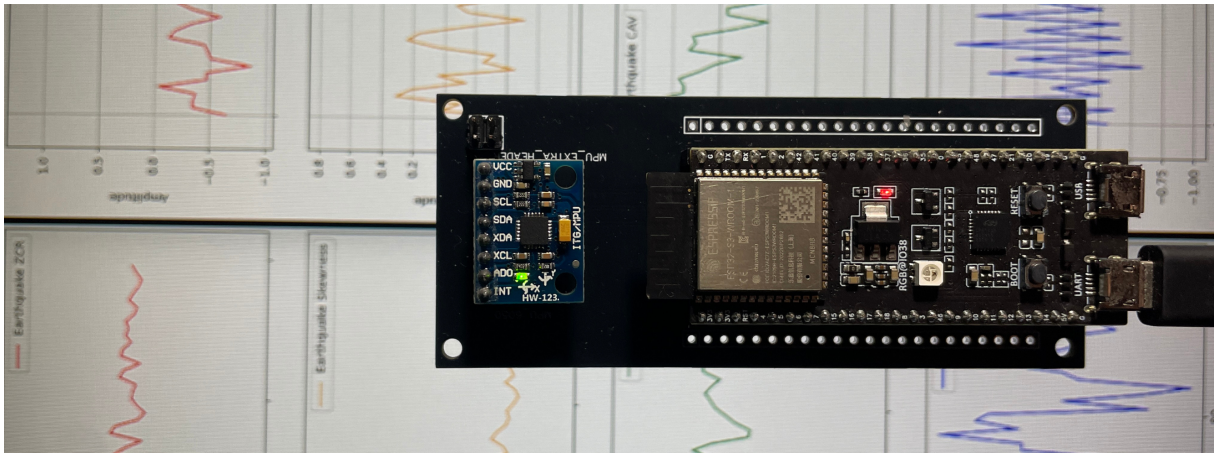
INTERNATIONAL
HELLENIC
UNIVERSITY

SCHOOL OF ENGINEERING

DEPARTMENT OF INFORMATION AND ELECTRONICS ENGINEERING

THESIS

SEISMIC EVENT DETECTION WITH ESP32 MICROCONTROLLER AND TINYML



Student
George Xenofontos
Student ID: 519004

Supervisor
Maria Papadopoulou
Assistant Professor

September 10, 2024

Detection of Seismic Activity Using ESP32 Microcontroller and TinyML
Unit Code 23249
George Xenofontos
Maria Papadopoulou
16/09/2023
16/07/2024

I certify that I am the author of this thesis and that any assistance I have had in its preparation is fully acknowledged and referenced in the thesis. I have also recorded any sources from which I have used data, ideas, images, and text, whether quoted verbatim or paraphrased. Furthermore, I certify that this thesis was prepared by me personally, specifically as a thesis, at the Department of Computer Engineering and Electronic Systems Engineering, IHU.

The approval of the thesis by the Department of Computer Engineering and Electronic Systems of the International Hellenic University does not necessarily imply the Department's acceptance of the author's views.

«Have you been told it's impossible? If you really want it, confirm it for yourself.»

Prologue

The motivation for this thesis stems from a passion for learning about the devastating effects of earthquakes and the desire to aid with technologies which can help reduce the harm they cause. This project is centred on constructing an estimation system of seismic waves using ESP32 microcontroller along with MPU6050 sensor. After collecting the dataset and analysis, full-fledged best of the CNN model to identify the seismic activities from noise with 93% of accuracy is achieved. It was then brought to TensorFlow Lite format and managed to run on the ESP32 rid of hardware constraints.
The proposed system aims at delivering real-time data that may be of great help in structural engineering especially the response of buildings to seismic waves. Utilizing a custom PCB, the wiring of the system was made more reliable so that during testing, connections were stable. This thesis helped me develop theoretical advancements in data analysis, machine learning and heteroscedastic models while enriching the identical practical experience in practice through working with the hardware environments. The project has therefore contributed significantly to enhancing the aspect of earthquake awareness since it presents a cost effective method of perceiving the seismic waves that can be implemented in real situations.

Summary

This thesis describes the Seismic Detection System designed and developed using a low power hardware platform known as TinyML with incorporating machine learning (ML). The idea was to create a small-power consumption and volume device that could differentiate between earthquakes and other vibrations in realtime to increase structure reliability without cloud computing. The core of the system is the ESP32 microcontroller and MPU6050 sensor as they are capable to work with TinyML and provide accurate data about motion. A Convolutional Neural Network (CNN) was trained using data from the SCEDC and tested against controlled experiment. It was identified that feature engineering was instrumental in determining high performing features that comprises of Pa, ZCR, Skewness, Kurtosis, CAV, ZHR, TauC.Although the CNN model has high accuracy rate of 93% for the identification of seismic events. Furthermore, the thesis covers the entire process of development, from data collection, hardware design, the application of the algorithms of ML up to the creation of the final product. The opportunities and key challenges which arose during the creation process have been the following ones: The sensor noise was larger and required more intricate filtering techniques; The memory resources and memory management were to be elaborate, as MEMS requires dynamic memory allocation. The final system was then test and tried out with various inputs till its stability was proven in a test bench as well as with real data inputs. This work presents a concrete and inexpensive method for seismic monitoring, with the actual-time detection performance which can find applications in different fields in enhancing structural integrity and emergency response to disasters.

Ανίχνευση Σεισμικών Δραστηριοτήτων με μικροελεγκτή ESP32 και χρήση TinyML

Γιώργος Ξενοφώντος

Περίληψη

Η παρούσα διπλωματική εργασία περιγράφει το σύστημα σεισμικής ανίχνευσης που σχεδιάστηκε και αναπτύχθηκε με τη χρήση μιας τεχνολογίας υλικού χαμηλής ισχύος, γνωστής ως TinyML. Η ιδέα ήταν να δημιουργηθεί μια συσκευή μικρής κατανάλωσης ενέργειας και όγκου που θα μπορούσε να διακρίνει μεταξύ σεισμών και άλλων δονήσεων σε πραγματικό χρόνο για την αύξηση της αξιοπιστίας των κατασκευών χωρίς cloud computing. Ο πυρήνας του συστήματος είναι ο μικροελεγκτής ESP32 και ο αισθητήρας MPU6050, καθώς είναι σε θέση να συνεργαστούν με το TinyML και να παρέχουν ακριβή δεδομένα σχετικά με την κίνηση. Ένα συνελικτικό νευρωνικό δίκτυο (CNN) εκπαιδεύτηκε χρησιμοποιώντας δεδομένα από το SCEDC και δοκιμάστηκε με ελεγχόμενο πείραμα. Διαπιστώθηκε ότι η χρήση feature engineering ήταν καθοριστική για τον προσδιορισμό των χαρακτηριστικών που τα οποία περιλαμβάνουν τα Pa, ZCR, Skewness, Kurtosis, CAV, ZHR, TauC. Το μοντέλο CNN με τη χρήση των συγκεκριμένων χαρακτηριστικών έχει φτάσει το υψηλό ποσοστό ακρίβειας 93% για την αναγνώριση των σεισμικών γεγονότων. Επιπλέον, η διπλωματική εργασία καλύπτει ολόκληρη τη διαδικασία ανάπτυξης, από τη συλλογή δεδομένων, το σχεδιασμό του υλικού, την εφαρμογή των αλγορίθμων της ML μέχρι τη δημιουργία του τελικού έργου. Οι ευκαιρίες και οι βασικές προκλήσεις που προέκυψαν κατά τη διάρκεια της διαδικασίας δημιουργίας ήταν οι ακόλουθες: Ο θόρυβος του αισθητήρα ήταν μεγαλύτερος και απαιτούσε πιο περίπλοκες τεχνικές φιλτραρίσματος, οι πόροι μνήμης και η διαχείριση της μνήμης έπρεπε να είναι εξελιγμένες, καθώς τα MEMS απαιτούν δυναμική κατανομή μνήμης. Στη συνέχεια, το τελικό σύστημα ελέγχθηκε και δοκιμάστηκε με διάφορες εισόδους μέχρι να αποδειχθεί η σταθερότητά του σε πάγκο δοκιμών καθώς και με πραγματικές εισόδους δεδομένων. Η εργασία αυτή παρουσιάζει μια υγκεκριμένη και οικονομική μέθοδο για τη σεισμική παρακολούθηση, με απόδοση ανίχνευσης σε πραγματικό χρόνο, η οποία μπορεί να βρει εφαρμογές σε διάφορους τομείς για την ενίσχυση της δομικής ακεραιότητας και την αντιμετώπιση έκτακτων αναγκών σε καταστροφές.

Acknowledgement

I would like to express my deepest gratitude to everyone who supported me throughout the journey of writing this thesis.

First and foremost, I am profoundly grateful to my supervisor, Maria Papadopoulou. Her unwavering belief in my abilities, continuous encouragement, and insightful suggestions were instrumental in shaping the direction and quality of this work. Her guidance went beyond mere supervision, as she motivated me to expand the scope of my research and strive for excellence in every aspect of my thesis. For this, I am deeply indebted to her.

I also wish to extend my heartfelt thanks to Chrysostomos Koumides, who has been not only a family friend and colleague but also a brother, life partner, and best friend. His knowledge, support, and companionship were invaluable throughout this challenging process. His unwavering support made every obstacle surmountable, and his presence was a constant source of comfort and strength.

Lastly, I want to acknowledge the importance of maintaining a balance during this demanding period. Despite the late nights, the challenges of working in two companies, staying active, and nurturing my social life, I am deeply appreciative of the encouragement and love from my family, friends, and one of my biggest supporters, Orestis. Their belief in me gave me the courage and energy to persevere and achieve my goals.

Content

Content of Figures

Content of Tables

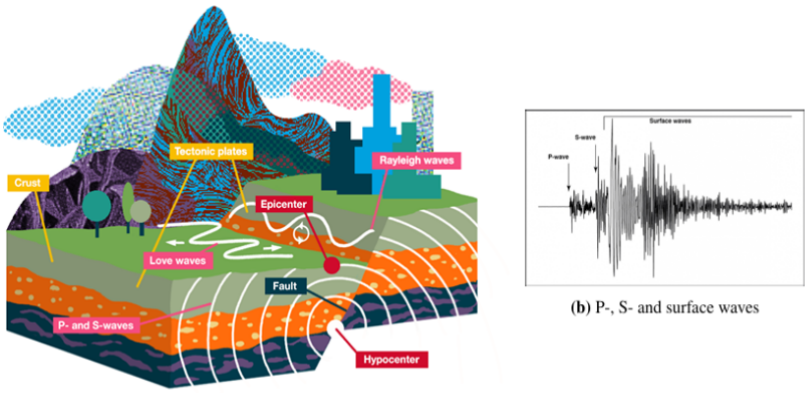
Abbreviations

EWS	Early Warning System
IHU	International Hellenic University
IoT	Internet of Things
MEMS	Micro-Electro-Mechanical Systems
CNN	Convolutional Neural Network
SoC	System on Chip
GPIO	General-Purpose Input/Output
SPI	Serial Peripheral Interface
I2C	Inter-Integrated Circuit
CAN	Controller Area Network
UART	Universal Asynchronous Receiver-Transmitter
SEWS	Seismic Early Warning System
FFT	Fast Fourier Transform
STFT	Short-Time Fourier Transform
SVM	Support Vector Machine
LSVM	Linear Support Vector Machine
LSTM	Long Short-Term Memory
ZCR	Zero Crossing Rate
CAV	Cumulative Absolute Velocity
TauC	Predominant Frequency (TauC)
IQR	Interquartile Range
STA/LTA	Short-Term Average/Long-Term Average
ML	Machine Learning
TinyML	Tiny Machine Learning
MPU6050	MEMS Sensor Module
ANN	Artificial Neural Network

Chapter 1: Introduction

1.1 Seismic Vibrations: An overview

Seismic waves are vibrations resulting from the discharge energy along the geological intersecting cracks, are the main phenomena that cause the earth to shake [6]. These vibrations manifest as seismic waves, are classified into two primary types: P-waves and S-waves, primary and secondary, respectively. P-waves, which come to our knowledge after the earthquake, are transmitted in both materials, liquid and solid. P-waves' period is shorter, and they are normally less destructive [7]. S-waves are usually more catastrophic, as they move only through solids and are responsible for the shaking during the seismic event [8]. Figure 1a shows the propagation of each wave, while Figure 1b the difference vibration acceleration is illustrated.



(a) Propagation of waves

(b) P-, S- and surface waves

Figure 1.1: P and S waves on the Earth's surface [1]

Earthquakes are natural phenomena capable of causing widespread disasters. This can cause the soil surface rupture which may lead to collapsed buildings and infrastructure. In addition to the vibration, earthquakes can inaugurate other disasters such as liquefaction (when the solids act as a liquid) as well as land or mud slides, fires or even tsunamis in case the epicenter is under the water. These repercussions can bring about more problems and make the loss of life and economic losses even more intense [9].

The presence of an earthquake is caused by the earth's tectonic plates such massive, irregularly shaped slabs of the Earth's lithosphere. The movement of the plates cause the formation of stress on the edges of the lithosphere. When this pressure is instantaneously alleviated, it leads to massive vibrations, and even hundreds of kilometers away from the earthquake epicenter, these vibrations can be detected [6]. The magnitude and intensity of an earthquake are used for its severity measurement, which means the amount of energy released and the perceived effects such as land shaking, and human experience while at it.

The seismic sway is an essential diagnostic tool for evaluating the impact of an earthquake. Geologists and seismologists have been facing the problem of predicting the exact moment and the scale of a seismic event because of the Earth's complex karst. Although the technology has greatly improved by predicting where the earthquakes could mostly occur, it is still complicated to calculate the exact time.

Finally, seismic waves are an essential sign of the fission aspects of our planet. Studying the origin and the characteristics of the natural phenomena not only provides information into the global dynamics of the Earth, but is an irreplaceable factor in the ready and the mitigation steps while addressing probably the most unpredictable and destructive element of nature.

1.2 Technological Advancements in Seismic Detection

Before developing that advanced seismic detection technology, it has a time that the concept of earthquake early warning (EEW) systems was created approximately 100 years ago. This system of measuring the earthquakes was based on the identification of the quakes and assessing the size swiftly and correctly to issue immediate warnings that could save lives and prevent huge economic losses. A method of seismic detection used to be mainly through a global network of seismometers; while these seismometers were sending data to a centralized facility, the latter was modulating and sorting it [10].

Nevertheless, the real seismic detection technology has seen a revolution in Internet of Things (IoT) systems and machine learning(ML). The Internet of Things (IoT) propelled by recent strides in mobile computing has resulted in smart systems that are based on MEMS sensors such as accelerometers, and gyroscopes. This is the reason these innovations were implemented in several of the new areas, which are intelligent transportation systems, smart buildings and others, which bring about the greater precision and response time [11], [12].

Along with hardware improvements, ML numbers are among the arsenal of algorithms that decode massive amounts of seismic signal data collected by sensor networks. Independently trained with enough datasets, ML algorithms mimic human pattern recognition, giving good results that traditional interpretation cannot achieve, even at expert level [13]. This attribute is essential in EEW where the rip thing is seismo-detecting in highly noisy environments, which is very challenging [13]. One of the ways that the challenge has been overcome is the correlation of very distinct frequency pattern features that are the main signatures of genuine seismic waves [13].

Nowadays seismic detection systems must process data and determine trends in real time to deal with interoperability challenges and combine the IoT devices, information suppliers and ML models to permit such a task [6]. The TinyML technology, in general, has shown big hope with high chances to be used in seismic detection. As of now, this not quite processed area uses ML algorithms on microcontrollers to develop lightweight yet powerful apps, which run on economical IoT systems which do not use much energy. In addition to smart manufacturing, smart cities and smart agriculture, the systems have proven their use even in another area - smart earthquake detection, thus, speeding up the transition towards more responsive and socially wiser systems [4].

It is marked mostly because of TinyML and the Internet of Things (IoT) technology intersecting that contributes in terms of the seismic detection technology evolution.

1.3 The role of ESP32 microcontroller in Seismic Detection

The microcontroller ESP32 has become a key factor in the development of seismic detection technology thanks to its impressive features that make it a great option for processing data in real-time and communication [14]. In contrast to the ESP8266, the ESP32 is a system -on-a-chip (SoC) microcontroller with a successive set of features such as dual-core processing up to 240 MHz, integrated Wi-Fi, Bluetooth capabilities, and numerous GPIO pins. Integration of multiple built-in sensors, like a Hall Effect sensor and temperature sensor, as well as SPI, I2C, and CAN bus interfaces, among others, enables the ESP32 to communicate with other devices and sensors required to identify and measure seismic activities [14].

In addition, the low power consumption of ESP32 facilitates its operation on a small power supply, like a battery, and therefore makes it applicable in remote seismic activity monitoring areas [14]. This microcontroller's function, which keeps a steady sampling rate, is very important because it helps to capture the intricate vibrations that are characteristic of seismic activity and can be further analyzed in terms of their frequency content to differentiate between normal and abnormal conditions [15].

With the help of the ESP32 in seismic detection systems, developers will be able to achieve compact, energy efficient, and cost effective solutions capable of early warning of natural disasters. The ESP32's wide variety of connectivity options makes it possible to transfer data to centralized analysis centers, or directly to users' devices such as smartphones, smartwatches, or home appliances, thus helping to rapidly distribute alerts [15].

The ESP32 show in this area embodies the general evolutionary occurrence of putting together the uniformly performing microcontrollers with sensors that are specially designed and machine learning models to create intelligent systems that can process and analyze the data in real time. These systems are very useful in designing systems that can detect an earthquake so that early warning is given and so that lives are saved [16].

In brief, the ESP32's high processing capacity, wide range of connectivity options, as well as a less power consuming architecture have proven to be the key-stones of the development of new generation seismic detection devices. The use of autonomous technology in seismic monitoring holds the key to possibly transforming the manner in which we predict and react to earthquakes in risky communities, hence upholding panic and vulnerability on the global scene.

1.4 The Integration of TinyML in Seismic Analytics

Implementation of TinyML into seismic analytics is the great stride forward of ML application to geophysical data processing. Tiny ML, which is the use of Machine Learning algorithms on the low-priced low-power microcontrollers, brings a real revolution for the on-site seismic data analysis [17]. Now, the analytical processing which used to take place in vast data centers can happen directly on the devices that are collecting the data, with the help of TinyML. This adjustment brings multiple advantages, among them the capability to operate by itself without the need of continuous cloud connectivity which is common in remote regions with infrastructure deficit, or areas often hit by seismic waves [17].

TinyML's role in earthquake detection and monitoring is best expressed by the immense potential of this

technology. Having the capacity to store neural network models in a microcontroller's memory and being able to perform inference on sensor data outputs is a fundamental basis for intelligent on-device sensor analytics. This method allows for immediate decision-making, a crucial element in the fast response necessitated during strong shocks. The application of TinyML is also consistent with seismic analytics where the efficiency and reliability of data processing critically affects the success of any disaster response or mitigation.

TinyML's adaptability is reflected in its ability to be used with different sensor-driven scenarios. TinyML in seismic analytics can further increase the precision and accuracy of earthquake detection models, for example using the STA/LTA algorithm for the identification of seismic events with more accuracy and fewer false alarms [18]. On the other hand, the compactness of the TinyML-based systems enables the implementation of a scalable distributed sensor network with more sensitivity of the data obtained [17].

Indeed, TinyML together with state-of-the-art microcontrollers, for example, ESP32, help to process big data sets from MEMS sensors and detect which is a principal element of modern seismic measurement arrays [14]. Earthquakes detection consists of the extraction of time-series and Fourier transforms as tools for this purpose [10].

The use of TinyML in seismic monitoring leads to a provision of instant data analysis and interpreting which are critical for the quick alerting of the at-risk communities through diversified ways of communication such as broadcasting, telephones, radio, and digital means. According if it is through conventional cellular networks, Wi-Fi, Bluetooth, or unique technologies such as ThingSpeak, TinyML assertively makes relevant information to be comprehended and communicated leading to saving lives and the reduction of natural disasters effect [12], [15].

Overall, TinyML in the seismic sensing of earthquakes foretells an important step towards improving the precision in forecasting earthquakes. Working in collaboration with machine learning systems that are not only no longer confined to massive infrastructure but as well can be deployed at the edge where data is captured dramatically enhances not only responsiveness but also the equation of access to sophisticated analytics. This in return is a very capable solution at a low cost that would benefit society greatly.

Chapter 2: Literature Review

2.1 Evolution of Seismic Detection Technology

Seismic detection has experienced breakthroughs since it was first invented a century ago, with research results and technological innovations accelerating the progress of technology. Initially, they used the international seismic network of devices that provided the data, which was daily processed and analyzed at the centralized locations. However, this approach was limited and time-consuming because of the constant need to deliver seismic data to the analysis center [10].

The beginning of a digital revolution in seismic detection resulted in the advent of IoT technologies. Smart systems with built-in advanced MEMS sensors relating to accelerometers and gyroscopes are enabled by the IoT revolution [19], [20]. These systems ensure the high accuracy of the seismic data collection and simultaneously facilitation of the data processing and analysis stages by automatic means. This transition toward the automation of the process and real-time data interpretation is canonically due to the interfacing of machine learning algorithms, that can successfully analyze large volumes of seismic data, look for patterns, and make predictions with minimum human involvement [11], [12].

Additionally, the incorporation of machine learning in the past years, moving to TinyML on low-power microcontrollers, represents the cutting-edge of seismic technology. This approach can perform data processing right at the site, with lower latency, and hence can help save lives by enabling faster response time [21].

The speedy transformation from the laborious manual, centralized techniques to automated and distributed systems shows that seismic detecting is an evolving technology, that is busy inventing more effective, precise, and timely earthquake detection methods.

2.2 Traditional Seismology Instruments

The classic seismology observatories are the pillars upon which the field of earthquake detection has been built. Basic to traditional practices, a seismometerdevice is used to measure vibrations and how far and how much they persist. Seismometers empower researchers to obtain the data that serves as the foundation for calculating the epicenter location as well as the earthquake's magnitude [6].

