operating large ML models in the cloud.

IoT devices have been successful in fields of wearable devices in tracking our health goals, and different industrial and commercial settings, such as in automated HVAC systems based on number of individuals detected in a

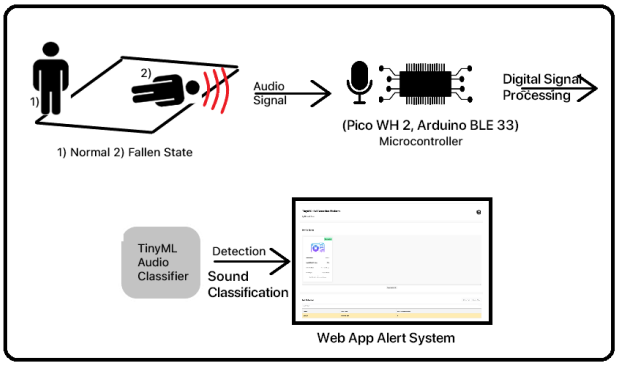


Figure 1 Architecture of Alert System, two different microcontrollers are used for detecting audio signals. Classifications are sent to a Python Webserver via serial connection which are then presented via to the VueJS web client.

building (Zacharia et al., 2022), or in healthcare for determining whether an individual has diabetes via a non-invasive human breath test (Gudiño-Ochoa et al., 2024)**.** Although these solutions have been excellent for bringing quality of life improvements, they often lead to overlooking the challenges of running programs on these tiny devices. These devices are severely limited in memory and processing power, with some microcontrollers lacking floating point arithmetic, leading to the need of redesigning current solutions for this new complex environment. Typical IoT resource constraint ranges are shown in **Table VIII of Appendix**.

The field of Tiny Machine Learning (TinyML) explores the concept of utilising machine learning algorithms on edge devices (consuming power in the magnitude of mW) (Warden and Situnayake, 2019). This challenging field is exacerbated by the addition that conventional algorithms have traditionally ran on datacentres and PCs, rebranded to run on edge devices. In light of this demanding task, continuous research has been conducted to provide algorithms and methodologies that utilise less computational resources while achieving promising results on accuracy for various tasks. TinyML techniques include pruning (Liu et al., 2020) (removal of weights), quantization (Jacob et al., 2017) (scaling of data types) and clustering (ARM, 2023) (grouping weights into clusters), which are expanded on in section III of this paper. These techniques are crucial to edge devices as they minimise the model and reduce the computational boundary for execution of ML models on memory and processor constrained devices while maintaining sufficient accuracy and temporal performance.

Currently there exists multiple research efforts in detecting human falls, this includes wearable devices with the incorporation of acceleration data (Santos et al., 2019) and commercial options such as the Apple Watch (Apple, 2019) or Samsung Smart Watch (Samsung, n.d.)**,** which have some capacity for emergency alerting. However, commercial solutions fail to provide an autonomous privacy orientated system, often relying on users' inputs for verification and an internet connection. Furthermore, very few works have covered the usage of audio-based fall detection classification in the context of TinyML.

This paper situates itself within the fields of domestic environment sound detection and audio-based alert systems for edge devices, especially in detecting human falls in elderly people.

II. Project Aims

The paper aims to provide a methodology for an audio classifier alert system, that can detect falls of elderly individuals based on purely audio signals. The aim is to assess the feasibility of local classification of audio signals, through the measurable evaluation of the model's accuracy and latency in scenarios where network connectivity is not present. The project aims to provide an inclusive evaluation of audio classification pipeline by utilising two different methodologies, one with the Edge Impulse platform and the other using TensorFlow Lite and Arduino libraries, with two different devices, the Pico WH and the Arduino BLE33 Sense Rev 2. This work will utilise inexpensive components to allow for easy reproducibility of work and allow further development and research. The proposed solution can be utilised in locations with a limited network infrastructure, such as in the least developed countries, for alerting authorities or guardians. The only requirement is to ensure there is a serial connection to the edge device. The paper aims to implement different ML solutions based on literature including Convolution Neural Networks (CNNs) (O'Shea and Nash, 2015) and Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997), where evaluation metrics such as accuracy, losses, and failed tests are reviewed, explaining possible improvements. With the expansion of these methodologies, the author aims to provide an extensive feasibility review into audio classification on TinyML devices. The proposed architecture of the solution is shown in **Figure 1**.

The solution is tested on a benchmark compromised of the authors' samples as well as A3FALL (Principi, 2015) dataset. Further analysis of models CNN and LTSM showcase feasibility with current microcontrollers in metrics including latency, and memory usage.

The remaining parts of the paper are presented as follows. Section III describes literature review, and aiding in the understanding and logic applied in later sections, it aims to provide the reader with the necessary information, Section IV proposes methodology of the solution, including CNN and RNN architectures as well as intermediate results, Section V details the testing and evaluation stage explaining how the author evaluated their results in comparison with other models and methods mentioned in Section IV, Section VI draws on the results of the method, including specifics of the models output. Section VII showcases a conclusion, summarising the papers value added based on project aims, addition to the field and future research opportunities.

III. Literature Work

This section will cover literature surrounding the papers' aims, including different techniques of fall detection, how machine learning solutions have been applied to solve these specific problems, how TinyML can help implement these solutions and optimisations required for running on edge devices. This should give a well-rounded discussion and provide a better understanding to this paper's decisions in section IV.

A. Fall Detection

Fall detection systems started peaking in interest during mid 2019 (Wang, Ellul and Azzopardi, 2020). During this time, detection techniques focused on alerting user on falls that already have occurred. However, recent methodologies hone into predictive methods aimed in learning and forecasting incidents before an occurrence happens.

Four different methodologies for fall detection are mentioned within the literature. These included wearable, vision, ambient based sensors and sensor fusion. Within this section the author aims to review these different sensors based, concluding with the motivation and gaps in research this paper aims to provide.

As this paper utilises microphones for detection, the majority of research will focus into ambient based methods. These methods include working with active infrared, RFID, pressure, ultrasonic and microphone sensors.

Solutions based on Wi-Fi (Wang, et al, 2020), provides a non-invasive methodology to detection. However, distinguishing between different individual falls remains a challenging. Additionally, research into fall detection through spatially distributed sensors in general (Principi, 2015), tends to be difficult and costly to implement in practice.

One paper that resonates closely with this paper's contribution is (Fang et al., 2021), which produced work on audio classification using CNNs. However the model size of 1.56MB failed to fit within the microcontroller memory constraints, and lacked consideration of possible improvements such as architectural changes.

Additional methods, such as wearables and visual based solutions have also been reviewed. However the author aimed to provide a novel approach of overcoming challenging classification of audio data.

Individual wearables are used in a majority of papers dealing with fall detection, where data from gyroscopes and accelerometers are utilised on various locations of the subject's person (such as waist, sole, torso, wrist). A key conclusion is that the use of a mobile phone, rather than additional wearable devices, is a more realistic solution on real world deployments. As subjects favoured carrying their mobile phone rather than wearing multiple portable devices (Wang, Ellul and Azzopardi, 2020).

