RSET
RAJAGIRI SCHOOL OF
ENGINEERING & TECHNOLOGY
(AUTONOMOUS)

Project Report On

DolphinAttack Detection

Submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology

in

Computer Science and Engineering

By

Nandana S Pillai (U2103147)

Ritu Maria Ajith (U2103176)

Shaun Mammen John (U2103194)

Shreya Veeraraghav (U2103198)

Under the guidance of

Dr. Anita John

Computer Science and Engineering
Rajagiri School of Engineering & Technology (Autonomous)
(Parent University: APJ Abdul Kalam Technological University)
Rajagiri Valley, Kakkanad, Kochi, 682039
April 2025

CERTIFICATE

This is to certify that the project report entitled ***"DolphinAttack Detection"*** *is a bonafide record of the work done by* ***Nandana S Pillai (U2103147), Ritu Maria Ajith (U2103176), Shaun Mammen John (U2103194), Shreya Veeraraghav (U2103198)****, submitted to the Rajagiri School of Engineering & Technology (RSET) (Autonomous) in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology (B. Tech.) in Computer Science and Engineering during the academic year 2025.*

Dr. Anita John
Project Guide
Associate Professor
Dept. of CSE
RSET

Ms. Sangeetha Jamal
Project Coordinator
Assistant Professor
Dept. of CSE
RSET

Dr. Preetha K G
Professor & HoD
Dept. of CSE
RSET

ACKNOWLEDGMENT

We wish to express our sincere gratitude towards **Rev. Dr. Jaison Paul Mulerikkal CMI**, Principal of RSET, and **Dr. Preetha K G**, Professor & Head of the Department of Computer Science and Engineering, for providing us with the opportunity to undertake our project, "DolphinAttack Detection."

We are highly indebted to our project coordinator, **Ms. Sangeetha Jamal**, Assistant Professor, Dept. of CSE, and project guide, **Dr. Anita John**, Associate Professsor, Dept. of CSE, for their patience and all the priceless advice and wisdom they have shared with us.

Last but not the least, we would like to express our sincere gratitude towards all other teachers and friends for their continuous support and constructive ideas.

Nandana S Pillai
Ritu Maria Ajith
Shaun Mammen John
Shreya Veeraraghav

Abstract

DolphinAttack is a technique that exploits the vulnerabilities of voice-controlled systems (VCSs) to execute malicious, inaudible commands. By modulating voice commands on ultrasonic carriers, it makes the commands inaudible, bypassing the human audible range. This attack can be used to manipulate VCSs without the user's knowledge, leading to a range of security risks. Therefore, this highlights the security challenges posed by the voice-controlled devices and underscores the need for countermeasures to protect users from these silent threats. Our project aims at detecting such attacks by developing a software defense that can classify ultrasonic audios from benign ones based on the comparison of the features that are extracted from both audios using different machine learning algorithms and then attempt to find the best model that can perform well in the scenario of an attack. Furthermore, our software alerts the users with a message and identifies the commands embedded in the ultrasonic signal transmitted to the voice assistant.