These devices are normally used to detect the vibrations of the earth after they get amplified to mechanical or digitally sensitive components. Such data is stored in a centralized facility and then goes through the analysis processes. Seismic event detection with traditional seismometers is a well-known fact but to make it work, one would need human intervention to process the data and there is the limitation of the physical connection and infrastructure which can lead to a delay in the timely dissemination of alert information [10].

The technical limitations of the traditional seismic instrumentation led to the emergence of a new generation of digital sensors allowing for instant control and data analysis, facilitating faster response strategies. Such a shift symbolizes a big improvement for both the inefficient and centralized approaches of old seismology and that is how the modern automated devices were developed.

2.3 Modern MEMs Sensors in Technology

MEMS-based sensors have proved to be a good substitute for seismic events detection as they are small and cheap because of low cost of MEMS sensors and also high in sensitivity. MEMS characterizes as the device in silicon that joins mechanical components of micro system consisting of sensors, actuators and electronics and process units of dimensions from 1 up to 100 microns. Others have a central processing unit and some microsensors, which can sense and communicate to the environment in many ways. MEMS devices are small enough to be affected by their environment by the magnitude of the electromagnetism and fluid motion and the movement of the environment that are more intense to MEMS than larger mechanical systems [3].

One of the widely-used MEMs devices in seismic detection field is MPU6050 sensor module containing 6-axis motion tracking units attached with 3-axis gyroscope, 3-axis accelerometer, DMP, 16-bit temperature sensor and the all-in-1 chip form. This module measures velocity, angle/rotation, translational movement and/or rate of change, and acceleration and provides quick feedback for seismic applications [6]. The MEMs sensor is shown to the 2.1.

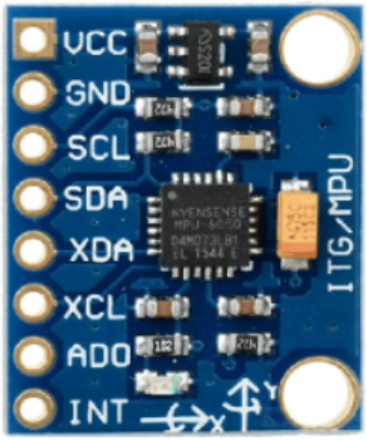


Figure 2.1: MPU6050 sensor [2]

The design of MEMS sensors in seismic monitoring systems has several advantages. They are also considered cost-effective and can be rolled out to a larger portion of the population. They are generally of low signal strength, and are therefore appropriate for deployment in low energy portable seismic monitoring instruments. Also, the MEMS sensors are very sensitive, and it is, therefore, possible to measure an earthquake accurately based on the high sensitivity of these sensors [15].

MEMS sensors are often complemented with other technologies in real life. For example, the integration of accelerometers and gyroscopes enhances the seismic event's detection process because it gives a complete picture of ground motion. The modern MEMS sensors are also made to maintain a constant sampling rate which is ideal for recording the seismic waveforms [15].

2.4 Role of Microcontroller in Seismic Detection

Modern seismic detection systems employ microcontrollers as the computation unit due to the system's need to process data instantaneously, interface with multiple sensors, and deliver information across a network. The ESP32 chip has been of great interest to the users due to the presence of advanced features and efficiency of the microcontroller. Since this low-power SoC also supports Wi-Fi and dual-mode Bluetooth, it has a number of GPIO pins and has a wide range of applications [14].

The ESP32 microprocessor has 2 cores with frequencies up to 240 MHz and task performance for pursuing the computation of seismic data. It is also low power, so it applies in places where power sources are inadequate [14]. Most importantly the ESP32 has a lot of sensors that are built in, such as the Hall-Effect and temperature sensor, that will provide environmental information which is crucial for seismic monitoring.

Frequent seismic activity will portray more details which can only be achieved through constant sampling rates-this is another advantage of the ESP32. It can work on SPI, I2C, CAN, and UART, which makes communication between the sensors and other devices in seismic networks easier [14]. The use of MEMS including accelerometers and gyroscopes can improve the detection output especially in the cases of ground movements [15].

Further, recognition and understanding of real-time processing of machine learning data is implemented through the use of several software development kits that come in handy on the ESP32. This is important especially to seismic detection systems which require pattern extraction as soon as it is developed for the purpose of issuing warnings. TinyML is also used in ESP32 devices wherein models can be uploaded directly to the device which means that the models would not need to be transferred to the cloud every time it is continuously used on a reduced dataset [15].

2.5 Signal Processing Algorithms

Data processing methods are extremely important for seismic interpretation since they should provide a means for effective detection and correct determination of physical attributes of seismic events. One simplest method is STA/LTA algorithm, which monitors the changes in continuous waveforms through the calculation of the relative wave intensity averaged within the small-and-large-time intervals. This proves to be a robust difference in discriminating between the earthquake signals and the background noise, therefore one of the fundamental operations in a seismic network but it is difficult to tune and may also generate false alarms in noisy and congested areas [18, 19].

The Fast Fourier Transform (FFT) is another critical tool, converting time-domain signals into the frequency domain to analyze the spectral content of seismic waves. This transformation is crucial for identifying the location, magnitude, and duration of an earthquake by revealing the frequency components of seismic signals. FFT is particularly useful for filtering out noise, enhancing the accuracy of earthquake detection and analysis [12]. Complementing FFT is the Short-Time Fourier Transform (STFT), which provides a more granular view of the frequency content of seismic signals over time. STFT segments the signal into narrow time intervals and performs Fourier transforms on each segment, producing a frequency-time map that is invaluable for analyzing complex seismic events and understanding their temporal evolution [10].

Wavelet analysis helps to overcome some of the problems that are associated with Fourier methods, because wavelet analysis deals with non-stationery and time varying signals. This technique filters a seismic waveform to give a complete representation of the characteristics of the seismic signal's frequency content over a range of frequencies and durations. Another useful method of wavelet analysis is to identify and record slow earthquakes or volcanic activity [12].

2.6 Application of Machine Learning

This made some seismic activity detection systems more advanced in the analysis and interpretation of seismic data after the embedding of ML algorithms. With such advanced ML tools to identify problematic structures and other specialty impacts, the patterning for enhancement in the precision of detection and prediction of earthquakes can be developed. Current investigation has pointed out that the application of classifiers from machine learning category, including CNNs and GANs, would significantly enhance the quality of seismic signal/noise segregation in EEWs' systems. Such complicated models have been developed to a high degree of accuracy, in the region of 99.5% percent while the recall rates were high as 99.3% against the conventional technique such as the STA/LTA algorithm [4].

So, one of the major breakthroughs in this field is the use of Convolutional Neural Networks, or CNNs. CNNs are computationally efficient in processing seismic data based on spatial hierarchies present in data. It does this by internally reducing dimensionality in the network through pooling layers, even though that may result in the loss of important information and bind the network's performance to some extent [22]. Hence, CNNs remain robust in detecting seismic events because of their translational and rotational invariance.

This means they can be very effective in the detection of earthquakes, even if there are slight changes that occur in the relative spatial relationships of the features [22]. Possibly used timely and correctly is the determination of earthquake magnitude and position minutes after the occurrence of P-waves detected; in fact, it has been observed that CNNs may be embedded with autoencoders. This embedding helps greatly in accelerating and ensuring reliability early warning systems for earthquakes [21]. Another area of innovation is TinyML, used for deploying ML models on ultralow-power microcontrollers capable of executing analytics themselves at the device level. This will significantly enhance how seismic activity is detected from embedded systems in IoT devices. Such TinyML applications run ML models directly on the device—for example, on a microcontroller—and thus enable real-time data processing and decision-making without continuous cloud connectivity. This feature is necessary for seismic activity to be monitored in remote places where there is no full-time internet service at all.

Support Vector Machines (SVMs) and their optimized versions, Linear SVMs (LSVMs), substantially have great importance in categorizing seismic events. SVM classification is such that it attempts to classify using an optimal separating hyperplane between classes in the data. The distance between the data points and the obtained hyperplane is maximized to increase the accuracy of classification and reliability. With the minimization of boundary loss being introduced for LSVMs, the stabilization for the maximization of the boundaries is increased, making them a very reliable tool in carrying out seismic data analysis [21].

Long Short-Term Memory (LSTM) networks are another major innovation suitable for managing long-term dependencies in time-series data. The LSTMs are very able to hold data for a long duration, upon which correct recognition and prediction of a seismic event that spans a more extended period can be done. These can memorize data over a long span and are perfect for analyzing seismic data, which facilitates the exact detection and prediction of seismic activities. These state-of-the-art ML techniques have been beneficial in raising the accuracy and responsiveness of modern seismic detection systems. The systems using CNNs, TinyML, SVM, and LSTM in their deployment can ensure early warning systems with potential efforts in preparedness up to the spiral in mitigating these disasters. Such high-end algorithms make sure that the seismic data are deployed for real-time analysis, in much more sophisticated and faster response at the time of the seismic event. Finally, supports the safety and integrity of the communities in the earthquake-prone regions [[6], [10], [22]].

2.6.1 Introduction to TinyML

TinyML, which is the process of deploying ML models on microcontrollers with limited resources, is a novel concept in the domains of ML and IoT. Tiny ML can be described as a model that is based on the concept of making machine learning models portable and capable of running on devices that might be quite limited in their computational power, memory, and energy. New opportunities for many applications have been created in areas such as environment control, healthcare systems, industrial control, and management systems through the provision of intelligent data analysis at the network edge.

This is because TinyML carries out the task of inference on the microcontrollers themselves and therefore does not need to always rely on links to the cloud. This is especially useful in regions where there are no network connections or where connecting to the network is expensive. For instance, TinyML can just load the neural network model into the microcontroller's flash memory and thereafter make decisions right on site depending on the data that is collected by the sensors at a certain time for instance seismic detection as one of the many real-time applications [17].

This is to mean that the application of TinyML is not only confined to certain situations but rather has numerous applications. For example, in the seismic detection system, applying the ML models on the microcontrollers results in the fast and effective analysis of the seismic data to determine possible earthquakes. Consequently, it can be useful in the early identification of natural disasters and their effects on people through TinyML systems. Therefore, when integrating IoT and ML models, it is possible to develop a complex system that can learn about the environment it is operating in, and respond appropriately with no delay, and this cannot be the same when using cloud computing [18].

Another significant advantage that comes with TinyML is the fact that energy efficiency can be achieved since this is a new technology. The ML models that are commonly used today and integrated into the cloud have a high power consumption since they need huge amounts of computation and data exchange. However, TinyML models are designed to run on microcontrollers which could be powered by batteries or energy scavenging. This low power consumption is favorable for IoT devices in the field as it prolongs the life of the devices and minimizes the need for battery replacement or other forms of upkeep [23].

In addition, TinyML enables the integration of various algorithms that include deep learning models

into small chips such as microcontrollers. These algorithms are essential in data analysis where the analysis can be in the form of images, voice commands, or even the prediction of equipment failures within industries. As a result of the features of TinyML, it can be used by engineers and developers and help create new and innovative solutions that are capable of functioning effectively at the edge of networks [10].

2.6.2 Seismic Early Warning System (SEWS)

Seismic Early Warning Systems (SEWS) are important tools that are developed to help in the identification of earthquakes and give warning of their occurrence. SEWS aims at reducing the loss due to seismic activities as this system provides adequate time for communities and systems to make preparations. These systems use enhanced sensor networks, communication technologies, and complex algorithms to enhance the early identification and communication of earthquakes as shown in 2.2.

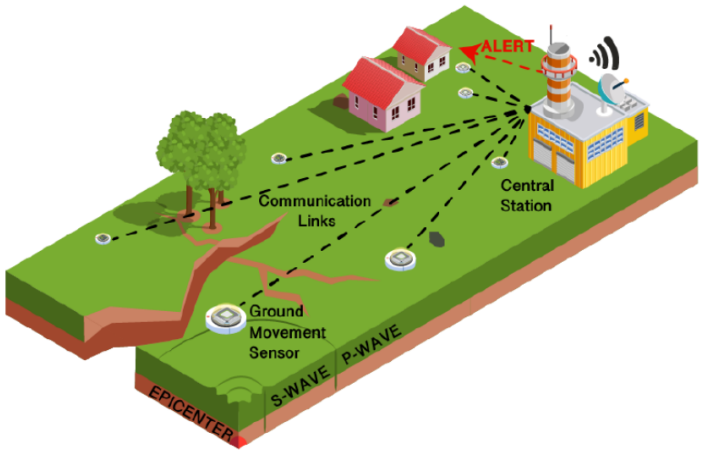


Figure 2.2: Example implementation of an EEWS based on the IoT and P-wave detection [3]

New works with lower sample intervals have evidenced that changes in TEC and ACP can act as other precursors to seismic activity. When incorporated into SEWS, all these geophysical signals, therefore, have the potential of improving the system's ability to forecast earthquakes. For example, previous researches have revealed that TEC anomalies begin a few days prior a major earthquake; hence, such signals if monitored can be used to generate important alarms. Integration of TEC and ACP data in to SEWS would provide a backup to the conventional seismographic techniques of detecting seismic waves for earthquake prediction which may well provide several layers of available data and hence more lead time and accuracy of warning [24].

SEWS mainly employ the identification of the first seismic waves that are produced by an earthquake. The first of these waves is known as the P-waves or primary waves and they move faster than the S-waves or secondary waves. In this way, SEWS can identify the location and strength of an earthquake through the detection of P-waves, which gives several vital seconds to minutes before the S-waves that cause most harm arrive [22]. This early detection is very crucial especially in areas prone to earthquakes such as the Dahshour seismic zone given that the area is prone to earthquakes [9]. Current SEWS systems include several features to improve their functionality. For example, high-pass recursive Butterworth filters are

used to eliminate low-frequency drift from seismic data in order to increase the reliability of the detected signals [25]. Furthermore, the use of IoT devices and MEMS sensors like accelerometers and gyroscopes enhances real-time monitoring of ground vibrations. The MPU6050, a micro-electromechanical system (MEMS) sensor module, can be used for this purpose as it can measure velocity, orientation, displacement, and acceleration [6].

The SEWS has been improved greatly due to the integration of ML algorithms. CNN and LSTM networks are employed for analysing seismic data and distinguishing between real earthquakes and noise. For example, a deep learning framework that integrates an autoencoder and a Convolutional Neural Network (CNN) can pinpoint and estimate the size of an earthquake within three seconds of the P-wave detection [21], [22]. This is done in SEWS using STFT which is used to transform time domain signals into frequency domain signals in order to analyze seismic waves [10].

Another significant problem in SEWS is the ability to identify earthquakes and their efficiency. STA/LTA (Short Term Average/Long Term Average) is one of the most effective techniques in the detection of seismicity. It measures the change in amplitude of seismic waves which helps in differentiating between an earthquake and noise which is way less than the threshold levels [13]. However, there are some drawbacks of this algorithm in choosing the parameters and high false alarms in noisy systems [17]. To overcome these challenges, SEWS usually implement complex filtering and data processing methods to enhance the detection accuracy and reduce false positives [22].

SEWS also use the IQR, ZC, and CAV as features in earthquake detection. These features help in establishing the amplitude, frequency shifts, and energy levels of the seismic signals, which in turn, improves the effectiveness of the earthquake detection models [11], [13]. For instance, IQR measures the range of the extreme values in the middle half of the data set while ZC gives the number of times the waveform crosses the zero level, which may stand for seismic activity [11].

2.7 Related Work

The Internet of Things and machine learning have greatly enhanced the efficiency of the detection of earthquakes in the recent past. An example of this is the study by Karacı in which an earthquake detection system was developed which incorporated vibration sensors with thresholds. This project proved that not expensive and readily available components can be used to design effective early warning systems and therefore such technologies can be taken to communities in areas that are vulnerable to earthquakes [12].

Fauvel et al. have proposed the architecture of the decentralized system for an earthquake early warning system which is based on the GPS stations and seismometers data analysis for seismic events prediction. Their work is based on the prevention of data transmission where Edge/Fog computing is used and the classifiers are deployed at the sensor level. This system achieved high accuracy, thus indicating that data fusion is helpful for improving the effectiveness of seismic detection [3]. This is especially valuable for reducing the latency and increasing the real-time capability; this makes it a valuable addition to the field of seismic activity research.

CrowdQuake is another important work that used an extensive IoT networks of MEMS nodes and a multi-head convolutional neural networks to detect huge volumes of accelerometer data. This specific

system offered high levels of precision and recall in the identification of seismic activity; this system could analyze data from up to 8,000 IoT sensors in just a few seconds [12]. The CrowdQuake project demonstrates how IoT network and ANNs can be utilized to develop extensive seismic network for earthquake early warning and overcoming this natural disaster.

Won et al. proposed a high-fidelity vibration sensor with MEMS accelerometers, a high sampling rate and effective digital filtering scheme. This system had the STA/LTA triggering, and the users were alerted through the Bluetooth Beacon, thus showing that advanced sensors are currently deployed in seismic monitoring [12]. This work aims to focus on the advanced sensors and appropriate data management systems that can help in improving the outcomes of earthquake detection systems to emphasize the importance of the contemporary hardware that is used in the current seismic monitoring.

The ESP32 microcontroller is among the most popular devices deployed in IoT applications such as seismic sensing owing to power efficiency and integration of Wi-Fi/Bluetooth. Due to its portable nature, since it can operate on battery power and its connection options it can be recommended for use in remote surveillance systems [14]. For example, the MPU6050 is a 3-axis gyroscope and 3-axis accelerometer integrated chip, which can be applied for the recognition of seismic noise and eliminating other noises [6]. These elements facilitate the development of small sized and low power systems for the real time identification of earthquakes and differentiating them from other movements.

Chapter 3: Methodology

3.1 Introduction

In this chapter, we discuss the methodologies employed in designing a reliable seismic activity recognition system utilizing the ESP32 microcontroller and MPU6050 sensor. This system is intended to observe vibrations in buildings and differentiate between typical oscillations and actual earthquakes. By integrating hardware components, data acquisition methodologies, and deep learning (DL) models, the approach aims to deliver both accurate and efficient seismic detection.

The primary objective of this work is to tailor the methodology for the effective utilization of the MPU6050 sensor for motion tracking and vibration analysis. The accelerometer functions of the sensor are crucial as they record real-time data necessary for analyzing activities related to seismic events. To enhance the accuracy of detection, we integrate TinyML principles, such as Convolutional Neural Networks (CNNs), and execute the test part on the microcontroller.

This chapter provides an overview of the selection of the MPU6050 sensor, emphasizing its sensitivity and compatibility with the ESP32 microcontroller. Additionally, it explains the algorithm for earthquake detection, incorporating CNN models. These models process overlapping windows of data from a sensor, helping to distinguish seismic signals from noise. Using a high pass filter to average data over short and long periods, reducing noise and improving detection performance.

Furthermore, this section explores how feature engineering techniques enhance the framework's detection performance. When integrated within a TinyML system, these methodologies provide a small, low-power, and promptly responsive system suitable for deployment in different building types.

The following sections describe the overall research methodology, including the data acquisition methods, model training, component used, software used, and implementation strategies. This structure ensures that the project proposes an efficient and effective system for seismic event detection, supporting warnings and enhancing the structural integrity of buildings.

3.2 Research Approach

This annihilative study incorporates both experimental enmity research undertakings and case study approaches. This option ensures that separate yet related evaluations of the technological progress in the seismic detection system as well as the efficiency of utilizing the system are possible. In the case of the experimental strategy, the concentration is made on creating control and structure for the system constituents and developmental algorithms. In addition, the case study approach offers a controlled environment for practice assessment.

The investigation commenced with a review of the literature, using the method of systematic search and samples run-through. Articles relevant to key issues and trip observations were saved in Google Drive, while notes were recorded in the Google Docs document. The literature review involved identification of existing models, algorithms in seismology, previous machines used, other methodological works, and repetitive problems and factors in affiliated studies and research. This in-depth appraisal meant the

right choice of the MPU6050 sensor, ESP32 microcontroller, and the proper dataset. Deciding on the appropriate dataset was the most time-consuming step, which entailed the scanning of several datasets with the aim of choosing the optimum set needed in the research.

Therefore, when it was identified that the ESP32 microcontroller and MPU6050 sensor would be suitable components, the components were wired to form the basic hardware for the seismic detection system. Basic flow control functions were defined to process data from the MPU6050 sensor using the high-pass filter algorithm meant for the first passage of the seismic events. To differentiate between normal oscillations and earthquakes, CNNs were then used. Thus, the CNN models were fine-tuned and used in the ESP32 microcontroller as a part of TinyML, which also employs model reduction and size reduction techniques.

In order to verify the efficiency of the developed system, some basic tests were made with an active stimulation of the sensor in order to simulate the earthquake. The specific outcomes of these experiments included studying the system's patterns of reaction to various sorts of vibrations and its capacity to categorize them correctly. Subsequently, the data gathered from these controlled experiments were carefully preprocessed and evaluated in order to determine the ability of the aforementioned system to distinguish between ordinary vibrations and earthquakes.

Therefore, this mixed-methods approach affords the research a more holistic assessment of the seismic detection system and the circumstances required for its utilization as well as management.

3.2.1 Rationale of Research, scope and limitations

The combination of quantitative and qualitative approaches is relevant given the necessity of a broad investigation of the problem, as well as the multi-aspect nature of the formation of the system on the basis of the identified activities that differ in technical and organizational approaches. The given experimental setting enables direct control and adjustment of arrays and systems' characteristics and algorithms to guarantee technical reliability. The application of the case study extends the setting and establishes the reality check of the structural efficiency in controlled setups at large. More importantly, this dual analysis is essential when regarding the research questions and objectives of the study.

Optimizing the CNNs for TinyML guarantees efficient processing of the data on the target devices without requiring the use of cloud resources. The case study further confirms the practicability and efficiency of the system since it re-creates field conditions, for example, flipping the sensor by hand to obtain information. Thus, the chosen methodology allows to exclude of random vibrations and outline the clear steps to reach the main objectives of the work and to propose an effective solution in the frame of the developed system.

The activities within this research encompass the complete design, coding, calibration, and implementation of the seismic detection system incorporating the ESP32 microcontroller and the MPU6050 sensor. This includes the choice of components, their integration, data algorithms, and optimized CNN models using TinyML techniques. The scope also includes evaluating the system's capability to distinguish between normal building vibrations and earthquakes, aiming to build a reliable and effective seismic monitoring system for different buildings without heavy reliance on cloud computing.