Visual based detection has been explored on infrared cameras (Mastorakis and Makris, 2012), RGB cameras (Charfi et al., 2012) and RGB-D cameras (Cai et al., 2016)**,** for detection of falls. However these methods face issues in addressing the violation of privacy, due to the nature of sensitive data captured in an individual's living environment. This issue aids the authors solution in the pursue of audio-based systems

for less critical capturing of sensitive data. Moreover, visual based systems fail when changes to the environment occur, such as illumination, occlusion or lighting, affecting the ability to detect falls in different scenarios, leading to higher false alarm rates.

B. Fall Detection Datasets

Many papers fail to use genuine falls in their training data, often relying on stunt actors to perform these falls. This is because capturing genuine falls is usually difficult. The main issues is that elderly individuals typically fall in an uncontrolled manner, which doesn't replicate the same trajectory of stunt actors. In addition to this, obtaining fall data from the elderly may be dangerous as may cause bodily harm that have potentially long term or even fatal effects.

This paper creates a dataset with the authors' own recorded samples, with the addition of A3FALL dataset**,** mentioned in Section 5 of IV Method.

C. TinyML Research & Use Cases

TinyML is being used extensively in many fields, including healthcare, agriculture, human-computer interfaces (HCI) and recent buzzing field of autonomous driving (Wang, Ellul and Azzopardi, 2020) **.**

Current edge solutions rely on machine learning applications running on the cloud or private premise hardware. This signifies huge energy consumptions, privacy issues and network latency, which may affect the companies' goals such as lower energy usage targets, boosting aims for increased adoption of TinyML on ubiquitous edge devices. TinyML developments aim to cover five main fields: latency, privacy, connectivity, memory size and power consumption. TinyML more formally is defined of 3 parts, software, hardware and algorithms, where ML algorithms should be novel and span kilobytes in size to be viable for these resource-strict devices.

D. TinyML Model Optimisations

To run machine learning models on these edge devices, considerations to latency, privacy, connectivity, memory size and power consumption is needed, to limit these potential problems, researched techniques including pruning, quantisation, and clustering all aim to compress model sizes.

Quantisation, allows for model weights to be scaled down to an int8 value. This is because model parameters are usually represented by floating point 32 bit numbers. The methodology aims to reduce the representation to 8-bits or lower integers, allowing for memory constraints to be respected. However this decrease in memory, increases accuracy error, as less bits represent valid parameters. Despite this trade off many researchers have managed to run LLMs such as LlaMa-2 quantised to 4 bit integers on edge devices (Dhar et al., 2024) showcasing it's feasibility. Eq 1 shows the quantization equation, whereas Eq 2 showcases the dequantized equation,

$$round\ (x\ /\ scale\ + zero\_point)\ = \gamma \tag{1}$$

$$(\gamma - zero\_point) * scale = x \tag{2}$$

where y is the quantised integer 8-bit value, and x is floating point 32-bit value.

Quantisation also aids in edge devices, as integer processing is usually faster than floating point, reducing the overall computation.

Pruning, aims to reduce certain weights to zero, reducing the number of operable weights, and lowers memory and processing requirements. TensorFlow Lite Micro utilises structure pruning in its framework. During training, weights/features that seem to provide no useful information to the approximation of specific problems are omitted. However pruning issues arise as models are compressed as it leads to accuracy loss. DTMM method (Han, Xiao and Li, 2024) provides a pruning strategy for improving performance efficiency and accuracy after pruning.

These methods remain integral in the paper, as they are utilised within the methodology section, providing key intuition to the approaches in the method.

E. TensorFlow Lite, TFLM and Edge Impulse

When deploying a model to a TinyML device, the model is trained on a larger machine and then compiled and compressed using TensorFlow Lite or EON Impulse Compilers.

TensorFlow Lite (TensorFlow, 2019), is an open-source framework aimed at allowing edge devices a method of inferencing with TensorFlow models. This framework can generate binaries ranging within 300kB to 1MB for an image classification tasks, making it feasible for most edge devices due to its low memory and processing footprint. Deploying a model to an edge device requires converting a TensorFlow (TF) model into a flat buffer file (.tflite). Flat Buffers (Google, 2014.), aid in increasing performance by reducing memory allocation and avoiding the need to deserialise data. During this conversion, reduction techniques such as quantisation was used. Resulting in a model size reduction of 23.9% for CNN model and 59.5% for RNN model.

Edge Impulse (Edge Impulse, 2024)**,** is a cloud platform catering the full pipeline of implementing machine learning models onto embedded systems. It provides features such as data acquisition, feature extraction, training and deployment onto an embedded device.

F. Summary

The extensive literature review on fall detection illustrates that most papers present solutions that fail to work in real time (Wang, Ellul and Azzopardi, 2020)**.** This paper addresses these problems by prioritizing latency as a metric of model selection, making it feasible in real-time applications. In addition, most methodologies overlook privacy issues, especially when working with vision-based sensors, as they require sensitive data to operate. With these solutions, robustness usually suffers due to many factors such as occlusion, illumination and lighting. The proposed solution aims to use a non-invasive data source, where all processing occurs onboard locally.

Furthermore, visual sensors methodologies such as infrared cameras or RGB cameras, rely on placing sensors at fixed locations, meaning constraints are in place in terms of the locations of the devices. This papers aims to improve flexibility of sensor position it can operate in. Moreover, many methodologies depend on the usage of expensive sensors and equipment including UWB radars (Pavan, Caltabiano and Roveri, 2022), sometimes requiring multiple sensors, to obtain valid predictions. This has the problem of making these solutions difficult to realise in the real world due to cost and complexity and may lead to difficulties in providing scalability to many individuals.

Moreover, although fall detection seems to have great research presence, the author found lack of TinyML implementations of fall detection via audio classification, thus motivates the audio-based classification approach. To conclude, the proposed solution aims to be accessible to least developed countries by 1) utilising easy to find and cheap components within the solution, and 2) enabling of alerts based notifications via serial connection, rather than Wi-Fi, a consideration made under the assumption, most countries would at least have the infrastructure to power these devices, even if Wi-Fi access points are not available. This paper aims to solve fall detection based on audio classification by considering all relevant respective approaches and faults. Based on these pointers, the paper will now guide the readers through the methodology deployed.

IV. Method

Figure 2 details the five-stage pipeline from audio preprocessing to model classification and microcontroller (MCU) handling. Within the pipeline, two different models are discussed and developed focussing on the model architecture and training. Lastly, details about model shrinking and conversion to TensorFlow Lite model for MCU flashing are discussed.

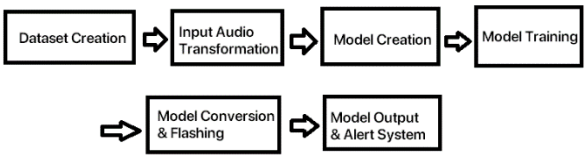


Figure 2 Six Stage Pipeline from Audio Preprocessing to Model Output and Alert System.