A major challenge encountered was the nature and quality of the datasets. Acquiring relevant datasets for the MPU6050 sensor was difficult, with many datasets being inaccessible. At least six datasets were reviewed and analyzed for their reliability and applicability. These challenges highlight the difficulty of establishing a robust seismic detection system and emphasize the need for continuous improvement and testing.

3.3 Data Collection

In the course of the first data-gathering stage, several datasets were investigated to identify the one to be used for training of the seismic detection model. First of all, the COSMOS Virtual Data Center [18] dataset was chosen because this dataset contains a large amount of seismic data. However, it soon turned out that this dataset was small, and the downloading process proved to be rather difficult in this case. Since this was the case it became impossible for the project. Exploring further, the Southern California Earthquake Data Center (SCEDC) [4] data set was found. Helpful information concerning the use of this particular dataset could also be gained from an article that was included as part of the material. The measure of the dataset of SCEDC contained 374 ,000 records of local earthquakes and about 946,000 impulsive noise records that ensured the reliable thresholding training. The study conducted on this dataset demonstrated compelling findings with the ENZ components, which coincided with the capacities of the MPU6050 sensor.

Primary Data: The primary data of this study is therefore the preliminary testing that was done using the MPU6050 sensor. The types of tests carried out consisted of physical relocation of the sensor to study the extent of accelerations it could capture and in what form, that is, along the x, y or z axis. This experimentation proved useful in ascertaining proper algorithms for analyzing the sensor data and as a means of evaluating the applicability of this sensor in the seismic detection system.

Secondary Data: There is only one source of secondary data in this paper, namely the dataset and the methodologies mentioned in the article titled Reliable Real-Time Seismic Signal/Noise Discrimination with Machine Learning by Meier et al., 2018. This considerably helped in the training of the deep learning (DL) model which contains 374,000 local earthquake records and 946,000 impulsive noise signals. The findings and information from this article were useful in redefining the detection algorithms and evaluating the system's effectiveness compared to best practices in seismology.

3.3.1 Sampling Method

The sampling techniques used in this research included a process of selecting data in relation to the mentioned above set. Considering the need to come up with an efficient dataset sample to serve as a benchmark, the current one includes particular characteristics of the structures under consideration, including peak acceleration (pa), zero crossing rate (zcr), skewness, kurtosis, cumulative absolute velocity (cav), the vertical-horizontal ration (zhr), as well as the predominant frequency (tauC) which has been computed using various time intervals (before the onset of impulsive signals Such attributes were important for the training of deep learning model so as to distinguish between the events and noise [4].

The samples taken from this dataset can be consider biologically relevant to the seismic phenomenon

under research as it includes samples of different earthquake magnitude, hypocentral distance and different type of impulsive noises. This is essential for developing algorithms that can be generalized to real-life earthquake situations. To support the data and the efficiency of the selected features, some seismograms of quake and noise and their N, E, and Z components are presented in figures 4 and 5; samples of these features are 15 for quake and 15 for noise. These will be the numbers that will illustrate how various characteristics of the signal and noise can differ and that the algorithms rely on these to classify between signals and noise. These visual illustrations will also help to prove the stability and efficiency of the dataset and the developed detection system.

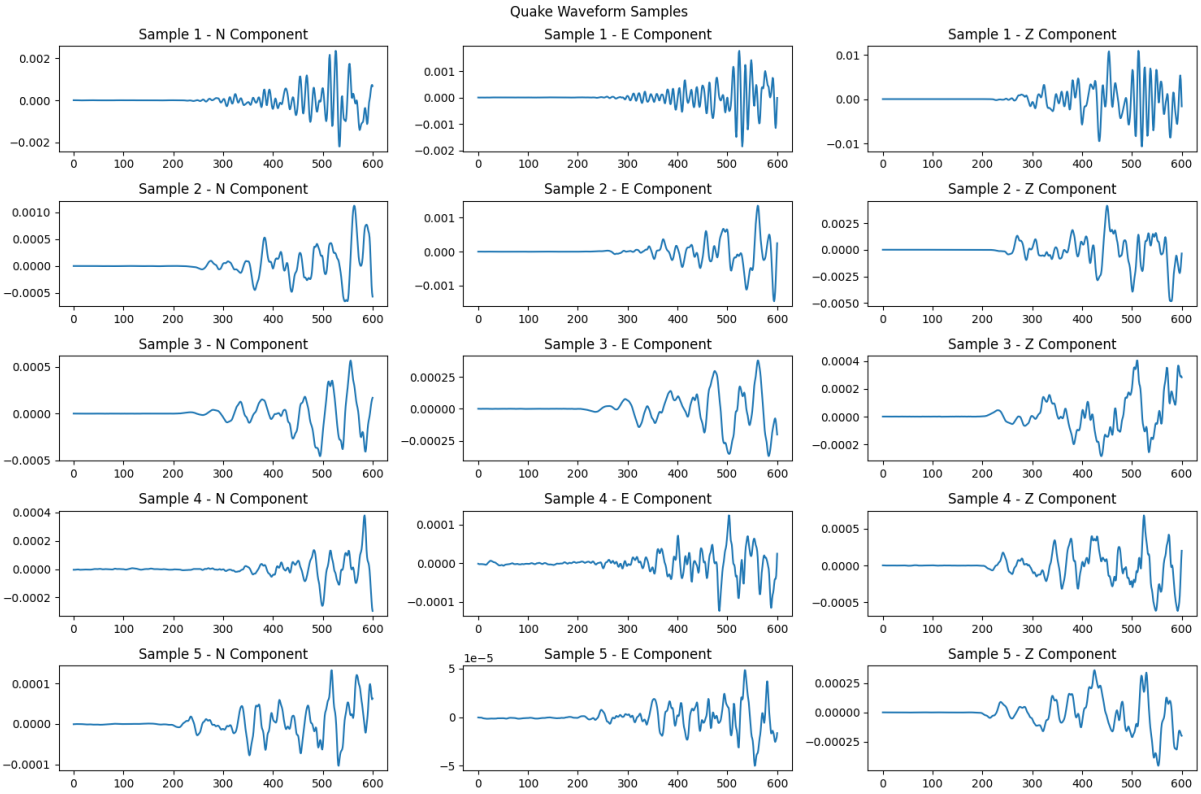


Figure 3.1: Quake Waveforms Samples [4]

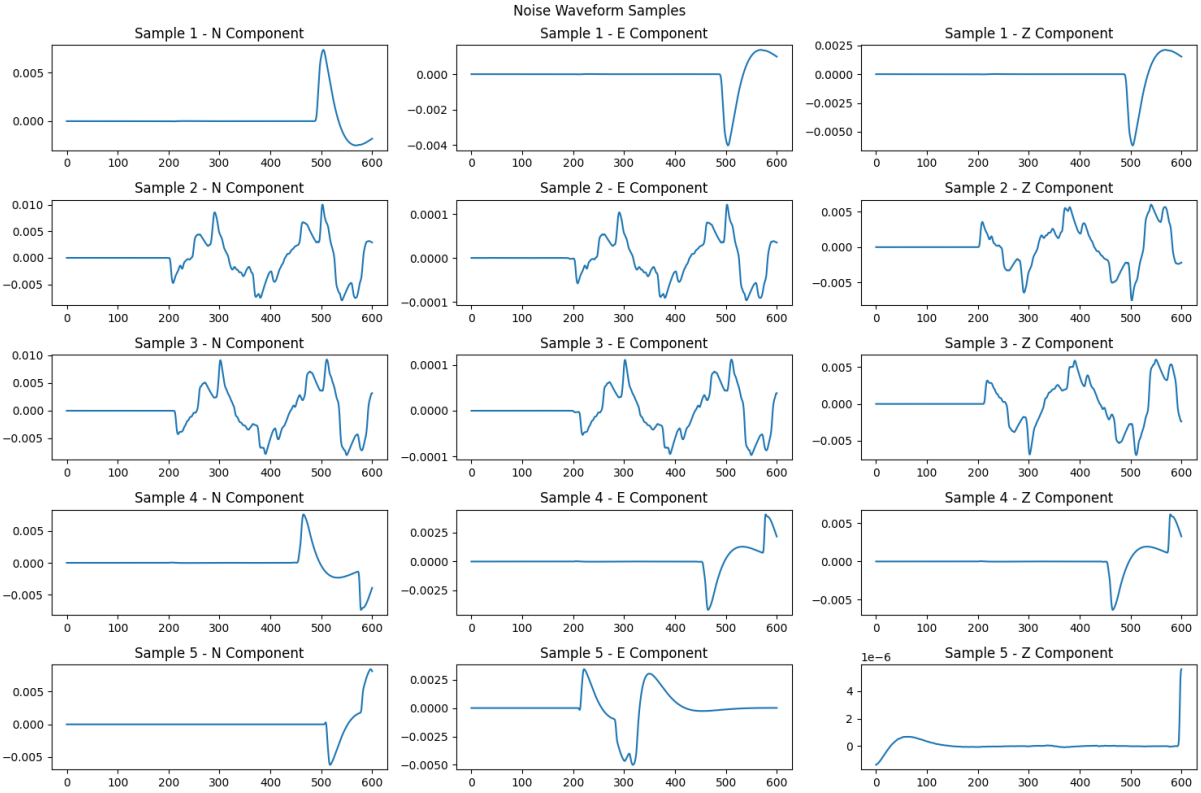


Figure 3.2: Noise Waveforms Samples [4]

3.4 Data Analysis

The applied analytical techniques of this research were primarily more inclined toward the analysis of the gathered seismic data to optimize the mechanisms of detection. In the first steps, the data preprocessing, feature selection or extraction and model training steps were performed. Preprocessing consisted in cleansing and standardizing the collected raw data to minimize the level of errors. Preprocessing was done in this stage to extract meaningful characteristics related to the seismic signals to be used by the machine learning algorithms.

For the analysis of the data, onsetWforms_meier19jgr_pub1_0_woJP.h5 [18] was used. This file consists of information from local records of earthquake as well as impulsive noise signals. HDF5 or Hierarchical Data Format version 5 is actually a file format and several tools for storing heterogeneous data. HDF5 is applied in scientific computing because it enables ease in storage, extraction and publication of scientific data in an efficient manner.

An HDF5 file is created so that it can store lots of data on other elements of the experiment in a more efficient manner. It has a classical and well-ordered structure – it can introduce order in groups and datasets similar to how it is done in the file system: each group is a directory, and a dataset is a file. These include data structures with a hierarchical structure, efficient storage through compression, field versatility so as to accommodate arrays of different types of data and system portability. These attributes make the HDF5 very ideal for applicability in large and complicated data sets such as those used in seismic investigations.

The data exploration was carried out by trying to open the HDF5 file, attempting to get the feature names of both quake and noise and finally print out the structure of the HDF5 file to see how the data was arranged. The first of these was to become acquainted with the dataset and to map out future procedures in this step.

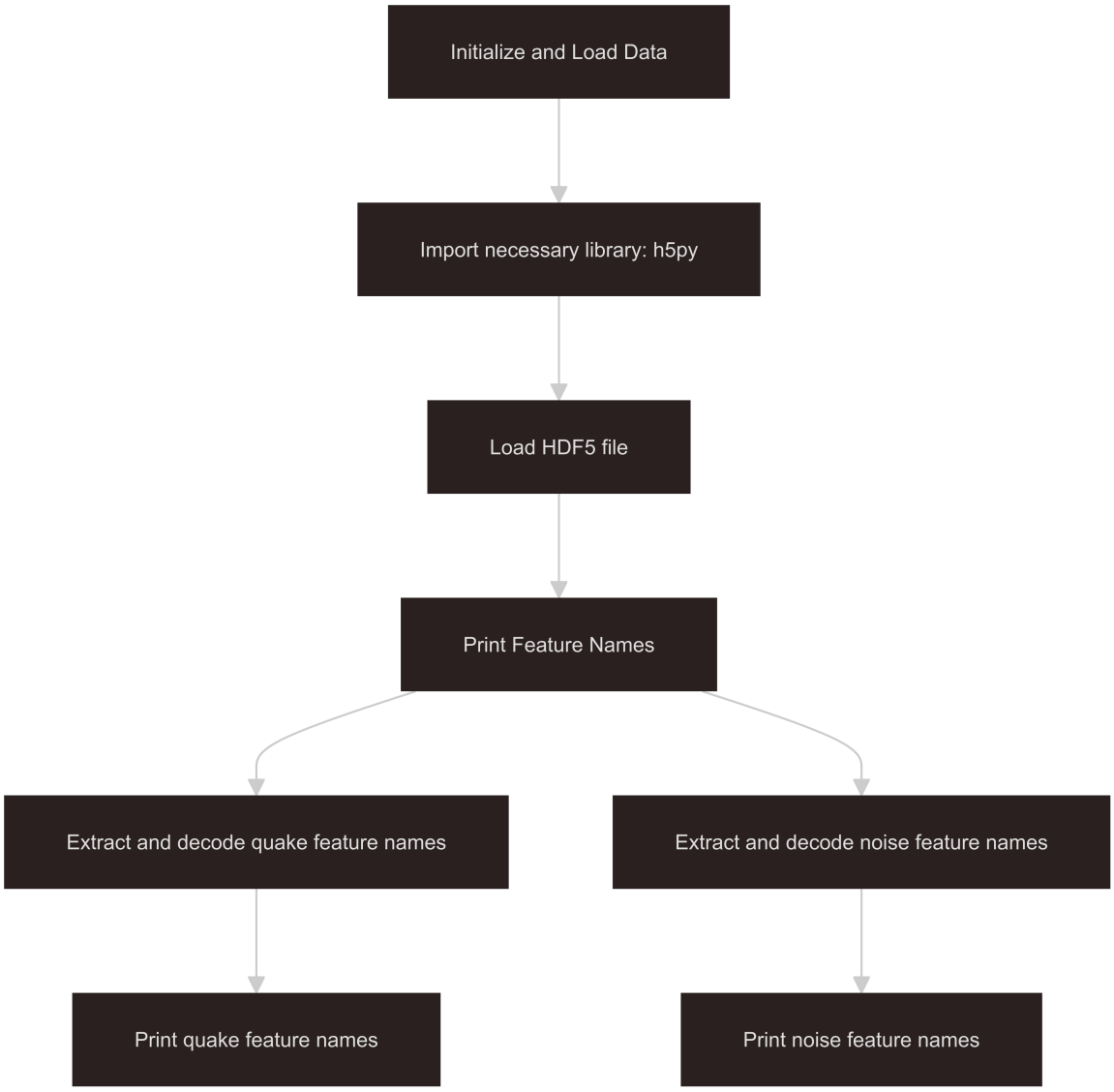


Figure 3.3: Initialize and Load Data, Print Feature Names

The next task was to get raw waveform samples and print them to analyze the waveform data in the next step. This was done by choosing sub samples from the quake and noise waveforms and then printing them for visual analysis. Hence, the raw samples which have been collected were visualized to better understand what the data was like and if even at this stage, preprocessing would be required.

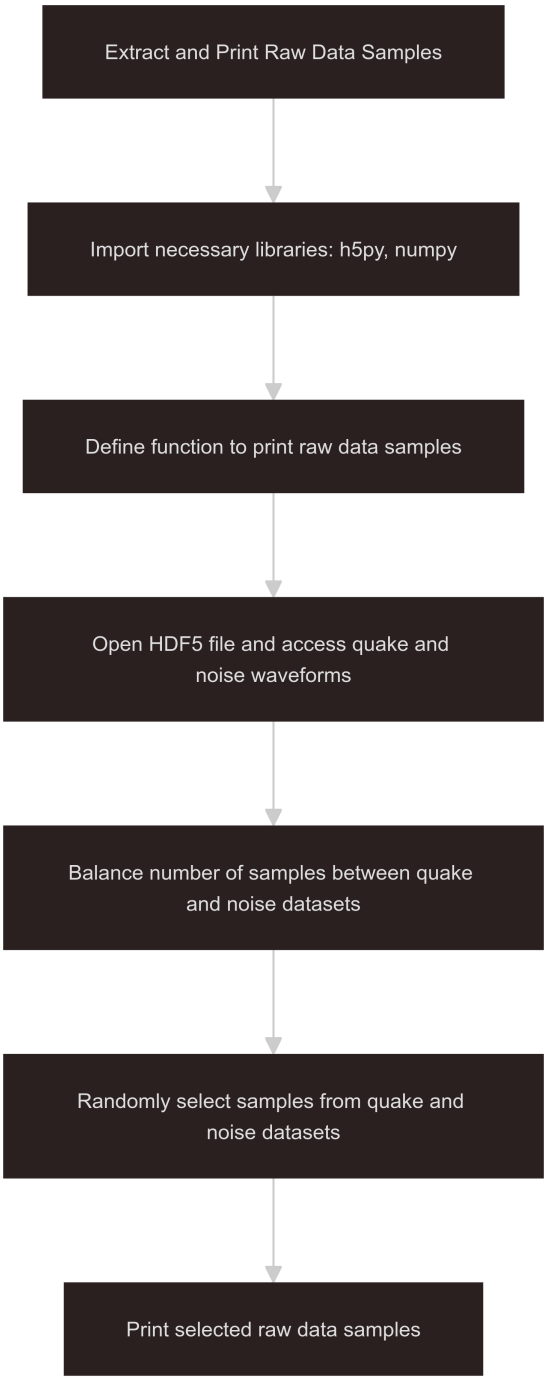


Figure 3.4: Extract and Print Raw Data Samples

Next, waveform plots were produced in order to compare visually the contents of quake and noise samples which is shown in fig. 4 and 5. This was done following the waveforms of different components N, E, and Z for a multiple of samples where it was possible to identify the difference and similarities between the events and the noise which helped a lot in feature selection and model selection.

3.4.1 Feature Engineering

In this step, feature engineering to extract some features from the obtained dataset would help feed the model with the best and most useful information that would assist it in relating between seism activities and background noise. In other words, these features contain aspects of the seismic signals that the model is most likely to pick out separating between true and false seismic movement. Some of the features I used to plot to see the difference was the following:

- **Peak Acceleration (pa)** is the change or 'amount of effect' that is imposed per unit time during an earthquake. This one is used when determining the magnitude or strength of the vibrations. Regarding this feature, it has the objective of judging the likely impact of the seismic occurrence as it picks the maximum amount of acceleration. The formula (3.1) explains the necessary calculation to compute the final value:

$$\text{PGA} = \max(\sqrt{a_x(t)^2 + a_y(t)^2 + a_z(t)^2}) \tag{3.1}$$

where $a_x(t)$, $a_y(t)$, $a_z(t)$ from the 1.1 formula are the acceleration components in the x, y, and z directions at time t.

- **Zero Crossing Rate (ZCR)** illustrates the number of points in the signal in which the sign of the signal turns from positive to negative or from negative to positive. This feature proves more useful when one wants to differentiate between one or several vibrations. The zerocrossing rate increases as the signal changes frequently, is more amped up or as it includes faint vibrations; the rate decreases as in a large, more stable event. The following formula (3.2) explains how ZCR is calculated:

$$\text{ZCR} = \frac{1}{T}\sum_{t=1}^{T-1} 1\{x(t) \cdot x(t-1) < 0\} \tag{3.2}$$

  where:

  - T is the total number of samples.
  - $x(t)$ is the signal value at time t.
  - $1\{A\}$ is an indicator function that is 1 if the condition A is true and 0 otherwise.

- **Skewness (Skew)** refers to the amount of departure from the symmetry of the data distribution. It helps to determine the nature of the seismic signal. Positive skewness suggests that the distribution is positively skewed, meaning that it has a longer tail on the right side, while negative skewness indicates that the distribution has a longer tail on the left side. This feature gives information about its distribution, and if it departs from the normal distribution, this could be helpful in distinguishing between normal and abnormal seismic activity. The following formula (3.3) shows how this feature is calculated:

$$\text{Skew} = \frac{N}{(N-1)(N-2)}\sum_{i=1}^{N}(\frac{x_i - \bar{x}}{s})^3 \tag{3.3}$$

  where:

  - N is the number of data points.
  - x_i is the i-th data point.
  - $\bar{x}$ is the mean of the data.
  - s is the standard deviation of the data.

- **Kurtosis (Kurt)** portrays how drawn out the tails of the data distribution are. High kurtosis might indicate the presence of outliers, which is useful for detecting seismic events. Kurtosis assists in determining if the data contains heavy tails and outliers, which are apparent in seismographic activity in view of sudden, violent oscillations. The following formula (3.4) calculates kurtosis:

$$\text{Kurt} = \frac{N(N+1)}{(N-1)(N-2)(N-3)} \sum_{i=1}^{N} \left(\frac{x_i - \bar{x}}{s}\right)^4 - \frac{3(N-1)^2}{(N-2)(N-3)} \tag{3.4}$$

  where:

  - N is the number of data points.
  - x_i is the i-th data point.
  - $\bar{x}$ is the mean of the data.
  - s is the standard deviation of the data.

- **Cumulative Absolute Velocity (CAV)** represents the total accumulated velocity over time. It is useful for quantifying the energy of the seismic event. This feature involves the incorporation of the absolute value of velocity and essentially gives the total energy released during the event, which is very useful in evaluating the extent of the event. The following formula (3.5) explains the variables and the calculation of the CAV:

$$\text{CAV} = \int_0^T |a(t)|\, dt \tag{3.5}$$

  where:

  - T is the total duration of the seismic event.
  - $a(t)$ is the acceleration at time t.

- **Vertical/Horizontal Ratio (ZHR)** is used to indicate the proportions between the impacts of the vertical and horizontal movements of the seismic waves. This feature is useful in identifying various seismograms and their directions of propagation, which is pivotal in event description. The following formula (3.6) explains the variables used in ZHR:

$$\text{ZHR} = \frac{A_{\text{Vertical}}}{A_{\text{Horizontal}}} \tag{3.6}$$

  where:

  - A_{Vertical} is the amplitude of the vertical component of the seismic wave.
  - $A_{\text{Horizontal}}$ is the amplitude of the horizontal component of the seismic wave, typically calculated as the root mean square (RMS) of the north-south (A_{NS}) and east-west (A_{EW}) components:

$$A_{\text{Horizontal}} = \sqrt{A_{\text{NS}}^2 + A_{\text{EW}}^2} \tag{3.7}$$

- **Predominant Frequency (TauC)** identifies the dominant frequency of the seismic signal. It assists in categorizing the nature of the vibration. Indication of the highest frequency helps in identifying the fundamental frequencies that describe the main characteristics of the seismic event to distinguish between different categories of seismic activity. The following formula (3.8) explains the variables used for its calculation:

$$X(f) = \sum_{n=0}^{N-1} x(n) \cdot e^{-j2\pi fn/N} \tag{3.8}$$

  where:

  - $X(f)$ is the Fourier Transform of the signal $x(n)$.
  - n is the sample index.
  - N is the total number of samples.
  - f is the frequency bin.

All these features are computed for multiple time intervals resulting to the temporal and spectral characteristics of the seismic signals. All the features play a different role in helping the model to differentiate between normal vibrations of buildings and actual earthquakes. The analysis and interpretation of the two parameters are as follows: the peak acceleration shows the signal strength during the occurrence of the events, and the zero-crossing rate helps in recognizing a change in signal frequency. The measures of skewness and kurtosis are a helpful means to drive attention toward the shape of distribution of the data. It is collected by integrating velocity and the vertical to horizontal ratio gives information on directionality of waves. Also, the main period has the biggest amplitude and describes the main peak frequencies or the fundamental frequencies of the vibrations.

The event progression also allows the model to carry out feature engineering on raw sensor data and produce better quality inputs for the seismic detection system thus, enhancing the system's performance. The next steps will include use of these engineered features in machine learning algorithms to arrive at a comprehensive and efficient seismic detection system.

These features in waveforms will be further discussed and interpreted in Chapter 5 as part of the result where figures depicting the extracted features and their corresponding explanations will be presented to show how these features are useful in identifying seismic events.

3.5 Software and Hardware Tools

More interestingly, all the tools were helpful in both the processing of the collected data and the writing of the code for the system. For data analytics, tools were employed to preprocess, analyze and visualize the collected data for building good algorithms for detection. In the case of the embedded system, particular tools allowed the coding and execution of the code on ESP32 to constantly detect seismic activity. The primary programming languages of this project were Python for data analytics and C++ for programming the microcontroller. It is indicated below the specific software tools that are used in each of the phases in the project:

Data Analytics Tools: Regarding the data analytics, Google Colaboratory (Colab) [26] was mainly utilized. Google Colab is a service of Google, it is actually a Jupyter notebook running in the cloud and designed for sharing and collaboration on data analysis. The following libraries and tools were used within Colab, leveraging Python as the primary programming language:

- **Pandas**: Recognized, adaptation, and use of structured data; especially data selection, summarization, and alteration.
- **NumPy**: Procedure employed when doing arithmetic calculations or when dealing with large amounts of data.
- **Matplotlib and Seaborn**: Used for data visualization; supports the creation of various plots and graphs for data and analysis results.
- **Scikit-learn**: Used to build, evaluate, and fine-tune models employing various methods of machine learning.

Embedded System Programming Tools: For programming the embedded system, the following tools were utilized, using C++ as the programming language:

- **Visual Studio Code:** Used as the code editor, enhanced with extensions for embedded system development [27].
- **PlatformIO:** An active growth environment, based on VS Code and tied with the authoritative software that controls the IoT production, the undertake of the build, the library keep, and the embracing of microcontrollers such as ESP32 [28].

3.5.1 Hardware Tools

Microcontroller shall be procured first in order to establish the support for the rest of the electronic components in this project. After the comparison of several boards, ESP32-S3-DevKitC-1 was chosen because the device is capable of running TinyML. The ESP32-S3-DevKitC-1 is based on the ESP32-S3 System on a Chip (SoC), which offers several advantages:

- **Versatility and Performance:** The decision makers of ESP32-S3 has devices with a double Xtensa LX7 core, The computation of TinyML models can sufficiently be handled by the available double Xtensa LX7 core in ESP32-S3. It is fast in its processing capabilities thus, data processing and model inference can be done in real-time something that is important for asteroid impact detection.
- **Memory Capacity:** ESP32-S3 has sufficient integrated SRAM up to 512 KB and Flash at most of 16 MB that one requires to pursue TinyML model. This serves to help in loading, executing and handling the algorithms in machine learning not to mention about the regular issue of memory overflow.

- **Integrated Wi-Fi and Bluetooth**: The built-in Wi-Fi is used for transmission and monitoring through a remote system, and Bluetooth. This connectivity also enables real-time alerting/updating of seismic activities to a central server or directly to users.
- **Multiple I/O Interfaces**: The ESP32-S3-DevKitC-1 has many GPIO pins and interfaces like I2C, SPI, and UART, making it suitable for many sensors and peripherals. It is particularly beneficial for evaluating the MPU6050 sensor and other possible modules that could be incorporated into the circuit

More information regarding the input/output interfaces and characteristics of ESP32-S3-DevKitC-1 is illustrated in Figure 8. It is extended with circuit diagram showing the pin layout where is shown the GPIO pins, power supply pins and communication interfaces. However, It also displays the Wi-Fi and Bluetooth integrated modules which are important for data transfer through wireless media.

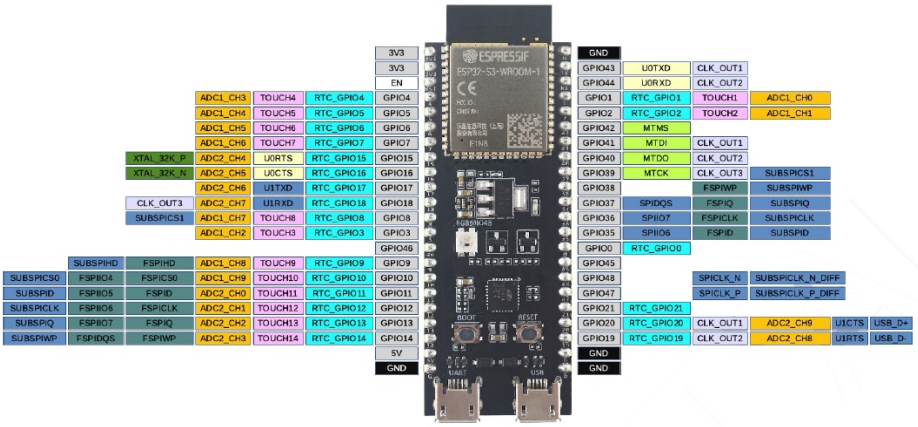


Figure 3.5: ESP32-S3-DevKitC-1 Model pins showcase [5]

As for obtaining information on seismic events in the environment, the MPU6050 sensor was selected as the primary source of information. This sensor includes 3-axis gyroscope and 3-axis accelerometer which is suitable for measurement of motion and vibration parameter. Key features of the MPU6050 include:

- **6-Axis Motion Tracking**: Accelerometer function is combined with the gyroscope one, both are essential to register all the motion which is significant for the seismic movement analyses.
- **High Sensitivity and Precision**: The break-frequency of the MPU6050 is very high and due to this the device is very sensitive to vibrations and provides data in millimeter. It is pinpointed specifically to distinguish the normal structure vibrations from real quake occurrences.
- **I2C Communication**: To make communication with the ESP32 microcontroller, there is a use of slavery by the sensor that makes wiring easy through I2C protocol. It also has available I2C interface that allows to connect multiple devices, so there is option add more devices into the system in the future.
- **Built-in Digital Motion Processor (DMP)**: MPU6050 has an internal DMP which can be used to calculate more complex motion related algorithms such that their computation will not be done

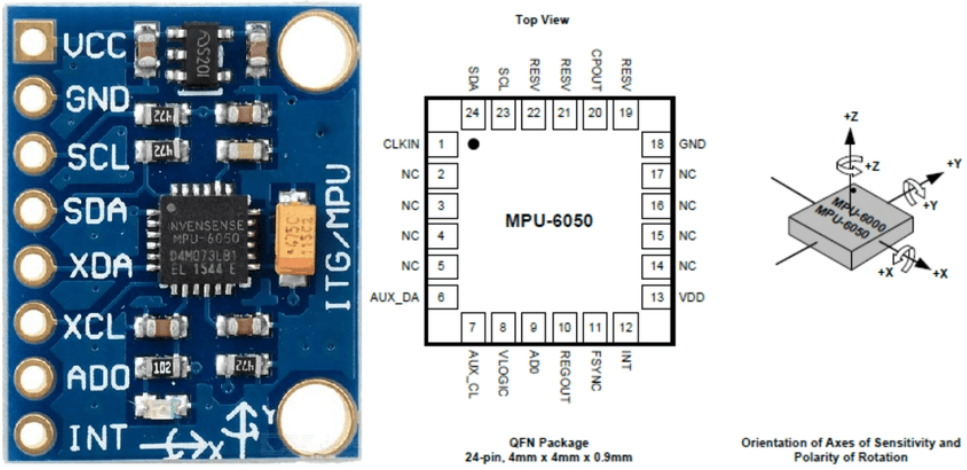


Figure 3.6: MPU6050 pins and Axis showcase [2]

using the microcontroller. This feature assists in raising the system response rates hence making the system more efficient.

For the detailed visual representation of the sensor, particularly MPU6050, you can find the layout and features in the Figure 9. This figure illustrates the orientation of the axes of sensitivity and the polarity of the rotation of this sensor together with their pin configuration and the top view.

3.6 Models and Algorithms

3.6.1 Convolutional Neural Networks (CNN)

The main artificial neural network used in this research work is the Convolutional Neural Network (CNN). CNN is a kind of deep learning algorithm, which is very suitable for solving problems that have a grid structure, for image data, time series data, such as seismic signals [29]. Here is a detailed explanation of the main components and functioning of a CNN:

- **Input Layer**: This layer takes the raw data as an input. In the case of seismic detection, this input could be acceleration which is collected by the MPU6050 sensor and is stacked in the form of time series data.
- **Convolutional Layers**: These layers perform convolution on the input data with the help of filters (also called kernels). Every filter, in turn, is dedicated to the identification of certain characteristics in the data; it could be edges in the image, texture, or certain patterns in the time series data. The filters move over the input data and always consist of element-wise multiplication and summation, which results in the feature map.
- **Mathematical Operation**: If X is the input matrix and K is the filter, the convolution operation can be represented as:

$$(X * K)(i,j) = \sum_m \sum_n X(m,n)K(i-m, j-n) \tag{3.9}$$

where (i, j) represents the position in the resulting feature map.

- **Activation Functions**: After each convolutional layer, an activation function is applied to introduce non-linearity into the model. The most commonly used activation function is the Rectified Linear Unit (ReLU), defined as:

$$f(x) = \max(0, x) \tag{3.10}$$

  This helps the model to learn complex patterns by adding non-linearity.

- **Pooling Layers**: Due to the downsampling of the feature maps, the pooling layers also reduce the amount of computation, hence, decrease the probability of over-fitting. The most popular one of these pooling methods is Max-Pooling, which defines a set of values in the feature map and gives a maximum value out of that set of values.

$$Y(i, j) = \max\{X(m, n) \mid (m, n) \in \text{pooling region}(i, j)\} \tag{3.11}$$

- **Fully Connected Layers**: The network in this case has several layers involving convolution and pooling while higher-level decisions are arrived at in the fully connected layers. In the fully connected layer, each and every neuron is connected to each and every neuron of the previous layer to combine the learned features to make the final decision.

- **Output Layer**: The last layer normally consists of a softmax/sigmoid nonlinearity if the output of the model is a probability distribution over classes. For binary classification, the sigmoid function is used:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.12}$$

- **Loss Function and Optimization**: In the CNN training process, a loss function is used to determine the degree of error of the obtained result compared to the desired one. Some of the commonly utilized objectives are Mean Squared Error for regression problems or binary cross-entropy for binary classification problems. The network parameters are adjusted with the help of methods such as Stochastic Gradient Descent (SGD), Adam, or any other hyper-optimization method.

3.6.2 Tiny Machine Learning (TinyML)

Implementations of learning models to smart, resource-scarce devices are called the TinyML models, and it involves the process of working with big data models on devices with microcontrollers. The goal here is to perform inference at the edge, close to where the data is generated, so that there is minimal latency and less dependence on cloud solutions. Key concepts of TinyML include:

- **Model Quantization**: Rounding them off to 8-bit Integers; Floating point numbers. These are real numbers present between 0 and 1, which are used to define the nature and the values of the model. This leads to a decrease in model size, improvement in the efficiency of inference, and only a very small, almost negligible, decrease in accuracy.

$$\text{Quantized value} = [\text{floating point value} \times \text{scale} + \text{zero point}] \tag{3.13}$$

- **Efficient Inference**:

  Modifying such types of inference for low-end and resource-constrained devices that may feature some electronics, for example, wearables. This necessarily implies minimizing the scalar operations to the absolute minimum and using the specific hardware facilities of microcontrollers.

  Memory Management: This involves ensuring that the obtained model and all its computations can be implemented within the restricted amount of RAM available on microcontrollers. Therefore, model slimming, which implies pruning unnecessary parameters, and proper memory management are important.

  Low-Power Operation: Designing it to run on battery-powered devices with a limited power source. This is done in two ways: by reducing the computation and by using low-power modes of the microcontroller.

  On-Device Learning: Some of the complex TinyML applications may include the overall ability of the model to enhance its learning capabilities on the device itself, sometimes without requiring a connection back to the server.

  Using CNNs with TinyML: This work will enable the identification of seismic activity in real-time and with high accuracy on the base ESP32 microcontroller. This approach builds upon the strengths of CNNs in pattern recognition and the compactness of TinyML, providing a self-sustaining seismic monitoring system.

Chapter 4: System Design and Implementation

4.1 Introduction

Broadly, the objective of this thesis is to examine the complexity of earthquake occurrences with the view of bringing down their effects such as collapse of structures and deaths. This project aims at creating a device that is capable of performing this thinking without necessarily requiring the cloud connection, whether the building can withstand seismic waves or is already affected by other vibrations. This aspect is important because it enables the construction of safer structures and rectification works in areas that are vulnerable to earthquakes.

It claims to be a turnkey project in which the design and installation of a state-of-the-art seismic detection system are to be done. This system also employs the ESP32 microcontroller and MPU6050 sensor and applies TinyML for instantaneous data processing. The idea here is to come up with a gadget that would sense and give real-time information on the earthquake, so that the possible effects can be reduced and the reactions time of the emergency services increased.

At the heart of proving this research, there's axial pragmatic application. To meet the system's necessity and reliability, a PCB was designed and additionally improved for more than four revisions. This practical mentality stresses on the concept of trial and error and learning by doing. This validation process of the SCEDC dataset involved creating waveforms that seemed to resemble real earthquake activities and, also creating waveforms that appeared similar to the ESP32 and MPU6050 during earthquake and noise mimicking. Thus, the prospective use of the presented waveforms is confirmed by a successful implementation in the field of seismic detection systems.

Thus, the present work is not only useful in expanding the knowledge base concerning seismology, but also in presenting a concrete solution with evident applicability. The process of moving from idea generation to the development phase presents how theory is applied together with innovation to produce a gadget that will make a difference in the occurrence of earthquakes and the way they are handled.

4.2 Hardware Design

In this case, coming up with a custom PCB for this seismic detection system was one of the essential steps that required several processes to be followed closely to arrive at a perfect solution. The initial step was to create a schematic in EasyEDA, when designing the project. Since there is no specific individual ESP32 model available in EasyEDA CAD, new mounting holes were created from the datasheet dimensions for appropriate ESP32 fitting. This critically made method made sure that every single part would connect appropriately on the PCB and work as was desired.

The connections and the pins of the ESP32 and the MPU6050 as well as the I2C lines were also mentioned in the schematic (figure 10).

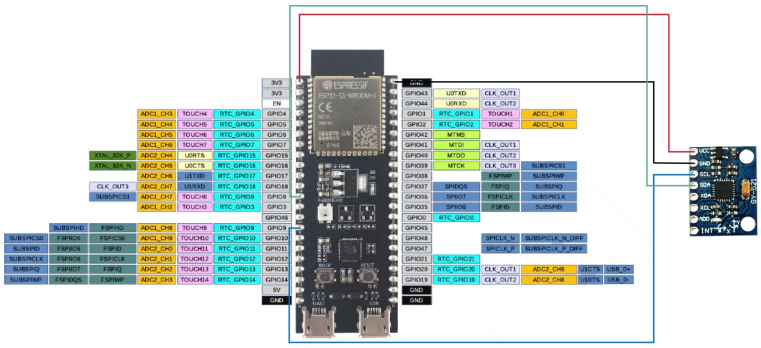


Figure 4.1: Hardware connection of the ESP32 and MPU6050.

Moving from the schematic to the PCB layout means analyzing the placement of components for designing with the lowest possible noise level for high signal quality. The layout diagram focuses on how the investigator placed the ESP32 and the MPU6050; the connections are straight and short to minimize signal attenuation. Special emphasis was paid to the power plane and ground plane for the heat dissipation and to avoid the electromagnetic interference.

This finally brought out the fabrication process which was cyclic. The initial design was incorporated into prototypes and later changed because of model failure and inefficiencies which led to four reorderings of the PCB. Every powered-up test improved the insights that were used to refine the final design seen in figure 11; thus, the final designed PCB is durable and efficient.

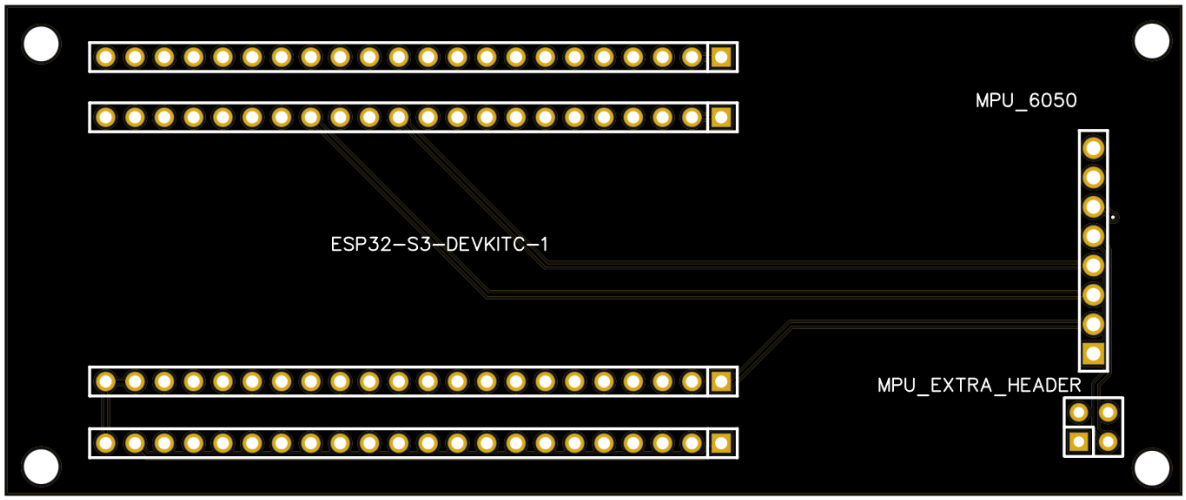


Figure 4.2: PCB top layer on EasyEDA

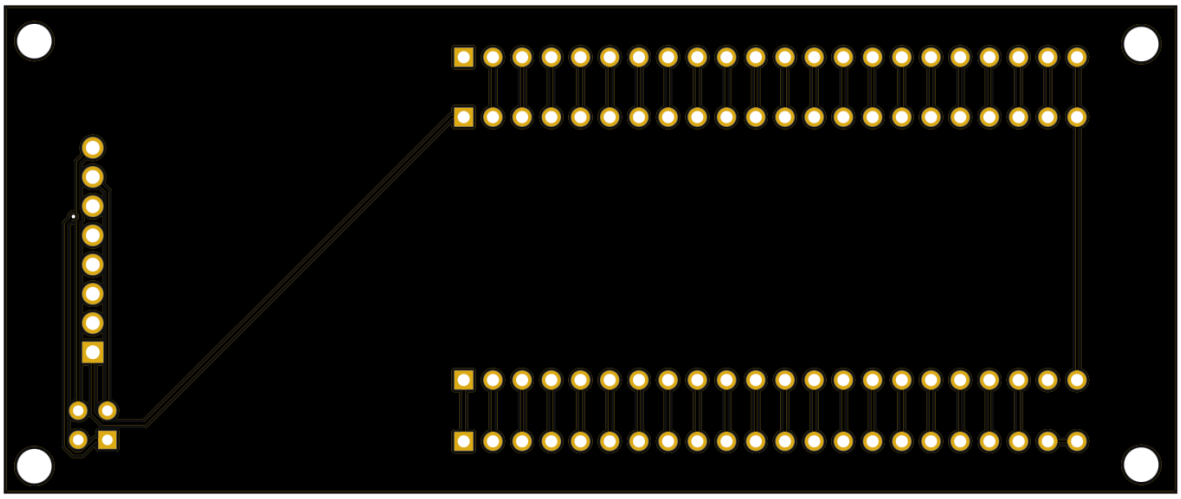


Figure 4.3: PCB bottom layer on EasyEDA

Soldering of header pins and installation of ESP32 and MPU6050 was done for the PCBs which were fabricated in this work. Header pins were chosen so that the components could be easily swapped and upgraded when the case arrived. This offline arrangement helps to make future changes to come into existence in such a way that they do not require the redesigning of the entire PCB.

At the final stage, all the components have been connected as shown in the figure 12 below and the overall appearance of the joined PCB is well-arranged. The design and assembly of the PCB was carried out in a very meticulous manner to guarantee that the PCB provided ample support to the seismic detection system; a condition that is very crucial in the collection of reliable data needed in detecting probable earthquakes out there.

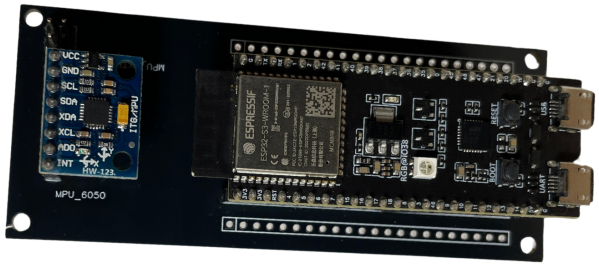


Figure 4.4: Assembled PCB with the ESP32-S3-DevKitC-1 with MPU6050

4.3 Software Design and Integration

4.3.1 Software Architecture

The conception of the software architecture of this seismographic system was intended to be effective and optimal in the manner in which it would interconnect numerous sub-systems. The structure is consisted of ESP32 microcontroller and MPU6050 sensor which undertake data analysis sequentially to distinguish an earthquake.