A. Dataset Creation

1) Microcontroller Configurations

Initially, a dataset was needed that covered all aspects of typical sounds heard within an elderly care setting. The dataset was collected using Raspberry Pico 2 WH (Ltd, n.d.) with a MAX9814 microphone as depicted in Figure 3. Four Picos placed in different corners of the room, two in high locations and two in low locations. This increased the number of samples produced per fall and reducing the risk to dangerous injuries. Moreover, each Pico had different microphone configurations such as gain and A/R alterations, as shown in **Table IX in Appendix**. Samples were recorded in four different locations, and sensitivities capturing variations of the same fall. This aimed to improve the model's robustness and reduce overall strain on number of human falls required. Although with the addition of multiple Picos comes complexity, the four Picos was necessary to trail all four different microphone configurations in each go. The author notes that such configuration is not necessary as sampling from different microphones, such as BLE33 and MAX9814 seem to be more effective.

2) Sampling and Recording Configuration

The microphone sampling frequency was set to 22.05 kHz with a representation of 16 bits per sample. The distance between the microphone and the subject fall locations was approximately 20-90 cm in range. The duration of samples collected from the microphone was 4 seconds. This sample duration was empirically set, due to it being the minimum amount of time needed to fall on cue and capture the full audio sample. A few non-fall audio samples were longer than 4 seconds in duration. In these cases, a random split of 4 seconds is used. To initialise a recording, each Pico interacted with a base control PC, which controlled recording of audio samples and saving these on a cloud storage location automatically. The Picos had a serial connection to base control, removing the need for manually interacting with each Pico and thus decreasing dataset collection time. Code is available on the papers' repository (GitHub, 2024)

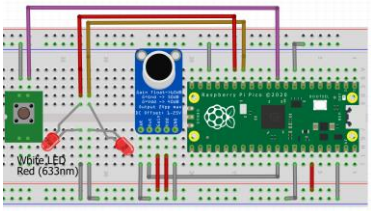


Figure 3 (Audio Recorder) Circuit Diagram for Pico with MAX9814 Microphone

3) Dataset Composition

To produce a robust and well-trained model, many different environments and sounds should be chosen, including variations of furniture, flooring like carpet, wood and tiles, as well as body type (light, average, heavy). **Table VI, VII in Appendix** showcases the composition of the different environments, samples count and two classifications of sounds, fall and non-fall.

4) Model Lack of Generalisation between Microphones

During model evaluation, it became evident that the accuracy of falls detection dropped significantly on the BLE33 microcontroller compared to the Pico's MAX9814. This drop was due to the model overreliance on data obtained purely from the MAX9814, with no samples made from the BLE33 built in microphone. This error went unnoticed throughout testing, as the devices relied on premade datasets, and did not sample microphone input from the MAX9814 to the model.

To resolve this, an addition set of recordings from the BLE33 built-in microphone was sampled, so that the model could generalise to both microphones. However it seemed that after the generalisation of both microphones, an increase in incorrect classifications on the Pico's model was apparent. This led to the hypothesis, that the model was not deep enough to classify the complexity of both varying microphones samples effectively.

5) Dataset Sample Count and Main Considerations

To ensure a balanced dataset, a total 1179 samples, consisting of 669 self-recorded samples, and 510 external dataset samples. Non-fall (or fall-like) audio samples counts are included in **Table XI in Appendix**. Non-fall samples included actions such as talking, dropping light items, shouting, television sounds, walking, running, jumping, random noises, ringtones and sliding of objects on the ground. These audio samples aimed to improve robustness and prevent false alarms,. The self-made dataset includes, 310 fall samples, 358 non fall samples, from both microphones, which is further shown in **Table XI in Appendix**. These samples are split equally between training and testing with 80%, 20% configuration respectively, such that only testing samples are inferred after the model is trained. Z-score normalisation was applied to the dataset.

A3FALL dataset provided additional background and fall audio samples by request from (Principi, 2015).

B. Input Audio Transformation

Given the dataset of fall and non-fall audio sounds were obtained, the next step in the methodology pipeline was raw audio signal processing, this is because raw audio signals usually fail to produce substantial outcomes and are difficult to train in the field of Tiny ML due to the mass amount of data provided by these audio signals and memory constraints on edge devices.
A dataset of audio samples was collected, in which the next step was digital signal processing (DSP), this is due to the large size and nature of raw audio signals. For example, a four second audio signal sampled at 22.05kHz would require 88.2 thousand bytes, assuming 8-bit representation. This is simply too large for model input.

To address this , many techniques aim to obtain a subset of highly representative audio features including Mel Spectrograms (MFE) and Mel-frequency cepstral coefficients (MFCCs). This enables the possibility of real time audio processing and classification on edge devices.

Raw audio signal is converted into MFEs through the pipeline provided in Figure 4. To provide the viewer some insights into a typical fall audio sample captured by an edge device, a four second sampled of a human fall sampled at 22.05 kHz input is shown in Figure 5.

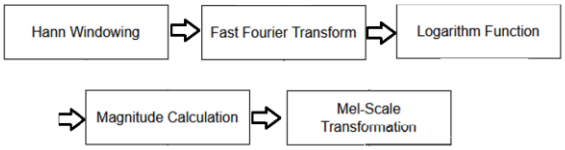


Figure 4 MFE Conversion Pipeline from raw audio signal to MFE feature set.

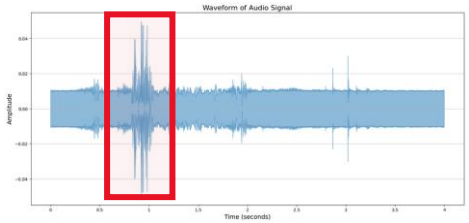


Figure 5 Audio Signal of Fall Audio Sample (4 second duration) sampled at 22.05kHz . With red box showing the specific period of the human fall.

To gain insights into the audio spectrum evolving over time, Short Time Fourier Transform is used with a sliding window mechanism, in which frames of the audio are cut into different subsets for calculating each chunk. These chunks then apply Fourier Transform function to obtain the frequency magnitudes. Figure 6 shows a FFT spectrum of a specific chunk, where audio signals related to the fall are usually composed of very low frequencies within the range of 1-200Hz, with a strong magnitude detected.

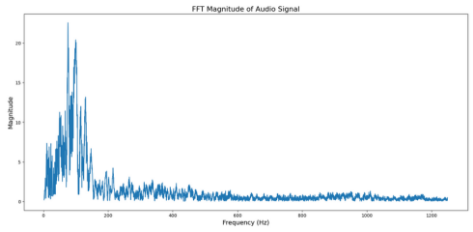


Figure 6 FFT Magnitude Spectrum of Fall Audio Sample with a majority occurrence within the range of 0 to 200Hz.

However, problems arise when audio sample durations are not an integer number of periods, such as a fraction of a period. Due to this, spectral leakage can occur, leading to noisy peaks in the frequency domain that are not present in the time domain. To solve this, the usage of Hann windowing function Eq (2), aims to provide a solution of reducing spectral leakage,

$$w(n) = 0.5\,(1 - \cos(2\pi \times n/N),\ 0 \leq n \leq N \quad (2)$$

assuming N samples are within one window, n is the current index of the window, N is the window length.

Figure 7 shows the weighting of Hann Window Function.

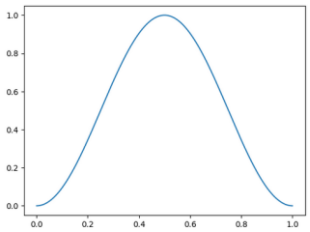


Figure 7 Default Weighting for Hann Window Function

Now that a spectrogram is obtained without spectral leakage, the next sub-step in obtaining MFE features, is applying logarithmic and Mel scale transformation. While frequency and sound intensities are presented linearly, human do not perceive sound linearly, but rather logarithmically. For example 200-400Hz and 13kHz-13.2kHz ranges will have different perceived pitch distance, despite both sharing the same 200Hz difference. To cater for this, the log is applied to the spectrogram's frequency and amplitude, this adjustment is shown in Figure 8, highlighting bright yellow for high intensities and dark dull purple for lower intensities and the red box showing the period of the fall event.

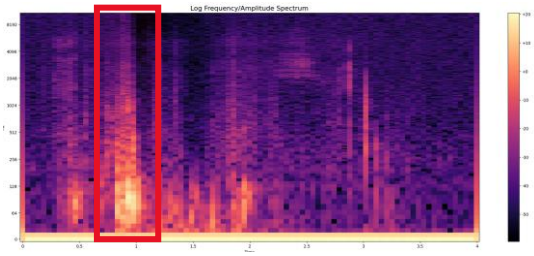


Figure 8 Log Frequency/Amplitude Spectrum of Fall Audio Sample

1) Mel Spectrograms & Filter banks

Mel spectrograms (MFE) are a flavour of spectrograms extensively used in AI research and real-world applications. Mel spectrograms main three goals are time-frequency representation, perceptually relevant (log) amplitude and frequency representation. The paper utilises Mel spectrograms, due to its improved perceptual relevance from

Mel scale, lower frequency emphasis which improve detection of low frequency sound such as falls.

Mel spectrogram utilises the Mel scale, that provides equal distances on the scale of perceived pitch, developed empirically, it is standardised with 1000Hz = 1000 Mel. The conversion from frequency to Mel's is shown in Eq (3)

$$m = 2595 \times \log(1 + f/500) \quad (3)$$

where m is Mel, and f is Frequency. The final Mel spectrogram for a typical fall is shown in Figure 9.

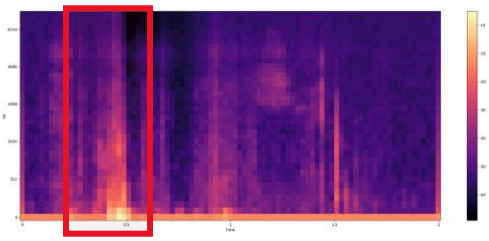


Figure 9 Mel Spectrum of Fall Audio Sample

Mel spectrum has three main hyperparameters that this paper will utilise, these include low and high frequency bands, and number of Mel bands. These parameters define the range and resolution of the Mel spectrum. This is then propagated to the next stage of the pipeline, the model input.

2) Mel-frequency cepstral coefficients (MFCC)

An additional step usually taken, is to calculate the MFCC, which will utilise this feature set when working with RNNs. The reason for this is due to the capture of spectral envelope, which are specific details that mimics human perceived audio scaling and tends to be more robust to noise compared to MFE. This is due to MFCC's focus overall spectral shape rather than individual frequency bins.

3) Audio Preprocessing Microcontroller Considerations

Before model creation, a main concern throughout the development was the MCU capabilities, as inference will run on it. Audio preprocessing adds a further constraint on latency for MCUs aiming to provide real time onboard inference. This is because processing is required to transform audio into the correct input format for the model. This paper considers this constraint by implementing pre-computed buffers for improved latency at the cost of memory usage.

C. Model Creation

During this study, two main models were developed and tested based on the capabilities of the two different microcontrollers as shown in Table I. CNN models are utilised for the Arduino BLE 33 Rev 2, while RNN LSTMs model were used for the Raspberry Pico 2 WH. The author notes, careful hyperparameter and architecture selection were made when designing of all mentioned models, as this affected the overall outcome in terms of accuracy, latency and robustness.

TABLE I.
MICROCONTROLLERS DETAILS

Microcontroller	Available Resources				
	Processor	***RAM***	***ROM***	***Microphone***	***Model Used***
Arduino Nano 33 BLE Sense Rev 2	Cortex-M4F-64MHz	256kB of RAM	1024 kB	Built-in Microphone	CNN Model
Raspberry Pi Pico WH	Dual-core Arm Cortex M0+ 133MHz	264kB of SRAM	2048 kB	MAX9814 Electret Microphone	RNN Model

1) CNN Model

For this model, a 2D convolutional neural network (CNN) was implemented. CNNs allow for working with large matrices and reduced parameter sets via a number of convolution operations. They inherently capture an inductive bias to locality of neighbouring values. To reduce memory usage during inference, max pooling layers were replaced for a standard convolution with stride of two. This saved in memory usage and processing time, by reduced operations and memory.

The CNN architecture consists of two convolutional layers, concluding with a final linear layer and two neuron output. The loss function used was cross entropy loss. The optimiser employed Adam. A dropout layer with a rate of 0.5 was implemented between the convolutions and linear layer. The learning rate was initially set to 0.05, with a cosine decay LR scheduler. ReLU was used as the chosen activation function and the model was trained with 50 epochs. Figure 10 showcases the model architecture used, the input takes in a two-dimensional matrix, Mel spectrogram (MFE) with dimensions, (number of Mel filter, number of frames), in which number of frames is in **Table XII of Appendix**.

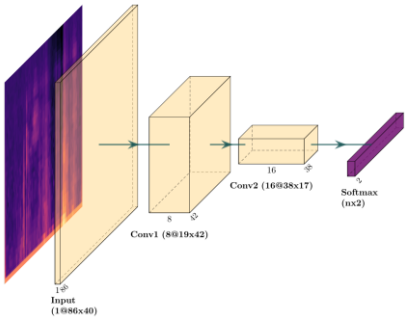


Figure 10 Proposed CNN Model Architecture used in Method 1

The Edge Impulse platform was used for training a CNN model on the Arduino BLE 33 Rev 2, handling normalisation, augmentation and the necessary data conversions.

Inspiration from Inception (Szegedy et al., 2014) models' choice of multiple convolution types was not utilised due to memory constraints. Furthermore, residual connections from Res Net (He et al., 2015) was also not used due to the shallow nature of the proposed CNN model. Strong focus on attempting to make the model as shallow and accurate as possible as this resulted in lower latency.

2) RNN LSTM Model

RNN LTSMs aim to provide a recurrent methodology for learning from temporal and sequential data, such that it is used within this solution to provide audio classification based

on audio in a temporal way. For example, CNNs failed to distinguish between a fall and constant thudding produced by someone walking, thus a time-series based model was necessary. LTSMs benefit from consistent number of weight parameters regardless on the number of audio samples inputted. To solve some of the misclassifications, an RNN model was deployed, utilising LTSM due to their improvements on vanishing and exploding gradients. However, due to LTSMs tendency to overfit, robust regularisation methods such as dropout were deployed.