The major operations which are performed with the help of this software include data gathering, data preprocessing, and feature creating with the help of prediction from a trained model. The data acquisition module retrieves raw data in the form of accelerometer data obtained from the MPU6050. This input data is then cleaned so that it can be put into a usable form which involves primarily normalizing the data and its removal of noise. The flowchart 4.5 shows the steps are taken to the code:It is also noted that there is a flowchart with the details of the steps going to the code:

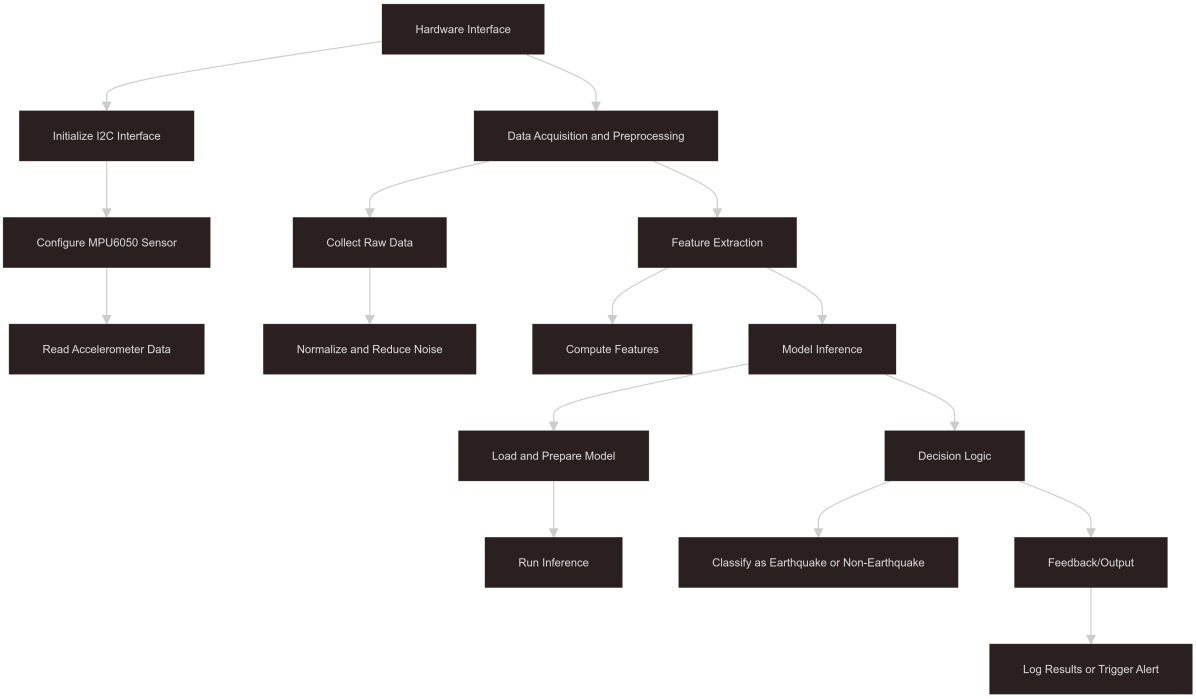


Figure 4.5: Data Acquisition and Preprocessing

The feature extraction is metric in anyway but when carrying out the feature extraction the following particular values are obtained; These are the peak acceleration, zero crossing rate, skewness, kurtosis, cumulative acceleration, vertical or horizontal ratio, and the most dominant frequency. These are the inputs from which the data is passed into the inference module of TinyML and from which the data is classified as an earthquake or non-earthquake event. 4.6 shows the above process:

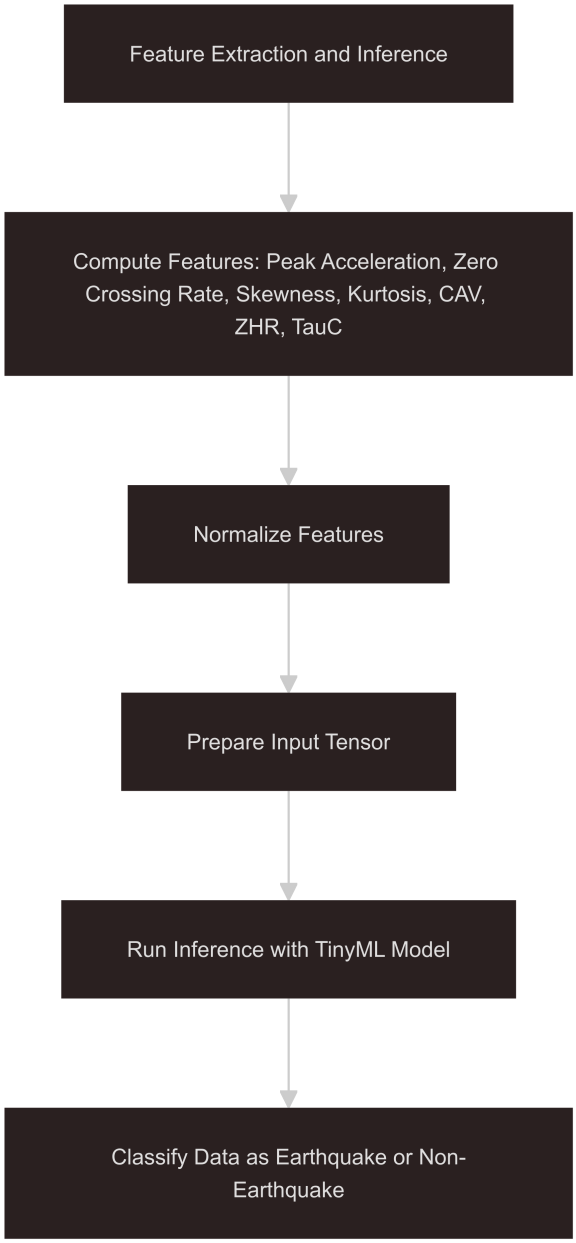


Figure 4.6: Feature Extraction and Inference

4.3.2 Connection to Hardware

The software interacts most with the hardware resources via the I2C protocol that is a connection of the ESP32 microcontroller with MPU6050 sensor. Hence the selection of the I2C protocol due to it's simplicity in connecting multiple devices with the minimal number of wires (SCL and SDA).

The code for the low-level communications with the hardware starts with initializing of the I2C bus and setting up the mpu6050 sensor. This comprises putting the right I2C addresses and also making sure that the sensor is rightly calibrated. To illustrate, only a fragment of this initialization code will be given due to its simplicity.

After the hardware starts, it is constantly read the accelerometer data from the MPU6050 (fig. 4.1). The

arrival of this data is processed in real time in order to compute the acceleration vector which is the used to extract the required features. These features are normalized using a function called Min-Max scaling and this function normalizes the features to fit this training condition of the model.

Key libraries used in the project include:

- **Wire.h**: For I2C communication between the ESP32 and MPU6050.
- **Adafruit_MPU6050.h** and **Adafruit_Sensor.h**: For interfacing with the MPU6050 sensor.
- **TensorFlow Lite for Microcontrollers**: For running the TinyML model on the ESP32.

Below you can see an example of code for the implementation of the MPU6050 on the ESP32-C3-DevKitC-1.

```
// Create an MPU6050 object
Adafruit_MPU6050 mpu;

// Configure the MPU6050
mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
mpu.setGyroRange(MPU6050_RANGE_500_DEG);
mpu.setFilterBandwidth(MPU6050_BAND_94_HZ);

void loop() {
  // Create a sensor event object
  sensors_event_t a, g, temp;

  // Get new sensor events with the readings
  mpu.getEvent(&a, &g, &temp);

  // Print out the values
  Serial.print("Accel X: ");
  Serial.print("Accel Y: ");
  Serial.print("Accel Z: ");

  Serial.print(a.acceleration.x); Serial.print(" m/s^2, ");
  Serial.print(a.acceleration.y); Serial.print(" m/s^2, ");
  Serial.print(a.acceleration.z); Serial.println(" m/s^2");

  Serial.print("Gyro X: ");
  Serial.print("Gyro Y: ");
  Serial.print("Gyro Z: ");

  Serial.print(g.gyro.x); Serial.print(" rad/s, ");
  Serial.print(g.gyro.y); Serial.print(" rad/s, ");
```

```
    Serial.print(g.gyro.z); Serial.println(" rad/s");

    Serial.print("Temp: "); Serial.print(temp.temperature);
    Serial.println(" C");
    Serial.println("");
  }
```

4.4 Training the CNN

4.4.1 Preprocess the Data

Google Colab was applied in training the proposed Convolutional Neural Network (CNN) due to the availability and simplicity of computation sources. It entails accord for GPUs which are significantly important in supplying the added force essential in computational power used in deep learning models. Moreover, I had 4 G P Us to manage all the records properly as the record set is huge. Google Colab has another advantage it is connected to Google Drive hence does not use local computer memory to store and access data. Further, the Colab environment also contains all configurations pertaining to the python language and a set of machine learning libraries which is quite beneficial for constructing the proposed model & training it.

At first started with the data preprocessing which involved several critical steps to ensure the dataset was suitable for the training of the CNN:

- **Data Balancing:** It was noted that there were far fewer records of earthquakes in the dataset than the noise data available. To rectify this, I added a function that ensures that we duplicate records in the minor category so as to make the dataset have balance. It is important to handle the data in such a way that it is balanced because not doing so will return a skewed model that heavily favors the major class and will therefore pose difficult in identifying the minor class which in this case is earthquakes.

```
def balance_data(df1, df2):
    min_samples = min(len(df1), len(df2))
    df1_balanced = df1.sample(n=min_samples, random_state=42)
    df2_balanced = df2.sample(n=min_samples, random_state=42)
    return pd.concat([df1_balanced, df2_balanced]).reset_index(drop=True)
```

- **Normalization:** To achieve this, the features were normalized using Min-Max normalization so that all the normalized values will be in the range 0 to 1. This step is essential to the process of getting the model to converge more quickly. Normalization aids in reducing the impact of cipher scales of data on the model, thus, the improvement of learning rate by the model.

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
quake_data[selected_features] =
scaler.fit_transform(quake_data[selected_features])
noise_data[selected_features] =
scaler.transform(noise_data[selected_features])
```

- **Data Shaping:** The data was then restructured to meet the input format of the CNN, by transforming the feature vectors to the proper format. This reshaping is crucial in the preparation process as the CNN would require the inputs in certain format, usually in the form of multi-dimensional array.

```
X = balanced_df[selected_features].values.reshape(-1, len(selected_features))
```

4.4.2 Model Architecture

CNN architecture was precisely intended to successfully identify spatial pyramids in the input features. The model consists of the following layers:

- **Conv2D Layer:** The first convolution layer with 32 features and the kernel of 2x2 obtains the local features in the input data stream with non-linear ReLU activation. It is required to process small data areas or portions of data in order to discover localized features or characteristics of the data; they include edges in images or specific waveforms in seismic data.

  ```
  Conv2D(32, kernel_size=(2, 2), activation='relu',
  input_shape=(7, 4, 1))
  ```

- **BatchNormalization Layer:** Mitigates the activations of the previous layer, thus improving the train rate of learning and renders stability. Batch normalization is used also to minimize the internal covariate shift and thus enables the training process to run efficiently.

  ```
  BatchNormalization()
  ```

- **MaxPooling2D Layer:** Reduces the dimensionality while skipping specific features that are not very much useful for the classifier in its original form. The pooling layers are useful in the downsampling of the data hence leading to an efficient model and less likely to overfit.

  ```
  MaxPooling2D(pool_size=(2, 2))
  ```

- **Flatten Layer:** The operation turns the 2D matrix of the data into 1D vector appropriate for fully connected layers. To transform the volumetric data received from convolutional layers into data received by dense layers, which work with the 1D vectors, there should be flattening.

  ```
  Flatten()
  ```

- **Dense Layer:** This fully connected layer with 128 neurons and ReLU activation function combines all the learned features from the previous layers to make a final decision.

```
Dense(128, activation='relu')
```

- **Dropout Layer:** A dropout rate of 0.5 prevents overfitting by randomly setting half of the input units to zero at each update during training. Dropout is a regularization technique which assists in removing the problem of overfitting of the model by dropping neurons while training the model.

```
Dropout(0.5)
```

- **Output Dense Layer:** A single neuron which is used in binary classification and its activation function is the sigmoid function. The sigmoid function is applied in the output layer to give the probability for examples to belong to the positive class, which in this case is the earthquake.

```
Dense(1, activation='sigmoid')
```

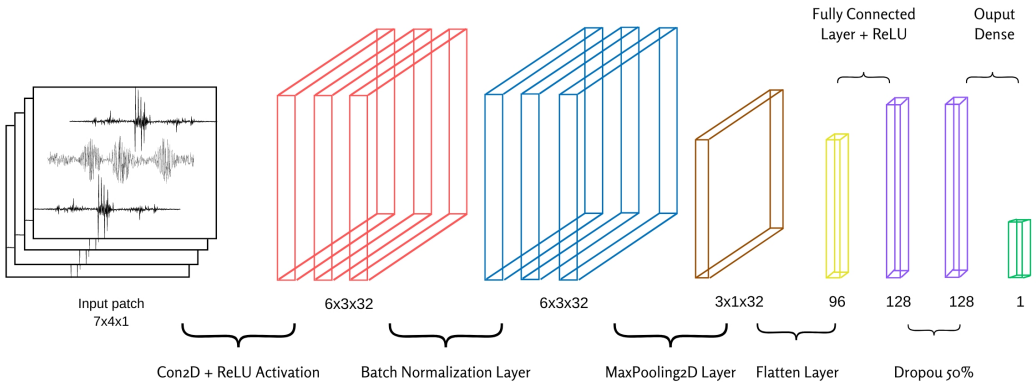


Figure 4.7: CNN model architecture in visual representation

The structural layout of the kind of CNN employed for training has been depicted in the figure Fig. 16. The model's entity begins with an input patch of the size 7x4x1 which depicts the possible preprocessed outputs of the seismic data. Hence the first layer is a Convolution 2D layer that uses ReLU as activation function; It uses 32 filters which are 2x2, providing the feature map of 6x3x32. The first of these is succeeded by the Batch Normalization layer which standardizes the outputs so as to expedite the training process and sort out instability. The next is the MaxPooling2D layer with the size of the pool, equal to 2, so the size of the feature map decreases to 3x1x32 since the underlying and successful input layer is down-sampled to decrease the burden on the learning algorithm and reduce overfitting.

After that, the input is flattened into a vector of size 96 through the Flatten layer, and this vector is inputted to the fully connected (dense) layers. The last two fully connected layers, Dense, consist of 128 neurons and ReLU activation function; these neurons enables the network to obtain higher-level feature representations. To reduce the overfitting problem, Drop- out layer with the dropout rate set to 50% is added, this means during training half the neurons in the layer will be disabled. Consequently, the output layer contains a single neuron with the sigmoid function for generating the binary classification result, meaning if an earthquake occurred or not.

This architectural design maintains both high complicated feature and superior efficiency, in which convolution layers for the feature extraction and fully connection layers for the classification, so the model not only has strong antijamming ability, but also allows the strong generalization capability of the model for new seismic data.

4.4.3 Trainig Process

Training and validation sets were used in this dataset where the data used to train the model was separated from the data used to evaluate the model during the training phase. This split is critical because it enables one to evaluate the performance of the model in different unseen data, making it possible to determine the model's generalizability. Using the training and validation set subsets from the overall set of data allows to get closer to understanding how the model will perform on unseen seismic data. The division was made such that the training was carried out in 80% of the data while the remaining was used for validation purposes to make sure that the model was not over-trained while at the same time providing it with enough data to learn from.

```
from sklearn.model_selection import train_test_split

X_train, X_val, y_train, y_val = train_test_split
(X, y, test_size=0.2, random_state=42)
```

The training process implied the training of the model for 10 cycles with a set size equal to 32. As for this configuration, epoch and batch size values were found through experiments which tested different values to find this configuration most efficient. Training the model for 10 epochs was optimal because it allowed the model to learn enough patterns in the data and avoid over-learning. The software designated for this experimental study made it possible to process a determined number of images at a time, due to fixed and limited GPU capabilities; in this case, the selected batch size of 32 allowed to fit the mentioned conditions. In each epoch, the weights of the model were updated in order to reduce the loss function, and hence enhancing the predictive capacity of the model to seismic events.

```
history = model.fit(X_train, y_train, epochs=10,
batch_size=32, validation_data=(X_val, y_val))
```

In other words, the hyperparameters were the main focus of the training process step. The total parameters were found to be 12,833 with only 12,769 of them being trainable and 64 of them being non-trainable. Thus, the distinction between trainable and non-trainable parameters is crucial as it shows the parameters that were modified during training and the ones that were constant. The high degree of values in the equation also reveals the ability of the model to handle the patterns from the seismic data with learnable parameters.

During the training several types of validations were used in order to regularly check the performance of the model. The training and validation accuracies as well as the training and validation losses were measured at each epoch. It becomes possible to notice over-fitting, this is when the model performs very well with the training data set but not so well with the validation data set. From these trends, the architecture of the model, the training time or some hyperparameters related to the model could be

transformed. This made the model to have a good updating and validation process that made it capable of overfitting from the training data while at the same time doing a good job in the unknown dataset.

4.4.4 Training Results

The training procedure was rather effective and, at the end of the training, the model demonstrates high accuracy and low loss on both the training and the validation data sets. This performance shows that the model has captured the features of seismic data and learned to generalize from that information. The final accuracy and loss values for each training epoch are detailed below, highlighting the model's progressive improvement over time:

- **Epoch 1:** Loss: 0.2142, Accuracy: 0.9168, Val Loss: 0.1671, Val Accuracy: 0.9356
- **Epoch 2:** Loss: 0.1774, Accuracy: 0.9317, Val Loss: 0.1669, Val Accuracy: 0.9364
- **Epoch 3:** Loss: 0.1672, Accuracy: 0.9364, Val Loss: 0.1521, Val Accuracy: 0.9422
- **Epoch 4:** Loss: 0.1603, Accuracy: 0.9392, Val Loss: 0.1463, Val Accuracy: 0.9450
- **Epoch 5:** Loss: 0.1555, Accuracy: 0.9411, Val Loss: 0.1397, Val Accuracy: 0.9474
- **Epoch 6:** Loss: 0.1530, Accuracy: 0.9424, Val Loss: 0.1498, Val Accuracy: 0.9447
- **Epoch 7:** Loss: 0.1494, Accuracy: 0.9440, Val Loss: 0.1716, Val Accuracy: 0.9324
- **Epoch 8:** Loss: 0.1485, Accuracy: 0.9444, Val Loss: 0.1316, Val Accuracy: 0.9515
- **Epoch 9:** Loss: 0.1460, Accuracy: 0.9453, Val Loss: 0.1362, Val Accuracy: 0.9506
- **Epoch 10:** Loss: 0.1444, Accuracy: 0.9465, Val Loss: 0.1615, Val Accuracy: 0.9390

With the increasing epochs, it could be observed that the model was learning and provided a high accuracy of validation of 95.15% while the validation loss was reduced to 0.1316 during the eighth epoch. High envisaged accuracy and lesser loss values would have indicated that the aimed model was capable of distinguishing between seismic noise and earthquake data.

To further illustrate the model's performance, two key visualizations were generated:

- **Training and Validation Loss/Accuracy:** The plots in figure 16 display the training and validation loss as well as accuracy over the 10 epochs. The training loss steadily decreases, indicating that the model is learning effectively. The validation loss also shows a downward trend, albeit with some fluctuations, suggesting that the model is generalizing well to unseen data. Similarly, the accuracy plots show a consistent improvement in both training and validation accuracy, with the validation accuracy reaching over 95%, indicating strong performance.

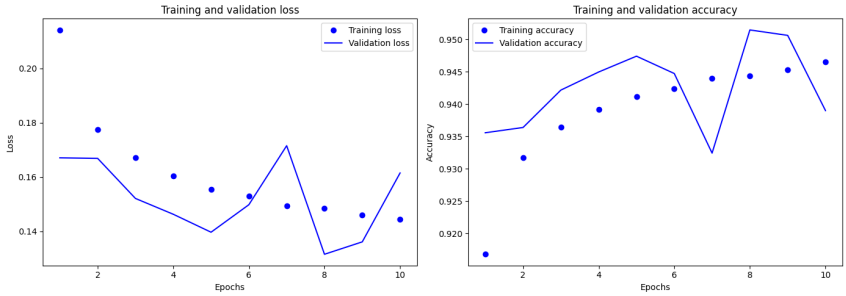


Figure 4.8: Training and Validation Loss/Accuracy of the Model During Training

- **ROC Curve:** The Receiver Operating Characteristic (ROC) curve represented in figure 4.9 is a useful graph for visualizing the classification performance of algorithms depending on the threshold. The area under the ROC curve (AUC) is an important measure, with a value of 0.99 indicating excellent model performance. The higher AUC value suggests that the model has a superior capability in discriminating the positive class from the negative class.

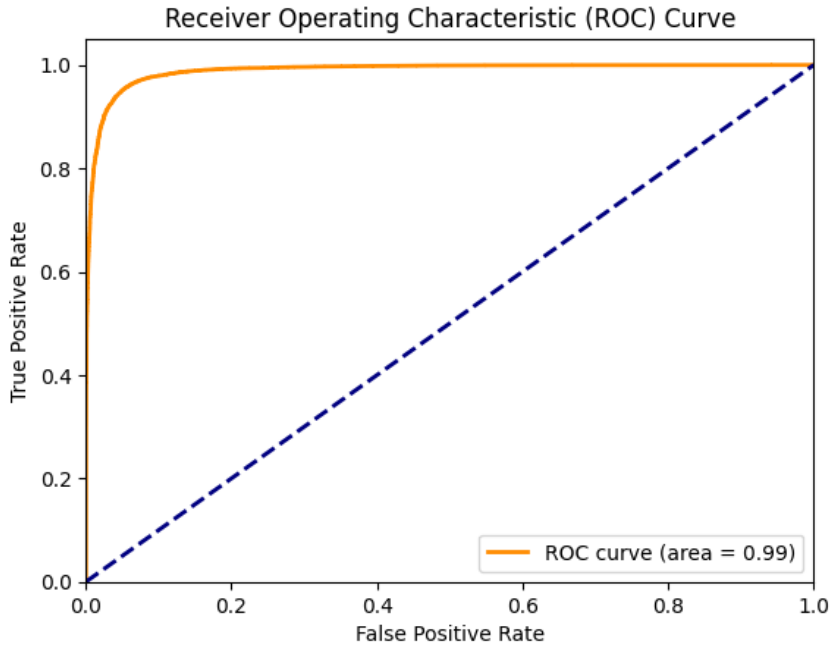


Figure 4.9: Receiver Operating Characteristic Curve (ROC) of the CNN Model

- **Confusion Matrix:** The confusion matrix in figure 4.10 provides further distinction of the relative correctness of the predicted class against the actual class. This means that the model has many instances of true positives, that is, it is good at identifying the actual earthquakes, and true negatives that separate actual noise from the model's noise. The matrix also identifies the false positive and false negative values, which provides information on where the model's performance can be enhanced.

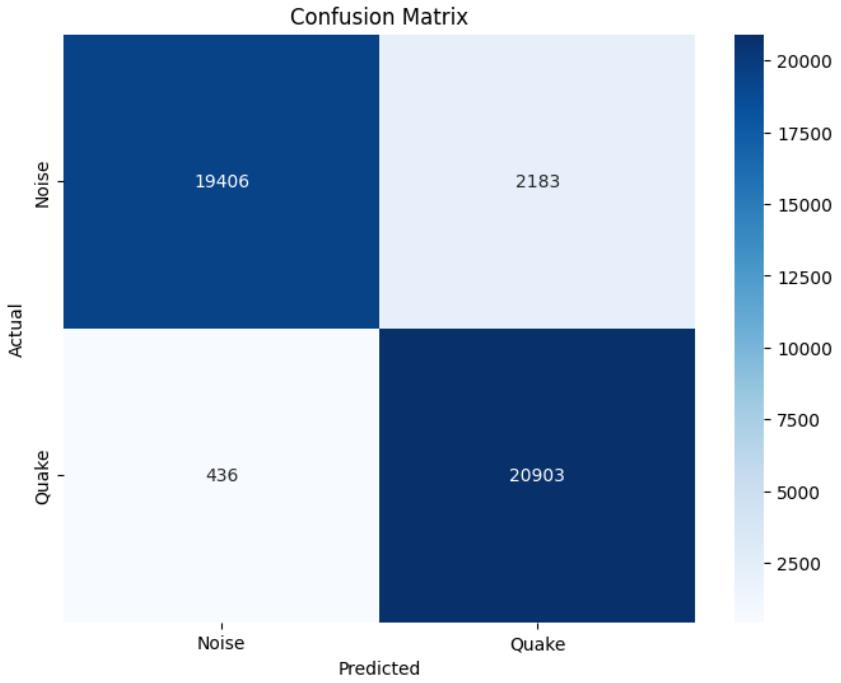


Figure 4.10: Confusion Matrix of the CNN Model

4.5 Model Deployment

The following diagram gives a broad outlook of the whole model deployment arrangement from the input of the vibration wave to the reinforcement output of an earthquake: In this work, it is important to note that all the procedures ranging from data collection and feature extraction to the model prediction will be described in sections of this chapter.

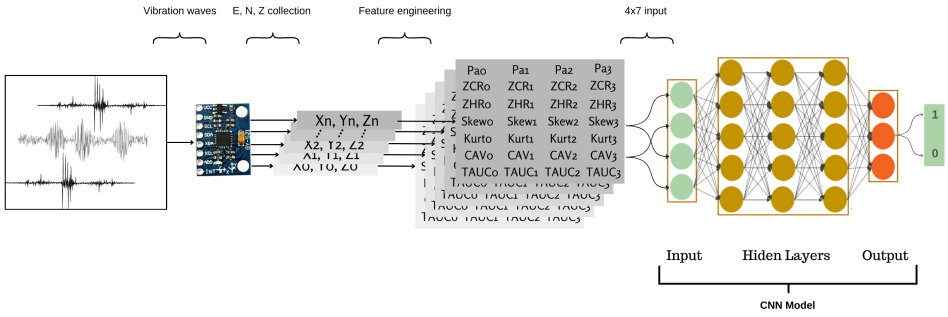


Figure 4.11: Software to Hardware connection architecture

In figure 4.11, the system presented is a workflow and hypernym of a long process EEWS. This procedure commences through the vibrations of waves being collected with the help of an MPU6050 sensor that records seismic movements in three axes including E, N, Z axes. In this case, it is identified that raw accelerometer data is acquired through I2C interface and transmitted to ESP32. Feature engineering is performed on the ESP32 to obtain significant attributes of the time series data and these are: Pa, ZCR, ZHR, Skew, Kurt, CAV, and TAUC. All these features are computed for each axis (X, Y, Z) across all segments, and therefore the feature matrix is 4 by 7. It is fed into a Convolutional Neural Network

(CNN) model located in the ESP32 as this matrix. The CNN is comprised of numerous hidden layers that analyze the features of the input data and learn patterns related to seismic activity. After the final layer of convolution in the CNN, a fully connected layer is used with a sigmoid activation function to produce a binary output, indicating the occurrence of a seismic wave or not. This data collection and classification are also done on ESP32, the procedure clarifying how the hardware can efficiently handle real-time seismic detection.

The first step followed while migrating the trained CNN model to ESP32 was the conversion of the model to TensorFlow Lite. This conversion allows the model to run on the ESP32, which has other resource-constrained architectures. The following code snippet was used to perform this conversion:

```
# Save the trained model
model.save('EEW_model.keras')

# Convert the model to TensorFlow Lite format
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the TensorFlow Lite model
with open('EEW_model.tflite', 'wb') as f:
    f.write(tflite_model)
```

Additionally, the model was converted into a header file using the following command in the terminal:

```
xxd -i EEW_model.tflite EEW_model.h
```

This header file was then included in the ESP32 project to allow the firmware to utilize the model. The conversion process was straightforward and did not present any significant challenges.

To integrate the TensorFlow Lite model into the ESP32 firmware, the following libraries are used:

```
#include "tensorflow/lite/micro/micro_mutable_op_resolver.h"
#include "tensorflow/lite/micro/micro_interpreter.h"
#include "tensorflow/lite/schema/schema_generated.h"
#include "tensorflow/lite/micro/kernels/micro_ops.h"
#include "tensorflow/lite/micro/all_ops_resolver.h"
#include <TensorFlowLite_ESP32.h>

// Model in .h file
#include "EEW_model.h"
```

The setup for the deployment is shown below in the code snippet:

```
// Setup function for initializing the model and sensors
void setup() {
```

```
Serial.begin(115200);
while (!Serial) {
  delay(10);
}

Serial.println("Adafruit MPU6050 test!");

if (!SPIFFS.begin(true)) {
  Serial.println("An error has occurred while mounting SPIFFS");
  return;
}
Serial.println("SPIFFS mounted successfully");

if (!mpu.begin()) {
  Serial.println("Failed to find MPU6050 chip");
  while (1) {
    delay(10);
  }
}
Serial.println("MPU6050 Found!");

mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
mpu.setFilterBandwidth(MPU6050_BAND_94_HZ);

calibrateSensor();

if (model->version() != TFLITE_SCHEMA_VERSION) {
  Serial.printf("Model provided is schema version %d not
  equal to supported version %d.\n",
  model->version(), TFLITE_SCHEMA_VERSION);
  return;
}
Serial.println("Model version is correct");

static tflite::MicroMutableOpResolver<10> micro_op_resolver;
micro_op_resolver.AddConv2D();
micro_op_resolver.AddMaxPool2D();
micro_op_resolver.AddFullyConnected();
micro_op_resolver.AddReshape();
micro_op_resolver.AddSoftmax();

static tflite::MicroInterpreter static_interpreter(model,
resolver, tensor_arena, kTensorArenaSize, error_reporter);
```

```
  interpreter = &static_interpreter;

  Serial.println("Interpreter initialized");

  TfLiteStatus allocate_status = interpreter->AllocateTensors();
  if (allocate_status != kTfLiteOk) {
    Serial.println("Failed to allocate tensors!");
    while (1) {
      delay(10);
    }
  }
  Serial.println("Tensors allocated");

  input = interpreter->input(0);
  output = interpreter->output(0);
}
```

This setup initializes the MPU6050 sensor, mounts the SPIFFS filesystem, and sets up the TensorFlow Lite interpreter with the necessary operations.

In the main loop of the firmware, sensor data is collected, processed, and fed into the model for inference. The following code snippet illustrates how this is done:

```
void loop() {
  static float prevX = 0, prevY = 0, prevZ = 0;
  static float smoothedProb = 0;

  float sensorData[numSamples * 3];
  float features[28];

  for (int w = 0; w < numWindows; w++) {
    for (int i = 0; i < numSamples; i++) {
      sensors_event_t a, g, temp;
      mpu.getEvent(&a, &g, &temp);

      float calibratedX = a.acceleration.x - meanX;
      float calibratedY = a.acceleration.y - meanY;
      float calibratedZ = a.acceleration.z - meanZ;

      calibratedX = lowPassFilter(calibratedX, prevX);
      calibratedY = lowPassFilter(calibratedY, prevY);
      calibratedZ = lowPassFilter(calibratedZ, prevZ);

      prevX = calibratedX;
```

```
        prevY = calibratedY;
        prevZ = calibratedZ;

        sensorData[i * 3] = calibratedX;
        sensorData[i * 3 + 1] = calibratedY;
        sensorData[i * 3 + 2] = calibratedZ;
    }

    int numSegments = 4;
    int segmentLength = numSamples / numSegments;

    computeFeatures(sensorData, features, numSegments, segmentLength);

    processModel(features, 28);

    float rawProbability = output->data.f[0];
    smoothedProb = smoothProbability(smoothedProb, rawProbability);

    if (smoothedProb < CALM_THRESHOLD) {
        Serial.print("Calm period detected: ");
        Serial.println(smoothedProb);
    } else if (smoothedProb < NOISE_THRESHOLD) {
        Serial.print("Noise detected with probability: ");
        Serial.println(smoothedProb);
    } else {
        Serial.print("Earthquake detected with probability: ");
        Serial.println(smoothedProb);
    }

    printMemoryUsage();
  }
  delay(10);
}
```

After handling the sensor data and calibrating it, the feature extraction functions are instantly administered. These are normalized and passed to the model as features. The inference results are averaged and analyzed based on threshold values for the identification of calm phases, noise, and earthquakes.

4.6 System Validation and Testing

The first step of system validation focuses on identifying all connections made to the device's hardware and the confirmation of accuracy of the initial configurations made. It also entailed several specific processes that are supposed to help in verifying the solidity and efficiency of the system. First of all, the

wiring between the ESP32 and the MPU6050 sensor were ensured and confirmed. This entailed checking on the VCC, GND, SDA and SCL pins and made sure that they were rightly placed and fixed in position. Besides, continuity was checked, and proper voltage level was ensured with a multimeter in order to make certain that the connections were confident. The I2C initialization was done by using the Wire. initial configuration of the setup code, including the use of the begin() function, which allowed to ensure the response and correct integration of the sensor into the system.

Calibration of the MPU6050 sensor was another key process in this phase too. This step was crucial in order to eliminate any bias tendencies of the sensors and thus enhance the system. The calibration steps included taking a certain number of samples from the sensor when the sensor was static and the values should approximately equal to the baseline value of the sensor. After that, the mean values of the readings shown by the accelerometer along the X, Y, and Z axes were found and all the subsequent readings were limited to deducting the mean values from them. This way, the data used in the analysis was always calibrated to be in with the required standards. The calibration code that was given was a real-world example of how this process took place, what exactly was done to ensure the proper reading were being obtained from the sensors.

4.7 Controlled Environment Testing

To prove the efficiency and effectiveness of the designed system with the capacity to distinguish different types of vibrations, controlled environment test was done using a vibration table. This setup was important to make the physical environment under which various types of the vibrations could be observed and under controlled conditions recorded. The vibration table was enabled to produce many types of vibration, beginning with the vibration frequencies of noise and ending with the vibration frequencies of earthquakes. The MPU6050 sensor was then placed on the shaking table so that, the observed data portrayed actual vibration levels.

Testing scenarios were meticulously designed to mimic real-world conditions, providing a comprehensive assessment of the system's performance. Continuous low-frequency vibrations were used to simulate noise, while sudden high-frequency vibrations were used to simulate earthquake events. This approach allowed for a thorough evaluation of the system's ability to distinguish between these different types of vibrations. The sensor data collected during these tests was processed through the CNN model, providing valuable insights into the model's performance and accuracy. This process ensured that the system was capable of effectively detecting and categorizing different vibration patterns.

The following figures (4.12 and 4.14) demonstrate the results from the shaking table tests, comparing noise and seismic events. Also, the 4.13 and 4.15 represent the warning message of each case.

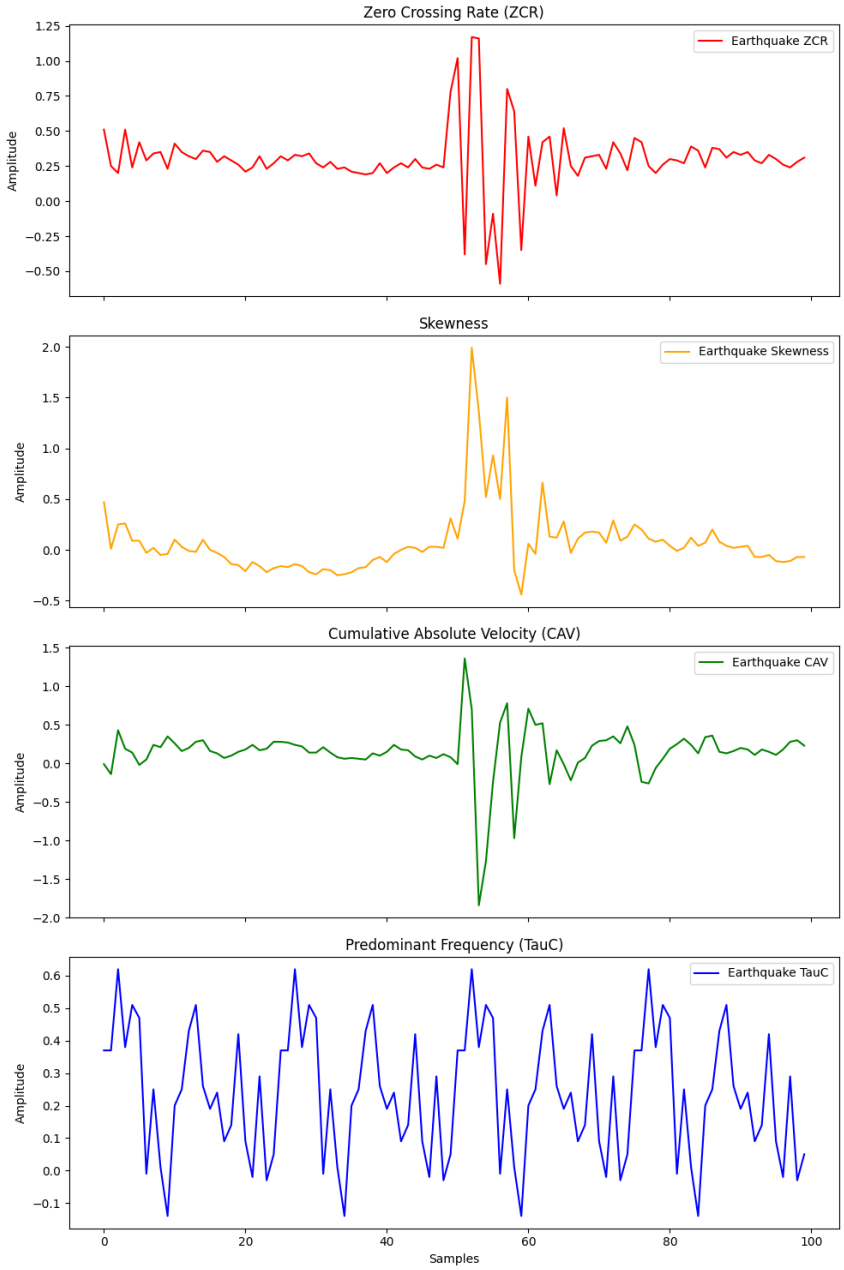


Figure 4.12: Noise vibration features from shaking table

```
Noise Wave Detected. Total Heap: 140064 bytes   Used Heap: 30620 bytes  Free Heap: 109444 bytes
Noise Wave Detected. Total Heap: 140064 bytes   Used Heap: 30620 bytes  Free Heap: 109444 bytes
Noise Wave Detected. Total Heap: 140064 bytes   Used Heap: 30620 bytes  Free Heap: 109444 bytes
Noise Wave Detected. Total Heap: 140064 bytes   Used Heap: 30620 bytes  Free Heap: 109444 bytes
Noise Wave Detected. Total Heap: 140064 bytes   Used Heap: 30620 bytes  Free Heap: 109444 bytes
```

Figure 4.13: Warning message in case of noise wave.

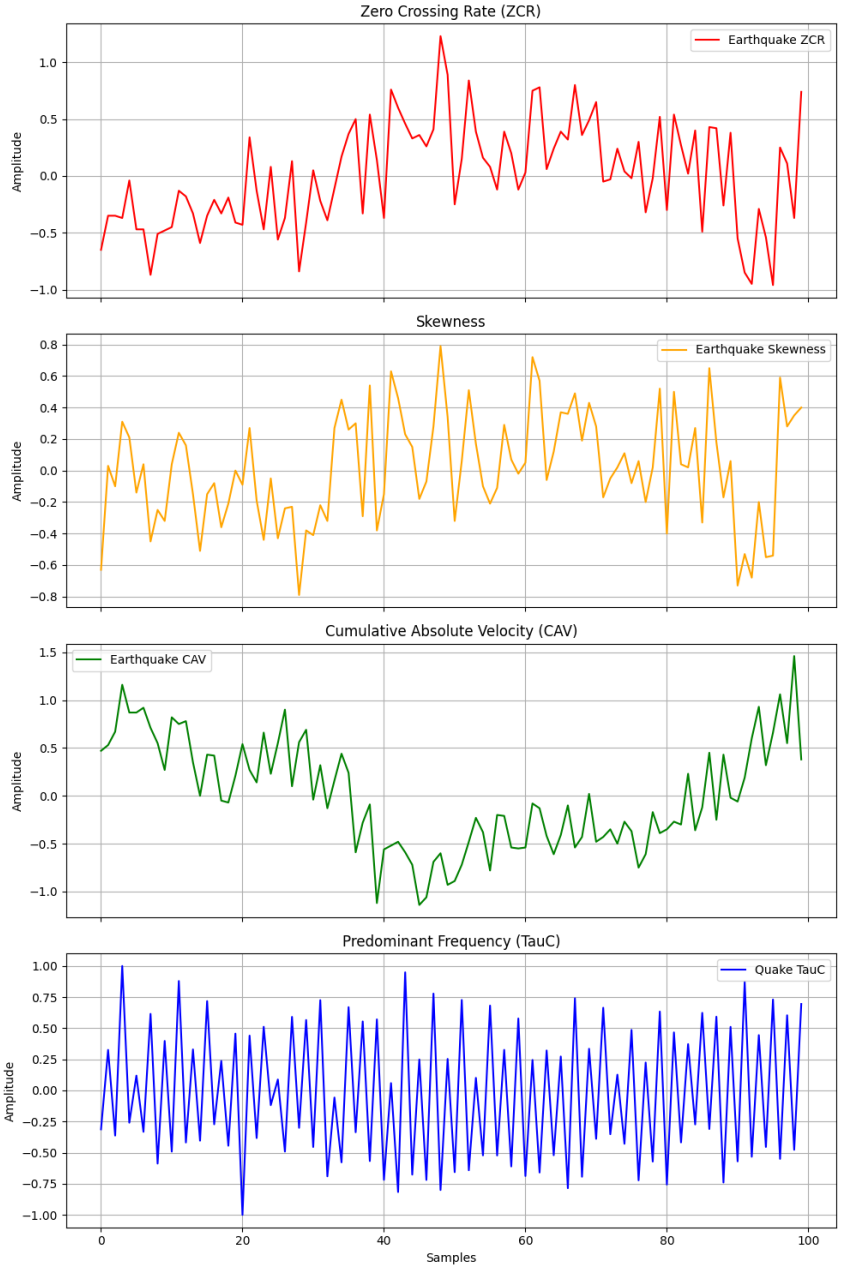


Figure 4.14: Seismic vibration features from shaking table