The model follows a simple architecture, consisting of two LSTM modules each with 48 hidden units, dropout and a linear layer. Z-score normalisation is performed based on a training subset. To backpropagate, Adam was used with the final layer utilising SoftMax and sparse categorical cross entropy loss. A batch size of 64 was chosen. The learning rate was set to 0.06235912 and epochs set to 200. No learning rate scheduler was utilised. Hyperparameter selection is mentioned in Section D.

The models' input relied on MFCC. This is a two-dimensional feature set matrix defined by the number of DCT coefficients and number of frames. The author did discover accuracy sensitivity in terms of number of Mel frequencies used when training, and further experimentation is used to derive reasoning for usage of 55-65 MFE parameter setting as shown in **Appendix A**.

The architecture for the RNN model is within Figure 11.

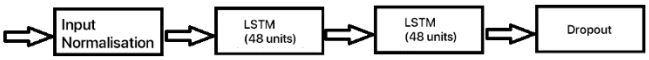


Figure 11 RNN Architecture with two LSTM modules (48 units).

D. Model Training and Hypertuning

The CNN network was trained via Edge Impulse platform, where the model was defined in TensorFlow on an online UI. For the RNN network , the model was trained on a local GPU RTX 3070Ti using a TensorFlow script, which achieved suboptimal accuracies, further working was done through the usage of a tool Weights and Biases (Weights and Biases, 2024). This performed a Bayesian search of the hyperparameter search space. The hyperparameter search covered different dropout values ranging from 0.3 to 0.7, learning rate from 0.00001 to 0.1, batch sizes from 8 to 128, for epochs of 200, this hyper tuner ran 15579 times.

E. Model Conversion and Flashing

TensorFlow model was converted into a TensorFlow Lite model, by initialising a TensorFlow converter with 8-bit quantisation, this is saved as a .tflite file. With this file, xxd command is used to convert this into a constant C-byte array as a C header file. This format allows to import the model into the application without the need for addition libraries. This outputs to a model.h file which can be imported into Arduino IDE and then subsequently flashed on to a device.

F. Model Output & Alert System

The model's output provides JSON response every 1000ms via serial connection to a web-based solution, which then presents this data in graphical manner.

1) Serial Connectivity for Device Results

Only serial connection to a personal computer was utilised, sending continuous JSON updates about the specific model status.
JSON sent by the device had the following format,

```
{"device": device_name, "model_version":
  "1.1", "class": "fall", "prob": 0.96}
```

2) Web Based Solution for Fall Detection Statuses

To provide a solution which did not require a technical understanding of IoT devices, a web-based platform was developed. The platform provided users, to view devices connected and the number of falls that occurred. It allows for filtering falls by device and time period, as well as alerting users of potential urgent care requests. An example of the web app UI is shown in **Appendix B.**

G. Additional Considerations and Challenges

1) MCU Considerations and Optimisations

Due to the lack of floating-point accelerators within the ARM Cortex M0+ CPU for the Pico WH MCU, this paper utilises CMSIS-DSP (Keil, n.d.) and fixed-point numerical format 16-bit integer fixed-point (Q1.15) allowing floating point data to be represented by integer values rather than floats. This addition increased overall performance of the MCU. Further optimisations included storing precomputed matrices for Mel and DCT weight coefficients, saving in real time processing of constants.

2) Active Listening Challenges, Inference & Latency Importance

When working with live classification of audio, many considerations were made. To record sampled audio and run inference concurrently, a mechanism named double buffering was used. In double buffering, two buffers were in constant use, with alternating objectives. The first objective was to fill the buffer with the recorded audio sample, and the other to process the previous audio sample. This ensured that the MCU did not miss any potential falls during processing. To decrease likelihood of missed events, overlapping of different audio sample phases were used, this was controlled by a variable namely window increase and window length as shown in Figure 12.

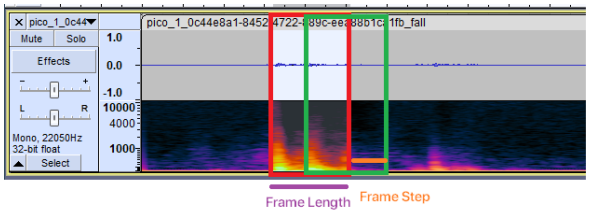


Figure 12 Window Inference (Frame Length) and Sliding Factor (Frame Step).

Implementation of Arduino BLE 33 utilises the Edge Impulse platform handling both input processing, model training and testing and further deployment onto the device, with minimal changes in the output effect.

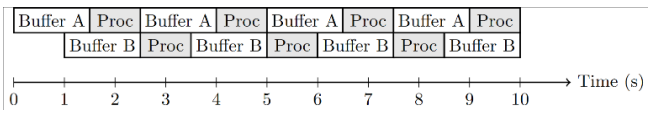


Figure 13 Double Buffer Pipeline for 10 second period.

The Pico WH implementation took an independent approach using purely TensorFlow, Arduino and CMSIS-DSP libraries to achieve inference. The proposed solution worked by storing buffers holding 1.5 seconds, or four kilobytes of audio as shown in Figure 13. During each block of audio sampling stored in buffers A_1 and A_2, led by an additional processing and inference block of one second.

Given an audio sampling constraint of 1500ms, the microcontroller must process the audio and make a model prediction within a 1000ms constraint, or active listening will fail. For this implementation, specific timing for audio signal processing stages within the feature extraction pipeline is shown in **Table IX of Appendix**. Considerable time is spent on the FFT Mel Scale Conversion calculation and model prediction which the author aimed to reduce to maintain feasibility.

Due to constraint, in order to actively listen, a total time for processing and inference must be less than a period of 950ms, however the solution achieved an inference of 1052ms on average with a variance of 35ms.
Further optimisation is needed to allow for active listening. As mentioned, initial optimisations worked by trading memory for performance, for example preloading computed matrixes for improved throughput reduced overall computation of logarithm, DCT and FFT Mel Scale conversion. However, due to the long processing time of large matrices like DCT matrix, 16 milliseconds was too long of a period per frame.
Future solutions should include utilisation of dual cores of the RP2040 ARM Cortex-M0+ which allows for execution of code in parallel. Due to the complexity involved with dealing with the multicore library using ArduinoCore Mbed library, and that most Arduino functions aren't thread safe, further optimisation was not considered. The author understands these limitations, and showcases these such that further research could improve this.

3) Model Accuracies, Size and Inference

Table II in Appendix showcases the different models and their respective accuracy and inference times, with comparison between normal and quantised versions.

V. Testing & Evaluation

A. Experiments, Method Failures and Limitation

A deeper analysis of the dataset and model classification errors presented where segments of silence in audio samples were labelled as falls, due to the presence of silence during falls. Thus multiple iterations on the dataset aimed on providing more reliable and representative samples of the fall action. This included sampling different sections of audio as well as augmentations. Audio samples of footsteps and jumps were most misclassified; however the author assumed that elderly individuals would not naturally jump in their daily activities and thus this misclassification may not affect the overall practicality of the solution in reality.
Although the paper aims to include many different environments, falls may lead to possibly dangerous injuries, such that a limited dataset was made. Other limitations such as elderly fall trajectory may be different from that of young adults, especially when falls are intentional rather than accidental. These types of simulated falls may be valuable to increase the dataset size however do not potentially cover the intentional examples of elderly accidental falls, which are hard and unexpected to record.