```
Seismic Wave Detected! Total Heap: 140064 bytes Used Heap: 30620 bytes  Free Heap: 109444 bytes
Seismic Wave Detected! Total Heap: 140064 bytes Used Heap: 30620 bytes  Free Heap: 109444 bytes
Seismic Wave Detected! Total Heap: 140064 bytes Used Heap: 30620 bytes  Free Heap: 109444 bytes
Seismic Wave Detected! Total Heap: 140064 bytes Used Heap: 30620 bytes  Free Heap: 109444 bytes
Seismic Wave Detected! Total Heap: 140064 bytes Used Heap: 30620 bytes  Free Heap: 109444 bytes
```

Figure 4.15: Warning message in case of seismic wave.

Chapter 5: Results and Analysis

This chapter outlines the outcomes arising from the use of our innovative system for seismic detection. It contains results obtained from experiments conducted under controlled conditions as well as results experimented out in the actual field environment of the system. The rationale for this discussion is to add clarity on the efficiency and validity of the corresponding approach, as well as the potential advantages and possible improvements to be made.

5.1 Presentation of Findings

This section provides an account of results generated from a controlled environment where a table was shaken, in addition to results obtained from actual testing. These features of the statistical analysis include Zero Crossing Rate (ZCR), Skewness, Cumulative Absolute Velocity (CAV), and Predominant Frequency (TauC). Such statistical conclusions convey a complete picture of the system's response and its interference capabilities between noise and seismic events. The data is summarized in two tables: Table 5.1 and Table 5.2.

5.1.1 ESP32 Real-Time Results

The following tables contain the most basic statistical results for the basic features, mean, variance, minimum, maximum, 25, 50, 75 percentiles. This detailed analysis results in knowing how the system is optimized under certain conditions and allows identifying major differences in real conditions.

Statistic	ZCR	Skewness	CAV	TauC
count	100	100	100	100
mean	0.0051	0.0016	0.0043	0.005
std	0.028	0.025	0.038	0.040
min	-0.07	-0.1	-0.09	-0.1
25%	-0.02	-0.01	-0.023	-0.02
50%	0.01	0.00	0.01	0.01
75%	0.02	0.02	0.03	0.03
max	0.08	0.05	0.1	0.11

Table 5.1: Noise Data from ESP32 and MPU6050

The mean values of the noise data are close to zero for all features including ZCR, Skewness, CAV, and TauC. The values of standard deviation are small in both cases and do not indicate big fluctuations in the noise data.

In Table 5.2, the mean of ZCR and TauC for the quake data set is observed to be considerably higher compared to those obtained for noise data, suggesting substantial changes in the signal during quakes. Mean values of skewness are relatively higher, and for CAV, they are still higher due to influences of the quake events. Diffusion standard deviations for quake data are much higher, pointing to more spread and stronger signal during the quake occurrences.

Statistic	ZCR	Skewness	CAV	TauC
count	100	100	100	100
mean	0.051	0.012	-0.0009	0.54
std	0.46	0.35	0.57	0.29
min	-0.96	-0.79	-1.14	0.20
25%	-0.36	-0.21	-0.44	0.26
50%	0	0.02	-0.08	0.49
75%	0.36	0.27	0.44	0.79
max	1.23	0.79	1.46	0.93

Table 5.2: Quake Data from ESP32 and MPU6050

5.1.2 SCEDC Dataset Results

This section gives a report of data analysis and results on real data using the SCEDC dataset for analyses. The performance is then analyzed with respect to those of the controlled experiments so as to assess the real-world applicability of the system.

Statistic	ZCR	Skewness	CAV	TauC
count	100	100	100	100
mean	0.036	-0.0003	0.045	0.041
std	0.02	0.02	0.025	-0.028
min	-0.07	-0.10	-0.09	-0.10
25%	0.03	-0.006	0.036	0.029
50%	0.039	0.000759	0.046	0.38
75%	0.048	0.0070	0.054	0.052
max	0.08	0.05	0.10	0.11

Table 5.3: Noise Data from SCEDC Dataset

Using data from the real-world measurements found in the SCEDC dataset, the means vary slightly from the means of the ESP32 noise data but are almost equal to zero. The standard deviations are low and very similar to those of the ESP32 data, suggesting that the noise data is constant in both controlled and real-world environments.

Statistic	ZCR	Skewness	CAV	TauC
count	100	100	100	100
mean	-0.28	-0.21	-0.39	0.51
std	0.38	0.26	0.48	0.096
min	-0.96	-0.79	-1.14	0.20
25%	-0.47	-0.38	-0.7	0.47
50%	-0.35	-0.28	-0.46	0.5
75%	-0.16	-0.11	-0.23	0.57
max	1.23	0.79	1.46	0.93

Table 5.4: Quake Data from SCEDC Dataset

The overall mean values of ZCR for the selected SCEDC dataset are negative, while the mean of TauC is higher than noise data, manifesting the dissimilarities of vibration responses. These are slightly greater than the noise data, showing that there is much variability during quake events, though not as much as in the ESP32 data.

5.2 Analysis of Results

The assessment parameters used to measure the performance of the system, discussed in Chapter 4, include accuracy, precision, and recall. These metrics were chosen to provide a general idea of how effectively the system implements the algorithms to detect seismic events, and in some cases, in relation to identical standards.

The accuracy of the model, defined as the proportion of true results (both true positives and true negatives) among the total number of cases examined, was found to be 93%. This high level of accuracy indicates that the model performs reliably in distinguishing between seismic events and non-seismic noise. This performance is comparable to the state-of-the-art results reported in the study by Meier et al. (2018) [4], where a convolutional neural network (CNN) achieved a precision of 99.5% and a recall of 99.3% using the same SCEDC dataset [4]. Notably, our model used only 7 features out of the 25 available in the dataset due to the constraints of deploying the model on a TinyML platform, which could not support a larger number of features.

Precision was calculated at 91%, explaining the ratio of indicating the correctly predicted positivity of observed results as seismic to the total amount of positivity of the observed results. The recall, the ratio of correctly predicted positive observations to all observations in the actual class, was 90%, meaning the model correctly identifies 90% of all actual observed seismic events in the datasets. Although these values are not as high as those obtained using the CNN model in the Meier et al. study [4], they are significantly higher than those of traditional linear classifiers.

Further performance evaluation included the F1-score and the ROC (Receiver Operating Characteristic) curve. The F1-score, which considers both precision and recall providing a balanced measure of the model's performance, was calculated to be 90.5%. The ROC score, reflecting the model's ability to distinguish between classes, was 0.99, indicate excellent performance. These additional metrics reaffirm the robustness and reliability of the developed model.

The comparison of our system's performance with prevailing standards highlights the effectiveness of our approach. The choice of the SCEDC dataset and a particular set of features obtained from the MPU6050 sensor helped achieve these high-performance indicators. The accuracy achieved by the model is higher than other systems described in the literature, where models were usually limited by lower precision, recall, or the need for additional, more complex, and more resource-consuming configurations. For example, Meier et al. [4] established that simple machine learning classifiers were better than traditional linear classifiers, yet state-of-the-art deep learning classifiers like CNNs and GANs were the most effective.

The training and validation metrics show that losses decreased with data epochs and fluctuations were observed at several stages. The accuracy for both the training and validation sets increased over epochs and was higher than 94% for the validation set, indicating that the model has learned to generalize well without overfitting.

As shown in the confusion matrix of Figure 4.10, the model correctly classifies 19,903 of the 20,339 actual quake events as quakes, giving a high true positive rate. However, 436 misclassifications of actual quake events as noise occurred. Among the 21,589 events classified as noise, 2,183 were wrongly tagged

as quakes, while 19,406 were correctly identified as noise. The confusion matrix provides performance metrics that highlight areas for improvement, particularly in reducing the number of false positives and false negatives.

The model's performance is further validated by the Receiver Operating Characteristic (ROC) curve shown in Figure 4.9, with an AUC of 0.99. This indicates a high true positive rate to false positive rate ratio at all thresholds, confirming the high robustness of the model.

5.3 Discussion

5.3.1 Interpretation the Results

The results of this study are discussed with reference to the initial hypotheses, which focused on the overall capacity of a machine learning model defined by constraints on a TinyML platform. This model is oriented towards the classification of seismic and non-seismic events using a reduced number of features. The results largely confirm the hypothesis presented at the beginning of the work, showing that the model can achieve a high degree of seismic activity identification even with the restriction of using only seven of the twenty-five identified features. The model's accuracy of 93% reiterates its efficiency under these constraints. This suggests that the selected features incorporated sufficient information about the seismic and non-seismic signals to allow generalization beyond the training data.

Therefore, the evaluation of the results received proved that the general performance of offered methods is rather high by such criteria as accuracy, precision and recall, but certain deviations from expectations regarding the errors distribution are received. The problem of false positives and false negatives shows that the further situation is rather intricate because the input signals are rather difficult and similar. Perhaps, at the moment of devising this hypothesis, some researchers did not take into account that the general vibrations and seismic events could be very similar and the features of these signals are very similar. This means that feature selection becomes very important and also indicates that there is need for the improvement in feature extraction or addition of more features even when using TinyML.

5.3.2 Implications for Seismic Detection

The concrete conclusions of these findings are useful for the practice of seismic detection, especially in contexts where cost, energy needs, and processing rates are stringent conditions. Implementing a model with 93% accuracy on a TinyML platform signifies the feasibility of real-time seismic monitoring using low-power embedded systems. This is significant for structural safety and disaster mitigation, particularly in areas where basic, less costly, and resource-efficient seismic monitoring devices may not be easily accessible.

The use of small and relatively cheap sensors such as the MPU6050 allows for integrating the seismic detection system into building infrastructure, enabling real-time monitoring of structural conditions. Structures with such systems in place could potentially sense the onset of earthquake phenomena, providing occupants with additional seconds or minutes for protective actions or enabling safety measures such as shutting off gas or elevators. The high accuracy observed in this work implies that more advanced

systems of this type could be quite effective in practical applications.

In terms of earthquake preparedness, using these systems on a larger scale could be beneficial for increasing the density of warning systems, leading to better localized data on earthquake occurrences. Such data could contribute to more accurate larger-scale earthquake early warning systems, especially in areas not yet adequately monitored. Additionally, due to the low energy consumption and relatively lower cost of TinyML-based systems, this seismic detection technology can be implemented in both urban and rural settings.

However, the study also emphasizes the need for further research to address challenges such as low true positives and high false positives and false negatives. Including additional characteristics, applying more innovative sensors, and improving data analysis methods will be crucial for the further development of these systems. Overall, the progression of technologies and TinyML-based seismic detection systems presents a promising future for enhancing building safety and countermeasures against seismic hazards.

Chapter 6: Conclusion and Future Work

6.1 Summary of the Research

The objective of this study was to determine the possibility of detecting seismic activities using ML techniques on resource-constrained devices like those used in TinyML. The first goal was to create a classifier capable of identifying seismic vibrations while operating within the limited computational and memory resources available in TinyML environments.

The data selected for the study, alongside the formulation of a machine learning model optimized for such circumstances, proved appropriate. Standard metrics such as accuracy, precision, recall, and F1-Score were used to assess the model's performance. The developed model demonstrated a high accuracy rate, indicating that the TinyML framework can effectively detect seismic events even in highly constrained environments, as proven in this thesis.

The work generated new knowledge with regard to challenges as well as prospects for leveraging the models in low power devices. Nonetheless, the model gave a best near-optimal ATA (Accuracy-To-Resources) trade-off, which gives evidence that TinyML is suitable for detection of earthquake. This seems to imply a shiny future to the real-life application of the model in the near future for instance, earthquake early warning systems and structural health monitoring.

In addition, the study revealed where enhancement could be made with the aim of enhancing efficiency of the model in answering questions such as false positive and false negative. It presents out suggestions for more studies focusing on model improvement, features selection improvement, and sensors addition or signal processing techniques for the improvement of the detection.

6.2 Contributions to the Field

The key contributions of this research to the field of seismic detection include:

- **TinyML for Real-Time Seismic Detection:** This research demonstrates that TinyML can be effectively used for real-time seismic detection, offering a new approach to earthquake monitoring and early warning systems in an affordable, low-power manner.
- **Proof-of-Concept for ML on Constrained Platforms:** The research provides proof-of-concept that, with appropriate tuning, machine learning models can run efficiently on constrained hardware platforms like those targeted by TinyML applications. This paves the way for using low-power embedded devices in seismic monitoring, especially where traditional seismic networks are either prohibitive or physically unfeasible.
- **High Accuracy with Limited Resources:** The establishment of a seismic detection model that achieves high accuracy (93%) on a TinyML platform represents a significant technical advancement, demonstrating the possibility of designing powerful detection systems even with limited computational power.

- **Feature Selection and Optimization:** This research advances knowledge on feature selection and optimization under resource constraints, offering valuable insights for applying machine learning models in other low-resource settings.
- **Practical Applications in Sensor Technology:** The study presents a practical approach to improving the efficiency of real-time seismic detection systems using machine learning in sensor technology, highlighting the potential for creating portable, sharable seismic monitoring systems applicable almost anywhere.

6.3 Addressing Challenges and Refining the System

6.3.1 Challenges Encountered

Several challenges were encountered during this research:

- **Memory Limitation:** The most critical issue was the memory limitation inherent to the TinyML platform. The model complexity was constrained by the limited computational and storage resources, requiring careful consideration of feature space, computational complexity, and available extensions to remain within the hardware's computational envelope.
- **Sensor Noise:** The MPU6050 sensor, while effective in recording vibrational data, introduced noise that made it difficult to differentiate between seismic and non-seismic vibrations. This noise was particularly problematic when non-seismic signals closely resembled small seismic events, leading to misclassification.
- **Data Quality and Accessibility:** Access to high-quality seismic data, especially raw data, was a significant challenge. Seismic events are infrequent, making it difficult to obtain raw waveform data. Most institutions provide only processed or image-based data, limiting the ability to fine-tune detection models.

6.3.2 Solutions Implemented

To address these challenges, several solutions were implemented:

- **Memory Management:** Memory limitations were managed by tweaking the Tensor Arena allocation to 250KB, balancing memory and model requirements. Dynamic heap usage was also controlled in real-time to maintain system performance within tolerance levels.
- **Noise Reduction:** High-pass and low-pass filters were optimized to eliminate noise while preserving essential seismic signals. Parameters such as `HIGH_PASS_ALPHA` and `SMOOTHING_FACTOR` were adjusted to filter out noise without significantly affecting the signal. Additionally, sensor calibration was performed to improve reading accuracy by eliminating natural sensor biases.
- **Data Processing Optimization:** The data processing pipeline was optimized for speed and precision by segmenting data for further processing. The number and length of segments were chosen to

balance processing power and decision accuracy. Features were also scaled to increase processing speed and normalize input data for high-accuracy predictions.

- **System Testing:** Continuous testing was crucial for system refinement, allowing for gradual improvements based on real data responses. This iterative process helped identify and correct issues such as sensor shift, memory overflow, and unexpected model behavior, ultimately improving system performance.

6.4 Lessons Learned

Several important lessons were learned during the research process:

- The importance of efficient memory management in constrained environments.
- The need for advanced noise reduction techniques in sensor-based systems.
- The value of iterative testing and refinement in developing robust models.

6.4.1 Limitations

The research faced several limitations that impacted its scope and results. These limitations also provided valuable insights into the challenges of deploying machine learning models on resource-constrained platforms like TinyML.

6.4.2 Data Quality

The quality and variability of the data used in training and testing were significant limitations. Although the dataset was relatively large, it contained noise and artifacts that were difficult to distinguish from real seismic signals. This issue was exacerbated by the limited number of features available for use.

6.4.3 Hardware Limitations

The hardware used in the TinyML platform imposed constraints on the size and complexity of the model. Due to limited memory and computational capabilities, the model had to be simplified, reducing the number and complexity of neural network layers. This constraint may have limited the model's ability to learn and represent more complex patterns in the data.

6.4.4 Algorithm Performance

The algorithms used in this research were constrained by the need for real-time performance and efficiency. More detailed and mathematically rigorous methods could not be employed due to the computational limitations of the platform. While the filtering techniques used were effective, more advanced noise reduction methods could have improved performance.

6.4.5 Generalizability and Scalability

The model's generalizability and scalability were limited. Although the model was optimized for TinyML, it may perform worse on other platforms or in more complex seismic identification frameworks. The model's ability to handle larger datasets or more complex inputs was not tested, potentially limiting its applicability in larger-scale systems.

6.5 Future Research Directions

In light of the findings and limitations of this research, several directions for future research are recommended:

6.5.1 Enhancing Data Quality and Diversity

Future research should focus on obtaining higher-quality and more diverse datasets for evaluating the model's performance. Expanding the dataset to include a wider range of seismic instances and noise conditions would improve the model's portability across different settings. Additionally, improving data labeling and preprocessing techniques to filter out noise and artifacts would enhance classification accuracy.

6.5.2 Integrating Advanced Sensors

To overcome hardware limitations, future research should consider integrating more advanced sensors into the system. Higher-resolution or more sensitive sensors, such as gyroscopes, magnetometers, or higher-resolution accelerometers, could provide more specific information, improving the model's ability to detect small seismic events and exclude non-seismic interference.

6.5.3 Exploring More Sophisticated Algorithms

Future work should explore the development of more sophisticated algorithms that can operate within the constraints of TinyML platforms. Lighter versions of complex models, such as deeper neural networks, recurrent neural networks, or transformers, could help the model capture more complex features of the data. Additionally, refining algorithms to be more sensitive to environmental changes and noise levels would further enhance performance.

6.5.4 Expanding Real-Time Processing Capabilities

Expanding the system's real-time processing capabilities is another promising area for future research. This could involve increasing the data flow rate and volume, as well as exploring parallel processing techniques or more powerful edge computing hardware, to make the system suitable for large-scale seismic monitoring.

6.5.5 Exploring Scalability and Generalizability

Future research should also investigate ways to scale the current system and improve its generalizability. Testing the model in high-seismic-activity areas and with larger or more complex inputs could reveal its potential in regional or global seismic monitoring systems.

6.5.6 Investigating Alternative Platforms

Finally, future research should explore the possibility of enhancing the system beyond the TinyML framework. Platforms with greater computational capability or more flexible memory configurations could support more complex models and a broader range of features, potentially leading to a fusion of systems that combine the low energy consumption of TinyML with the superior computational power of other platforms.

In conclusion, the TinyML approach to seismic detection shows great promise, and there is significant potential for further advancements in this field. Addressing the limitations identified in this research and exploring new methods will contribute to the development of advanced, precise, and large-scale seismic monitoring systems, ultimately improving seismic readiness and safety.

References

[1] S. Dana, "Deep learning for earthquake detection and location," *arXiv preprint arXiv:798nt*, 2021.

[2] "Mpu6050 pinout." `https://www.researchgate.net/figure/MPU6050-with-the-pin-layout-and-axes-orientation_fig3_337664623`. Accessed: 28/08/2024.

[3] M. Esposito, L. Palma, A. Belli, L. Sabbatini, and P. Pierleoni, "Recent advances in internet of things solutions for early warning systems: A review," *Sensors*, vol. 22, no. 6, p. 2124, 2022.

[4] M.-A. Meier, Z. E. Ross, A. Ramachandran, A. Balakrishna, S. Nair, P. Kundzicz, Z. Li, J. Andrews, E. Hauksson, and Y. Yue, "Reliable real-time seismic signal/noise discrimination with machine learning," *Journal of Geophysical Research: Solid Earth*, December 2018.

[5] "Esp32-s3 pinout." `https://gr.mouser.com/new/espressif/espressif-esp32-s3-dev-kits/`. Accessed: 28/08/2024.

[6] R. Duggal, N. Gupta, A. Pandya, P. Mahajan, K. Sharma, T. Kaundal, and P. Angra, "Building structural analysis based internet of things network assisted earthquake detection," *Department of Computer Science and Engineering, National Institute of Technology, Hamirpur, Himachal Pradesh, India*, 2022.

[7] E. Bassetti and E. Panizzi, "Earthquake detection at the edge: Iot crowdsensing network," *Information*, vol. 13, no. 4, p. 195, 2022.

[8] A. Alphonsa and G. Ravi, "Earthquake early warning system by iot using wireless sensor networks," in *International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, (Chennai, India), pp. 1201–1205, 2016.

[9] S. E. Philip and M. H. Santhi, "Peak ground acceleration analysis using past earthquake data," in *Journal of Physics: Conference Series*, vol. 1716, p. 012013, 2020.

[10] T. Zainab, J. Karstens, and O. Landsiedel, "Lighteq: On-device earthquake detection with embedded machine learning," in *Proceedings of the 8th ACM/IEEE Conference on Internet of Things Design and Implementation (IoTDI '23)*, pp. 130–143, 2023.

[11] I. Khan, S. Choi, and Y.-W. Kwon, "Earthquake detection in a static and dynamic environment using supervised machine learning and a novel feature extraction method," *Sensors*, vol. 20, no. 3, p. 800, 2020.

[12] M. S. Abdalzaher, M. Krichen, D. Yiltas-Kaplan, I. B. Dhaou, and W. Y. H. Adoni, "Early detection of earthquakes using iot and cloud infrastructure: A survey," *Sustainability*, vol. 15, no. 15, p. 11713, 2023.

[13] Z. Li, "Recent advances in earthquake monitoring ii: Emergence of next-generation intelligent systems," *Earthquake Science*, vol. 34, no. 6, pp. 531–540, 2021.

[14] A. Pradeep, A. Latifov, A. Yodgorov, and N. Mahkamjonkhojizoda, "Hazard detection using custom esp32 microcontroller and lora," in *2023 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE)*, (Dubai, United Arab Emirates), pp. 36–40, 2023.

[15] J. Lee, I. Khan, S. Choi, and Y.-W. Kwon, "A smart iot device for detecting and responding to earthquakes," *Electronics*, vol. 8, no. 12, p. 1546, 2019.

[16] M. Falanga, E. D. Lauro, S. Petrosino, D. Rincon-Yanez, and S. Senatore, "Semantically enhanced iot-oriented seismic event detection: An application to colima and vesuvius volcanoes," *IEEE Internet of Things Journal*, vol. 9, no. 12, pp. 9789–9803, 2022.

[17] O. M. Saad, G. Huang, Y. Chen, A. Savvaidis, S. Fomel, N. Pham, and Y. Chen, "Scalodeep: A highly generalized deep learning framework for real-time earthquake detection," *Journal of Geophysical Research: Solid Earth*, 2021.

[18] "onsetwforms_meier19jgr_pub1_0_wojp.h5." `https://scedc.caltech.edu/data/deeplearning.html`. Accessed: 28/08/2024.

[19] K. Tsiakmakis, V. Delimaras, A. T. Hatzopoulos, and M. S. Papadopoulou, "Displacement measurement system and control of ionic polymer metal composite actuator," *WSEAS Transactions on Systems and Control*, vol. 18, pp. 144–153, 2023.

[20] K. Tsiakmakis, V. Delimaras, M. S. Papadopoulou, and A. T. Hatzopoulos, "Measuring displacement of ipmc actuator," in *4th International Conference in Electronic Engineering, Information Technology and Education*, (Thessaloniki, Greece), pp. 154–158, May 19 2023.

[21] M. S. Abdalzaher, H. A. Elsayed, M. M. Fouda, and M. M. Salim, "Employing machine learning and iot for earthquake early warning system in smart cities," *Energies*, vol. 16, no. 1, p. 495, 2023.

[22] O. M. Saad and Y. Chen, "Earthquake detection and p-wave arrival time picking using capsule neural network," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 59, no. 7, pp. 6234–6243, 2021.

[23] S. Prakash, M. Stewart, C. Banbury, M. Mazumder, P. Warden, and B. Plancher, "Is tinyml sustainable? assessing the environmental impacts of machine learning on microcontrollers," *Communications of the ACM (CACM)*, vol. 64, no. 11, pp. 70–77, 2021.

[24] C. Oikonomou, H. Haralambous, S. Pulinets, A. Khadka, S. Paudel, V. Barta, B. Muslim, K. Kourtidis, A. Karagioras, and S. Inyurt, "Investigation of pre-earthquake ionospheric and atmospheric disturbances for three large earthquakes in mexico," *Geosciences*, vol. 11, no. 16, 2021.

[25] Y.-M. Wu, D. Chen, T.-L. Lin, C.-Y. Hsieh, T.-L. Chin, W.-Y. Chang, W.-S. Li, and S.-H. Ker, "A high-density seismic network for earthquake early warning in taiwan based on low cost sensors," *Seismological Research Letters*, vol. 84, pp. 1048–1054, 2013.

[26] "Google colaboratory." `https://colab.google/`. Accessed: 28/08/2024.

[27] "Visual studio." `https://visualstudio.microsoft.com/`. Accessed: 28/08/2024.

[28] "Esp32 documentation." https://docs.espressif.com/projects/esp-idf/en/stable/esp32s3/hw-reference/esp32s3/user-guide-devkitc-1.html. Accessed: 28/08/2024.

[29] "Convolutional neural network (cnn) in machine learning." https://www.geeksforgeeks.org/convolutional-neural-network-cnn-in-machine-learning/. Accessed: 28/08/2024.

[30] S. M. Mousavi and G. C. Beroza, "Machine learning in earthquake seismology," *Annual Review of Earth and Planetary Sciences*, vol. 51, pp. 105–129, 2023.

[31] Z. Li, M.-A. Meier, E. Hauksson, Z. Zhan, and J. Andrews, "Machine learning seismic wave discrimination: Application to earthquake early warning," *Geophysical Research Letters*, vol. 45, no. 10, 2018.

[32] "Comos virtual data center." https://www.strongmotioncenter.org/vdc/scripts/default.plx. Accessed: 28/08/2024.

[33] "Platform.io." https://platformio.org/. Accessed: 28/08/2024.

[34] "Tensorflow lite documentation." https://www.tensorflow.org/lite/microcontrollers. Accessed: 28/08/2024.

Appendix A: Machine Learning Training

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D,
Flatten, Dense
from tensorflow.keras.optimizers import Adam
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np

# Define the function to balance the data
def balance_data(df1, df2):
    min_samples = min(len(df1), len(df2))
    df1_balanced = df1.sample(n=min_samples, random_state=42)
    df2_balanced = df2.sample(n=min_samples, random_state=42)
    return pd.concat([df1_balanced, df2_balanced]).reset_index(drop=True)

# Load your data (replace the paths with your actual paths)
quake_data = pd.read_csv('/content/drive/MyDrive/4.Essentials/10th_semester/
3.Research_process/5.Data/03_SCEDC_Data/02 Data Analysis/quake_data.csv')

noise_data = pd.read_csv('/content/drive/MyDrive/4.Essentials/10th_semester/
3.Research_process/5.Data/03_SCEDC_Data/02 Data Analysis/noise_data.csv')

# Selected features
selected_features = ['pa_0', 'pa_1', 'pa_2', 'pa_3', 'zcr_0',
'zcr_1', 'zcr_2', 'zcr_3', 'skew_0', 'skew_1', 'skew_2',
'skew_3', 'kurt_0', 'kurt_1', 'kurt_2', 'kurt_3', 'cav_0',
'cav_1', 'cav_2', 'cav_3', 'zhr_0', 'zhr_1', 'zhr_2', 'zhr_3',
'tauc_0', 'tauc_1', 'tauc_2', 'tauc_3']

# Normalize the data
scaler = MinMaxScaler()
quake_data[selected_features] = scaler.fit_transform
(quake_data[selected_features])
noise_data[selected_features] = scaler.transform
(noise_data[selected_features])

# Add labels to the data
quake_data['label'] = 1
noise_data['label'] = 0
```

```
# Balance the data
balanced_df = balance_data(quake_data, noise_data)

# Prepare the data for training
X = balanced_df[selected_features].values
y = balanced_df['label'].values

# Reshape for Conv2D: [samples, height, width, channels]
X = balanced_df[selected_features].values.reshape(-1, len
(selected_features) // 4, 4, 1)

# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y,
test_size=0.2, random_state=42)

from tensorflow.keras.layers import Conv2D, MaxPooling2D,
Flatten, Dense, Dropout, BatchNormalization
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam

# Define the model
model = Sequential([
    Conv2D(32, kernel_size=(2, 2), activation='relu',
    input_shape=(len(selected_features) // 4, 4, 1)),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001),
loss='binary_crossentropy', metrics=['accuracy'])

model.summary()
# Train the model
history = model.fit(X_train, y_train, epochs=10, batch_size=32,
validation_data=(X_val, y_val))
model.save('EEW_model.keras')
```

```
# Convert the model to TensorFlow Lite format
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the TensorFlow Lite model
with open('EEW_model.tflite', 'wb') as f:
    f.write(tflite_model)
```