In addition to this, during the optimisation stage, selection of number of Mel Coefficients frequency was empirically selected to provide the highest test accuracy as well as reduce the overall latency as shown in **Appendix A**. This concluded the lowest Mel Coefficients frequency of 55-60 was most ideal with highest validation accuracy.

VI. Results

A. CNN Model Results

*CNN results are presented in **Table XII, XIII in Appendix**, showing strong F1 scores for both fall and non-fall at 0.94, as well as quite strong testing and validation accuracy of 92.41% and 93.7% respectively.*

RNN Model Results

RNN results are presented in **Tables XIV, XV in Appendix**, showing slightly improved validation & test accuracies as well as false position rates for fall classifications than the CNN confusion matrix, however reduced F1 score achieved for fall category.

VII. Conclusion and Future Work

A. Conclusion

This paper provides a comprehensive methodology for classifying audio into two different categories, fall-like and non-fall audio, with the aim to produce an alert-based system. The system developed relied on inexpensive components and could be easily reproduced, preserving privacy with all processing and inference occurring locally. The proposed solution provided the creation of a dataset utilising multiple microphones to generalise models well.
The main problems identified through the development, included trade-offs with Mel-frequency selection, model accuracy and latency. One main bottleneck was in the digital signal processing (DSP), specifically in large matrix operations on the FFT Mel Scale conversion, which prevented real time inference.
Due to bottlenecks in DSP, the impact of real time inference was significantly higher compared to model inference, with majority of DSP operations taking 790ms compared to model inference operation taking 262ms. It was evident techniques such as pruning, clustering may not have such a massive impact on the success of the project. Subsequently the paper delves into some solutions to reduce DSP latency, such as the pre-computed weighted matrix, to reduce the burden of recalculating on each sample, and many other trade-offs between memory and processing time.

B. Future Work

Readers are encouraged to research into faster methodologies for collecting audio data through audio-based simulation, similar to accelerometer data simulation research by OpenSim (Georgios Mastorakis et al., 2016). Further research should cover ambitious transformer-based methodologies for audio classification using methods like Tiny ViT (Wu et al., 2022) and KANs (Liu et al., 2024) (Kolmogorov–Arnold Networks) with the potential of improved parameter efficiency.

VIII. Acknowledgment

Thank you, **Dr Stefan Poslad**, *Queen Mary University of London*, for aid as supervisor. Thank you, Prof. Emanuele Principi, Università Politecnica delle Marche, for access to dataset A3FALL (Principi, 2015)

IX. References

Apple (2019). *Watch*. [online] Apple (United Kingdom). Available at: https://www.apple.com/uk/watch/.

ARM (2023). *Pruning And Clustering For Arm Ethos-U NPU*. [online] Arm.com. Available at: https://community.arm.com/arm-community-blogs/b/ai-and-ml-blog/posts/pruning-clustering-arm-ethos-u-npu [Accessed 4 Aug. 2024].

Cai, Z., Han, J., Liu, L. and Shao, L. (2016). RGB-D datasets using Microsoft Kinect or similar sensors: a survey. *Multimedia Tools and Applications*, 76(3), pp.4313–4355. doi:https://doi.org/10.1007/s11042-016-3374-6.

Charfi, I., Miteran, J., Dubois, J., Atri, M. and Tourki, R. (2012). Definition and Performance Evaluation of a Robust SVM Based Fall Detection Solution. *2012 Eighth International Conference on Signal Image Technology and Internet Based Systems*. doi:https://doi.org/10.1109/sitis.2012.155.

Dhar, N., Deng, B., Lo, D., Wu, X., Zhao, L. and Suo, K. (2024). An Empirical Analysis and Resource Footprint Study of Deploying Large Language Models on Edge Devices. doi:https://doi.org/10.1145/3603287.3651205.

Edge Impulse (n.d.). *About Edge Impulse*. [online] edgeimpulse.com. Available at: https://edgeimpulse.com/about.

Fang, K., Xu, Z., Li, Y. and Pan, J. (2021). A Fall Detection using Sound Technology Based on TinyML. doi:https://doi.org/10.1109/itme53901.2021.00053.

Georgios Mastorakis, Hildenbrand, X., Grand, K. and Makris, D. (2016). Customisable Fall detection: a Hybrid Approach Using Physics Based Simulation and Machine Learning. (64162981).

GitHub (2024). *TinyAED*. [online] GitHub. Available at: https://github.com/makiisthenes/tinyaed..

Google (n.d.). *FlatBuffers: FlatBuffers*. [online] flatbuffers.dev. Available at: https://flatbuffers.dev/.

Gudiño-Ochoa, A., Julio Alberto García-Rodríguez, Ochoa-Ornelas, R., Jorge Ivan Cuevas-Chávez and Daniel Alejandro Sánchez-Arias (2024). Noninvasive Diabetes Detection through Human Breath Using TinyML-Powered E-Nose. *Sensors*, 24(4), pp.1294–1294. doi:https://doi.org/10.3390/s24041294.

Han, L., Xiao, Z. and Li, Z. (n.d.). *DTMM: Deploying TinyML Models on Extremely Weak IoT Devices with Pruning*. [online] Available at: https://arxiv.org/pdf/2401.09068 [Accessed 4 Aug. 2024].

He, K., Zhang, X., Ren, S. and Sun, J. (2015). *Deep Residual Learning for Image Recognition*. [online] arXiv.org. Available at: https://arxiv.org/abs/1512.03385.

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, [online] 9(8), pp.1735–1780. doi:https://doi.org/10.1162/neco.1997.9.8.1735.

Jacob, B., Skirmantas Kligys, Chen, B., Zhu, M., Tang, M., Howard, A.W., Adam, H. and Dmitry Kalenichenko (2017). Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. *arXiv (Cornell University)*. doi:https://doi.org/10.48550/arxiv.1712.05877.

Keil (n.d.). *CMSIS DSP Software Library*. [online] www.keil.com. Available at: https://www.keil.com/pack/doc/CMSIS/DSP/html/index.html.

Mastorakis, G. and Makris, D. (2012). Fall detection system using Kinect's infrared sensor. *Journal of Real-Time Image Processing*, 9(4), pp.635–646. doi:https://doi.org/10.1007/s11554-012-0246-9.

Liu, J., Tripathi, S., Kurup, U. and Shah, M. (2020). *Pruning Algorithms to Accelerate Convolutional Neural Networks for Edge Applications: A Survey*. [online] arXiv.org. Available at: https://arxiv.org/abs/2005.04275 [Accessed 4 Aug. 2024].

Liu, Z., Wang, Y., Vaidya, S., Ruehle, F., Halverson, J., Soljačić, M., Hou, T.Y. and Tegmark, M. (2024). KAN: Kolmogorov-Arnold Networks. *arXiv (Cornell University)*. doi:https://doi.org/10.48550/arxiv.2404.19756.

Ltd, R.P. (n.d.). *Buy a Raspberry Pi Pico*. [online] Raspberry Pi. Available at: https://www.raspberrypi.com/products/raspberry-pi-pico/ [Accessed 17 Aug. 2024].

O'Shea, K. and Nash, R. (2015). An Introduction to Convolutional Neural Networks. *arXiv:1511.08458 [cs]*, [online] 2. Available at: https://arxiv.org/abs/1511.08458.

Pavan, M., Caltabiano, A. and Roveri, M. (2022). TinyML for UWB-radar based presence detection. *2022 International Joint Conference on Neural Networks (IJCNN)*. doi:https://doi.org/10.1109/ijcnn55064.2022.9892925.

Principi, E. (2015). *Floor Acoustic Sensor Fall Dataset*. [online] Univpm.it. Available at: http://www.a3lab.dii.univpm.it/research/fasdataset [Accessed 4 Aug. 2024].

Samsung (n.d.). *Use the Detect fall feature on your Samsung smart watch*. [online] Samsung Electronics America. Available at:

https://www.samsung.com/us/support/answer/ANS00087244/.

Santos, G., Endo, P., Monteiro, K., Rocha, E., Silva, I. and Lynn, T. (2019). Accelerometer-Based Human Fall Detection Using Convolutional Neural Networks. *Sensors*, [online] 19(7), p.1644. doi:https://doi.org/10.3390/s19071644.

Simonson, P., Park, D.W. and Pooley, J. (2022). Exclusiones/Exclusions: El papel de la historia en saldar la deuda histórica del campo. *History of Media Studies*, 2. doi:https://doi.org/10.32376/d895a0ea.cb32b735.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A. (2014). *Going Deeper with Convolutions*. [online] arXiv.org. Available at: https://arxiv.org/abs/1409.4842.

TensorFlow (2019). *TensorFlow Lite | TensorFlow Lite | TensorFlow*. [online] TensorFlow. Available at: https://www.tensorflow.org/lite.

Vailshery, L. (2023). *IoT Connected Devices Worldwide 2019-2030*. [online] Statista. Available at: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/.

Wang, X., Ellul, J. and Azzopardi, G. (2020). Elderly Fall Detection Systems: A Literature Survey. *Frontiers in Robotics and AI*, 7. doi:https://doi.org/10.3389/frobt.2020.00071.

Warden, P. and Situnayake, D. (2019). *TinyML*. O'Reilly Media.

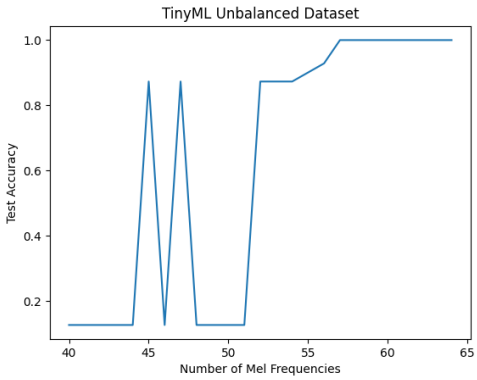
Weights and Biases (2024). *Weights & Biases – Developer Tools for ML*. [online] wandb.ai. Available at: https://wandb.ai/site [Accessed 21 Aug. 2024].

World Health Organization (2021). *Falls*. [online] World Health Organization. Available at: https://www.who.int/news-room/fact-sheets/detail/falls.

Wu, K., Zhang, J., Peng, H., Liu, M., Xiao, B., Fu, J. and Yuan, L. (2022). *TinyViT: Fast Pretraining Distillation for Small Vision Transformers*. [online] arXiv.org. doi:https://doi.org/10.48550/arXiv.2207.10666.

Zacharia, A., Zacharia, D., Karras, A., Karras, C., Giannoukou, I., Giotopoulos, K.C. and Sioutas, S. (2022). *An Intelligent Microprocessor Integrating TinyML in Smart Hotels for Rapid Accident Prevention*. [online] IEEE Xplore. doi:https://doi.org/10.1109/SEEDA-CECNSM57760.2022.9932982.

X. Appendix



Appendix A Empirical results of accuracy training balanced dataset for 100 epochs and reviewing number of Mel frequencies between 40-65.

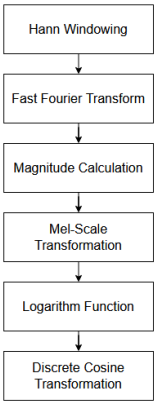


Figure 12 Audio Preprocessing Pipeline from Audio Signal to Mel Spectrogram (MFE)

TABLE II. ACCURACY AND SPACE REQUIREMENTS OF TENSORFLOW AND TENSORFLOW LITE MODEL

Method & Platform	**Parameter**		
	Test Accuracy (%)	***Model Size***	***Inference (ms)***
Arduino BLE 33 / CNN Model			
TensorFlow	**94.9 %**	**73.6 KB**	**959ms**
TensorFlow Lite	**93.7 %**	**56.0 KB**	**418ms**
Raspberry Pico WH / RNN LSTMs			
TensorFlow	**95.25 %**	**280.6 KB**	**29863ms**
TensorFlow Lite	**95.23%**	**113.6 KB**	**636ms**

TABLE III. NON-FALL DATASET COMPOSITION TABLE

Non-Fall Data Category	***Sample Quantities***
Normal Talking in the Room	5 samples x 3 devices x 4 second duration
Dropping light items on the ground.	5 samples x 3 devices x 4 second duration

TABLE IV. MFCC FEATURE EXTRACTION PIPELINE TIMELINE

Operation Name	*Operation Duration (ms)*
Hann Multiplication	**1**
RFFT Calculation	**6**
Magnitude Calculation	**2**
FFT Mel Scale Conversion Calculation	**13**
Logarithm Apply Calculation	**1**

Operation Name	*Operation Duration (ms)*
DCT Coeff's MFCC	**1**
Processing Time per Frame	**24**
Total Processing Time (33 Frames)	**790**
Total Model Inference Time	**262**
Total Time for Processing + Inference	**1052 +-35**

Appendix B: Image of GUI of Alert Detection Web App Platform.