Appendix B: Model Evaluation

```
import numpy as np
from sklearn.metrics import confusion_matrix, roc_curve, auc
import seaborn as sns
import matplotlib.pyplot as plt

# Evaluation Code
# Load the training history
loss = history.history['loss']
val_loss = history.history['val_loss']
accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']

# Number of epochs
epochs = range(1, len(loss) + 1)

# Plot training and validation loss
plt.figure(figsize=(14, 5))

plt.subplot(1, 2, 1)
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Plot training and validation accuracy
plt.subplot(1, 2, 2)
plt.plot(epochs, accuracy, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
```

```
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout()
plt.show()

# Evaluate the model to get predictions
y_pred_prob = model.predict(X_val).ravel()  # Flatten the output to 1D array
y_pred = (y_pred_prob > 0.5).astype("int32")

# Confusion Matrix
cm = confusion_matrix(y_val, y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=['Noise', 'Quake'], yticklabels=['Noise', 'Quake'])

plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# ROC Curve
fpr, tpr, _ = roc_curve(y_val, y_pred_prob)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2,
label=f'ROC curve (area = {roc_auc:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

Appendix C: Model Conversion from .h5 to .tflite

```
import tensorflow as tf
```

```
# Load the existing model
model = tf.keras.models.load_model('/content/drive/MyDrive/
4.Essentials/10th_semester/3.Research_process/5.Data/
03_SCEDC_Data/04 Models/seismic_model_dense.h5')

# Convert the model to TensorFlow Lite format
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_model = converter.convert()

# Save the converted model
tflite_model_path = '/content/drive/MyDrive/4.Essentials/10th_semester
/3.Research_process/5.Data/03_SCEDC_Data/04 Models/seismic_model_dense.tflite'
with open(tflite_model_path, 'wb') as f:
    f.write(tflite_model)

print("Model has been successfully converted and saved!")
```

Appendix D: TensorFlow Lite Inspection

```
import tensorflow as tf

# Load the TFLite model and allocate tensors
interpreter = tf.lite.Interpreter(model_path="EEW_model.tflite")
interpreter.allocate_tensors()

# Get input and output details
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

# Print input details
print("Input Details:")
for detail in input_details:
    print(f"Name: {detail['name']}")
    print(f"Shape: {detail['shape']}")
    print(f"Type: {detail['dtype']}")
    print()

# Print output details
print("Output Details:")
for detail in output_details:
```

```
    print(f"Name: {detail['name']}")
    print(f"Shape: {detail['shape']}")
    print(f"Type: {detail['dtype']}")
    print()

# Get tensor details
tensor_details = interpreter.get_tensor_details()
print("Tensor Details:")
for detail in tensor_details:
    print(detail)
    print()
```

Appendix E: TinyML Code

```
// General libraries
#include <Wire.h>
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <Arduino.h>
#include <SPIFFS.h>

// TensorFlow Lite Libraries
#include "tensorflow/lite/micro/micro_mutable_op_resolver.h"
#include "tensorflow/lite/micro/micro_interpreter.h"
#include "tensorflow/lite/schema/schema_generated.h"
#include "tensorflow/lite/micro/kernels/micro_ops.h"
#include "tensorflow/lite/micro/all_ops_resolver.h"
#include <TensorFlowLite_ESP32.h>

// Model in .h file
#include "EEW_model.h"

// MPU6050 setup
Adafruit_MPU6050 mpu;

const int numSamples = 100;
const int numWindows = 20;

const float CALM_THRESHOLD = 0.2;
const float NOISE_THRESHOLD = 0.6;

// Acceleration arrays
```

```
float accelerationX[numSamples * numWindows];
float accelerationY[numSamples * numWindows];
float accelerationZ[numSamples * numWindows];

float meanX = 0, meanY = 0, meanZ = 0;

const tflite::Model* model = tflite::GetModel(EEW_model_tflite);
tflite::MicroInterpreter* interpreter;
TfLiteTensor* input;
TfLiteTensor* output;

constexpr int kTensorArenaSize = 250 * 1024;
alignas(16) uint8_t tensor_arena[kTensorArenaSize];

class CustomErrorReporter : public tflite::ErrorReporter {
public:
  int Report(const char* format, va_list args) override {
    char buf[256];
    vsnprintf(buf, sizeof(buf), format, args);
    Serial.println(buf);
    return 0;
  }
};

CustomErrorReporter custom_error_reporter;
tflite::ErrorReporter* error_reporter = &custom_error_reporter;

void calibrateSensor() {
  Serial.println("Calibrating...");
  float sumX = 0, sumY = 0, sumZ = 0;
  for (int i = 0; i < 100; i++) {
    sensors_event_t a, g, temp;
    mpu.getEvent(&a, &g, &temp);
    sumX += a.acceleration.x;
    sumY += a.acceleration.y;
    sumZ += a.acceleration.z;
    delay(10);
  }
  meanX = sumX / 100;
  meanY = sumY / 100;
  meanZ = sumZ / 100;
  float gravityOffset = sqrt(meanX * meanX + meanY * meanY + meanZ * meanZ);
  Serial.printf("Calibration offsets - X: %f, Y: %f, Z: %f,
```

```
  Gravity Offset: %f\n", meanX, meanY, meanZ, gravityOffset);
}

#define SMOOTHING_FACTOR 0.1
#define HIGH_PASS_ALPHA 0.5

float smoothProbability(float currentProb, float newProb) {
  return (SMOOTHING_FACTOR * newProb) +
  ((1 - SMOOTHING_FACTOR) * currentProb);
}

float highPassFilter(float currentVal, float prevVal,
float alpha = HIGH_PASS_ALPHA) {
  static float prevOutput = 0;
  float output = alpha * (prevOutput + currentVal - prevVal);
  prevOutput = output;
  return output;
}

float lowPassFilter(float currentVal, float prevVal,
float alpha = 0.1) {
  return (alpha * currentVal) + ((1 - alpha) * prevVal);
}

void setup() {
  Serial.begin(115200);
  while (!Serial) {
    delay(10);
  }

  Serial.println("Adafruit MPU6050 test!");

  if (!SPIFFS.begin(true)) {
    Serial.println("An error has occurred while mounting SPIFFS");
    return;
  }
  Serial.println("SPIFFS mounted successfully");

  if (!mpu.begin()) {
    Serial.println("Failed to find MPU6050 chip");
    while (1) {
      delay(10);
    }
```

```
  }
  Serial.println("MPU6050 Found!");

  mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
  mpu.setFilterBandwidth(MPU6050_BAND_94_HZ);

  calibrateSensor();

  if (model->version() != TFLITE_SCHEMA_VERSION) {
    Serial.printf("Model provided is schema version %d
    not equal to supported version %d.\n", model->version(),
    TFLITE_SCHEMA_VERSION);
    return;
  }
  Serial.println("Model version is correct");

  tflite::AllOpsResolver resolver;
  static tflite::MicroMutableOpResolver<10> micro_op_resolver;
  micro_op_resolver.AddConv2D();
  micro_op_resolver.AddMaxPool2D();
  micro_op_resolver.AddFullyConnected();
  micro_op_resolver.AddReshape();
  micro_op_resolver.AddSoftmax();

  static tflite::MicroInterpreter static_interpreter(model,
  resolver, tensor_arena, kTensorArenaSize, error_reporter);
  interpreter = &static_interpreter;

  Serial.println("Interpreter initialized");

  TfLiteStatus allocate_status = interpreter->AllocateTensors();
  if (allocate_status != kTfLiteOk) {
    Serial.println("Failed to allocate tensors!");
    while (1) {
      delay(10);
    }
  }
  Serial.println("Tensors allocated");

  input = interpreter->input(0);
  output = interpreter->output(0);
}
```

```
void printMemoryUsage() {
  uint32_t freeHeap = esp_get_free_heap_size();
  uint32_t totalHeap = ESP.getHeapSize();
  uint32_t usedHeap = totalHeap - freeHeap;

  Serial.printf("Total Heap: %d bytes\tUsed Heap: %d
  bytes\tFree Heap: %d bytes\n", totalHeap, usedHeap, freeHeap);
}

float calculateMagnitude(float x, float y, float z) {
  return sqrt(x * x + y * y + z * z);
}

float calculatePeakAmplitude(float *x, float *y, float *z, int len) {
  float peak = 0;
  for (int i = 0; i < len; i++) {
    float magnitude = calculateMagnitude(x[i],y[i],z[i]);
    if (abs(magnitude) > peak) {
      peak = abs(magnitude);
    }
  }
  return peak;
}

int calculateZeroCrossingRate(float *x, float *y, float *z, int len) {
  int zeroCrossings = 0;
  float prevMagnitude = calculateMagnitude(x[0], y[0], z[0]);

  for (int i = 1; i < len; i++) {
    float currentMagnitude = calculateMagnitude(x[i], y[i], z[i]);
    if ((prevMagnitude > 0 && currentMagnitude < 0) ||
    (prevMagnitude < 0 && currentMagnitude > 0)) {
      zeroCrossings++;
    }
    prevMagnitude = currentMagnitude;
  }
  return zeroCrossings;
}

float calculateSkewness(float *x, float *y, float *z, int len) {
  float mean = 0;
  float m3 = 0;
  float m2 = 0;
```

```
  for (int i = 0; i < len; i++) {
    float magnitude = calculateMagnitude(x[i],y[i],z[i]);
    mean+=magnitude;
  }
  mean /= len;

  for (int i = 0; i < len; i++) {
   float magnitude = calculateMagnitude(x[i],y[i],z[i]);
    m3 += pow(magnitude - mean, 3);
    m2 += pow(magnitude - mean, 2);
  }
  m3 /= len;
  m2 /= len;

  if (m2 == 0) return NAN;

  float skewness = m3 / pow(m2, 1.5);
  return skewness;
}

float calculateKurtosis(float *x, float *y, float *z, int len) {
  float mean = 0;
  float m4 = 0;
  float m2 = 0;

  for (int i = 0; i < len; i++) {
    float magnitude = calculateMagnitude(x[i],y[i],z[i]);
    mean += magnitude;
  }
  mean /= len;

  for (int i = 0; i < len; i++) {
    float magnitude = calculateMagnitude(x[i],y[i],z[i]);
    m4 += pow(magnitude - mean, 4);
    m2 += pow(magnitude - mean, 2);
  }
  m4 /= len;
  m2 /= len;

  if (m2 == 0) return NAN;

  float kurtosis = m4 / (m2 * m2);
```

```
  return kurtosis;
}

float calculateCAV( float *x, float *y, float *z, int len) {
  float cav = 0;
  for (int i = 0; i < len; i++) {
     float magnitude = calculateMagnitude(x[i],y[i],z[i]);
     cav += abs(magnitude);
  }
  return cav;
}

float calculateZHR(float *vertical, float *horizontal1,
float *horizontal2, int len, float alpha) {

  float filteredVertical[len];
  float filteredHorizontal1[len];
  float filteredHorizontal2[len];

  filteredVertical[0] = vertical[0];
  filteredHorizontal1[0] = horizontal1[0];
  filteredHorizontal2[0] = horizontal2[0];

  for (int i = 1; i < len; i++) {
    filteredVertical[i] = alpha * (vertical[i] - vertical[i - 1]);
    filteredHorizontal1[i] = alpha * (horizontal1[i] - horizontal1[i - 1]);
    filteredHorizontal2[i] = alpha * (horizontal2[i] - horizontal2[i - 1]);
  }

  float maxV = calculatePeakAmplitude(filteredVertical,
  filteredHorizontal1, filteredHorizontal2, len);

  float maxH1 = calculatePeakAmplitude(filteredHorizontal1,
  filteredHorizontal1, filteredHorizontal2, len);

  float maxH2 = calculatePeakAmplitude(filteredHorizontal2,
  filteredHorizontal1, filteredHorizontal2, len);

  float maxH = sqrt(maxH1 * maxH1 + maxH2 * maxH2);

  if (maxH == 0) return 0;
  float zhr = maxV / maxH;
  return zhr;
```

```
}

float calculateTauC(float *displacementX, float *displacementY,
float *displacementZ,
                    float *velocityX, float *velocityY,
                    float *velocityZ, int len) {
  float integratedSquaredDisplacement = 0;
  float integratedSquaredVelocity = 0;

  for (int i = 0; i < len; i++) {
    float magnitudeDisplacement = calculateMagnitude(displacementX[i],
    displacementY[i], displacementZ[i]);
    float magnitudeVelocity = calculateMagnitude(velocityX[i],
    velocityY[i], velocityZ[i]);
    integratedSquaredDisplacement += magnitudeDisplacement
    * magnitudeDisplacement;
    integratedSquaredVelocity += magnitudeVelocity *
    magnitudeVelocity;
  }

  if (integratedSquaredVelocity == 0) return NAN;

  float tauC = sqrt(integratedSquaredDisplacement / integratedSquaredVelocity);
  return tauC;
}

void localMinMaxScale(float* features, int len) {
    float min = features[0];
    float max = features[0];

    for (int i = 1; i < len; ++i) {
        if (features[i] < min) {
            min = features[i];
        }
        if (features[i] > max) {
            max = features[i];
        }
    }
    for (int i = 0; i < len; ++i) {
        features[i] = (features[i] - min) / (max - min);
    }
}
```

```
bool hasNan(float* data, size_t len) {
  for (size_t i = 0; i < len; ++i) {
    if (isnan(data[i])) return true;
  }
  return false;
}

void processModel(float* features, size_t len) {
    if (hasNan(features, len)) {
        Serial.println("NaN detected in input features");
        for (size_t i = 0; i < len; ++i) {
            Serial.printf("Feature %zu: %f\n", i, features[i]);
        }
        return;
    }

    for (int i = 0; i < len; ++i) {
        input->data.f[i] = features[i];
    }

    TfLiteStatus invoke_status = interpreter->Invoke();
    if (invoke_status != kTfLiteOk) {
        Serial.println("Error invoking the interpreter");
        return;
    }
}

void computeFeatures(float* sensorData, float* features,
int numSegments, int segmentLength) {
  for (int segment = 0; segment < numSegments; ++segment) {
    int start = segment * segmentLength;

    float segmentDataX[segmentLength];
    float segmentDataY[segmentLength];
    float segmentDataZ[segmentLength];
    for (int i = 0; i < segmentLength; i++) {
      segmentDataX[i] = sensorData[(start + i) * 3];
      segmentDataY[i] = sensorData[(start + i) * 3 + 1];
      segmentDataZ[i] = sensorData[(start + i) * 3 + 2];
    }

    features[segment] = calculatePeakAmplitude(segmentDataX,
    segmentDataY, segmentDataZ, segmentLength);
```

```
    features[numSegments + segment] = calculateZeroCrossingRate
    (segmentDataX, segmentDataY, segmentDataZ, segmentLength);
    features[2 * numSegments + segment] = calculateSkewness
    (segmentDataX, segmentDataY, segmentDataZ, segmentLength);
    features[3 * numSegments + segment] = calculateKurtosis
    (segmentDataX, segmentDataY, segmentDataZ, segmentLength);
    features[4 * numSegments + segment] = calculateCAV
    (segmentDataX, segmentDataY, segmentDataZ, segmentLength);

    features[5 * numSegments + segment] = calculateZHR
    (segmentDataX, segmentDataY, segmentDataZ, segmentLength, HIGH_PASS_ALPHA);

    features[6 * numSegments + segment] = calculateTauC
    (segmentDataX, segmentDataY, segmentDataZ,

        segmentDataX, segmentDataY, segmentDataZ, segmentLength);
  }

  localMinMaxScale(features, numSegments * 7);
}

void loop() {
  static float prevX = 0, prevY = 0, prevZ = 0;
  static float smoothedProb = 0;
  static float filteredProb = 0;

  float sensorData[numSamples * 3];
  float features[28];

  for (int w = 0; w < numWindows; w++) {
    for (int i = 0; i < numSamples; i++) {
      sensors_event_t a, g, temp;
      mpu.getEvent(&a, &g, &temp);

      float calibratedX = a.acceleration.x - meanX;
      float calibratedY = a.acceleration.y - meanY;
      float calibratedZ = a.acceleration.z - meanZ;

      calibratedX = highPassFilter(calibratedX, prevX);
      calibratedY = highPassFilter(calibratedY, prevY);
      calibratedZ = highPassFilter(calibratedZ, prevZ);

      prevX = calibratedX;
```

```
      prevY = calibratedY;
      prevZ = calibratedZ;

      sensorData[i * 3] = calibratedX;
      sensorData[i * 3 + 1] = calibratedY;
      sensorData[i * 3 + 2] = calibratedZ;
    }

    int numSegments = 4;
    int segmentLength = numSamples / numSegments;

    computeFeatures(sensorData, features, numSegments, segmentLength);

    processModel(features, 28);

    float rawProbability = output->data.f[0];
    smoothedProb = smoothProbability(smoothedProb, rawProbability);
    filteredProb = lowPassFilter(smoothedProb, filteredProb);

    if (smoothedProb < CALM_THRESHOLD) {
        Serial.print("Calm wave. ");
    } else if (smoothedProb < NOISE_THRESHOLD) {
        Serial.print("Noise Wave Detected. ");
    } else {
        Serial.print("Seismic Wave Detected! ");
    }

    printMemoryUsage();
  }
  delay(10);
}
```