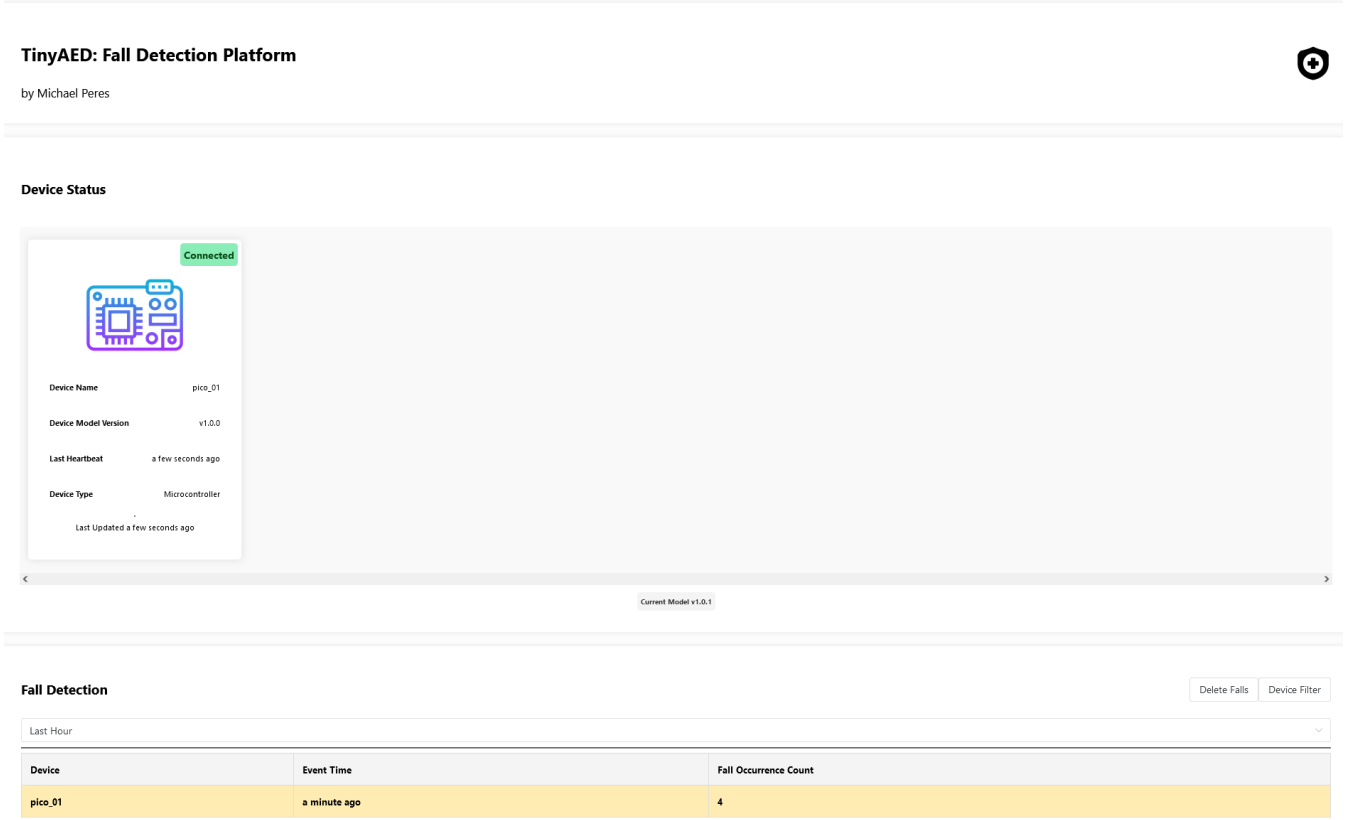


TABLE V. FALL DATASET COMPOSITION TABLE

Scenario – Environment	*Sound Type*	*Device Microphone*	*Sample Count*	*Sample Duration*	*Total Duration of Samples*
Carpet – Bedroom	*Fall*	*BLE 33*	*225 samples*	*4 seconds*	*900 seconds*
Background Human Activity - Bedroom	*Non-Fall*	*BLE 33*	*225 samples*	*4 seconds*	*900 seconds*
Carpet – Bedroom	*Fall*	*MAX9814*	*225 samples*	*4 seconds*	*900 seconds*
Background Human Activity - Bedroom	*Non-Fall*	*MAX9814*	*225 samples*	*4 seconds*	*900 seconds*

TABLE VI. NON-FALL DATASET COMPOSITION TABLE

Non-Fall Data Category	*Sample Quantities*
Normal Talking in the Room	5 samples x 3 devices x 4 second duration
Dropping light items on the ground.	5 samples x 3 devices x 4 second duration
Shouting in the room	5 samples x 3 devices x 4 second duration
TV or YouTube in the room	5 samples x 3 devices x 4 second duration

Non-Fall Data Category	*Sample Quantities*
Walking near microphone	5 samples x 3 devices x 4 second duration
Running near microphone.	5 samples x 3 devices x 4 second duration
Jumping near microphone	5 samples x 3 devices x 4 second duration
Jumping far microphone.	5 samples x 3 devices x 4 second duration
Silence with noise.	5 samples x 3 devices x 4 second duration
Normal with noise.	5 samples x 3 devices x 4 second duration
Sliding Small Object through floor	5 samples x 3 devices x 4 second duration
Sliding Big Object through floor	5 samples x 3 devices x 4 second duration
Digital Sounds/ Music of Devices (beeps, ringtones)	5 samples x 3 devices x 4 second duration

TABLE VII. HYPERPARAMETER TABLE ENTRIES

Hyperparameter Name	*Hyperparameter Value*
Audio Sample Duration (seconds)	4
Sample Rate (Hz)	22050
Frame Length (Sample)	2048
Frame Step (Sample)	1024
FFT Length	2048
Min Frequency Band (Hz)	20
Max Frequency Band (Hz)	11025
Number of Mel Frequencies (Hz)	40
Number of MFCCs (Count)	18

TABLE VIII. PROJECT MICROCONTROLLER CONSTRAINTS

Constraints	
Resource	***MCU Average***
RAM Space	> 500KB
ROM Space	> 2MB
Computation Frequency	10-1000 MHz
Power Utilisation	10-100mAH

TABLE IX. MICROCONTROLLER CONFIGURATIONS

Configurations		
Pico Device #	***Attack/Release Ratio***	***Gain***
Pico 1	GND	GND
Pico 2	GND	V_{DD}
Pico 3	V_{DD}	GND
Pico 4	V_{DD}	V_{DD}

where GND is equal to (a A/R ratio of 1:500, maximum gain of 50.6dB, minimum gain of 29.0dB),
V_{DD} has (a A/R ratio of 1:2000, maximum gain of 40.5dB, minimum gain of 18.7dB),
obtained from the MAX9814 documentation (https://www.analog.com/media/en/technical-documentation/data-sheets/MAX9814.pdf).

TABLE X.

TABLE XI. DUAL MICROPHONE SAMPLES

Samples Recorded			
Sample Type	**Total Samples**	**Fall-Like Samples**	**Non-Fall Samples**
Microphone MAX9814			
Training	451	228	223
Testing	73	31	42
Built-in Microphone BLE33			
Training	119	43	76
Testing	26	8	17

TABLE XII. CNN CONFUSION MATRIX

	Fall	***Non-Fall***
Fall	100%	0%
Non-Fall	11.9%	88.1%
F1 Score	0.94	0.94

TABLE XIII. CNN ACCURACY AND LOSS

Train Accuracy	***Validation Accuracy***	***Test Accuracy***
98.40%	92.41%	93.7%
Train Loss	***Validation Loss***	***Test Loss***
0.0641	0.3056	0.150

TABLE XIV. RNN CONFUSION MATRIX

	Fall	***Non-Fall***
Fall	79.2%	20.8%
Non-Fall	1.6%	98.4%
F1 Score	0.87	0.95

TABLE XV. RNN ACCURACY AND LOSS

Train Accuracy	***Validation Accuracy***	***Test Accuracy***
100%	92.60%	95.23%
Train Loss	***Validation Loss***	***Test Loss***
1.078e-05	0.19	0.3238