Contents

List of Abbreviations

VCS - Voice-Controlled System

AM - Amplitude Modulation

SR - Speech Recognition

MEMS - Micro Electro Mechanical System

LPF - Low-Pass Filter

ADC - Analog to Digital Converter

TTS - Text-to-Speech

SPL - Sound Pressure Level

SVM - Support Vector Machine

FAR - False Acceptance Rate

FRR - False Rejection Rate

CVAE - Convolutional Variational AutoEncoder

VCD - Voice-Controlled Device

LTAS - Long-Time Average Spectrum

STFT - Short-Time Fourier Transform

RFE - Recursive Feature Elimination

AUC - Area under Curve

EER - Equal Error Rate

AI - Artificial Intelligence

HVC - Hidden Voice Commands

SVC - Support Vector Classification

ASR - Automatic Speech Recognition Systems

GFCC - Gammatone Frequency Cepstral Coefficient

TAR - True Acceptance Rate

MFCC - Mel-Frequency Cepstral Coefficients

CNN - Convolutional Neural Network

KNN - K-Nearest Neighbours

MSE - Mean Squared Error

ReLU - Rectified Linear Unit

KL - Kullback-Leibler

XAI - Explainable AI

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Speech-recognition devices have become increasingly popular in voice assistants and are being used for a wide range of purposes, such as ordering food or calling a friend. Apart from the benefits, it was discovered that these systems are also able to respond to malicious and stealthy attacks that are unrecognizable to humans. For example, an attacker can send a physically inaudible voice command to a smart speaker and make it open the home door without the knowledge of the user. Humans are only able to detect sounds in the range between the frequencies, 20 Hz and 2 kHz. Any sounds outside this range are inaudible to humans. Inaudible voice attacks involve sending malicious commands at frequencies beyond the range of human hearing. These attacks manipulate devices that are programmed to recognize voice commands without the user being aware. By embedding voice commands into ultrasonic signals, attackers can silently control these devices from a distance. A group of researchers first demonstrated this attack in 2017, which they termed the Dolphinattack. They were able to make a call on an iPhone and activate Google Now. Users were also able to switch their phones to airplane mode and manipulate navigation systems in an Audi automobile without detection.

1.2 Problem Definition

In the present world, where there is a rise in the usage of voice-controlled systems, attackers are developing malicious methods to gain unauthorized control over these devices. This poses significant risks, including data theft, financial manipulation, and unauthorized access to confidential information, as this attack can bypass traditional defenses. Hence, the project aims to develop a model for detecting and classifying ultrasonic audio from benign audio in the scene of an attack on voice recognition systems that can be

inserted into voice-controllable systems right before the demodulation step is performed after the audio signal has been received.

1.3 Scope and Motivation

1.3.1 Scope

The scope of this research focuses on developing a software-based mitigation method to protect the voice-controlled devices (VCDs) from inaudible ultrasonic attacks like DolphinAttack. The proposed approach leverages the unique acoustic signatures introduced by the microphone's nonlinearity when exposed to ultrasonic frequencies. By analyzing these signatures, the system aims to distinguish between legitimate and malicious inaudible voice commands. The solution ensures compatibility with the existing VCDs without the need for additional hardware. In addition to improving the security of existing devices, this strategy provides a workable and deployable defense mechanism that can be used with a variety of voice-controlled systems.

1.3.2 Motivation

Voice-controlled devices like Amazon Echo and Google Home are vulnerable to ultrasonic or inaudible voice command attacks, which exploit microphone nonlinearity to issue covert commands that are inaudible to humans. These attacks pose serious security risks by allowing unauthorized control of devices. Despite the existence of hardware-based defenses, their practicality and cost prevent their widespread use. To enhance device security in a cost-effective and scalable way, a software-based solution that can detect and prevent these attacks instantly without necessitating hardware modifications is needed.

1.4 Objectives

The objectives of the project include:

1. **Audio Classification:** Detecting and classifying an inaudible voice command from an audible command in order to identify a possible event of DolphinAttack.

2. **Alert Generation :** In the event of an attack, users are alerted with a message.

3. **Command Recognition :** The command used for the attack is identified and displayed to the user along with the alert, when the classification result indicates an inaudible command.

1.5 Challenges

- Since the working of the microphone is simulated using Python code due to the non-availability of hardware equipments, it is difficult to study the impact of variations in distances on the attack.
- Collecting diverse and sufficient data to train and validate the model, covering various environments, devices, and command variations to ensure robustness is another challenge.

1.6 Assumptions

1. **No interference or distortion of ultrasonic signals in the operating environments:** It is assumed that ultrasonic signals transmitted by an attacker will travel with minimal interference or distortion within the environment, ensuring that they reach the target device clearly and effectively.
2. **Injected commands will be interpreted by the voice assistants as legitimate human voice commands:** It is assumed that the ultrasonic commands are accurately crafted to mimic legitimate human commands, allowing the voice assistant to interpret and execute them as if they were from an authorized user.
3. **No user authentication:** The target voice recognition system does not have robust authentication measures, meaning any recognizable voice command, regardless of the source, can trigger actions on the device without verifying user identity.

1.7 Societal / Industrial Relevance

This project has significant societal and industrial implications. It highlights the security risks in voice-controlled systems by demonstrating how inaudible commands can manipulate devices without the user's knowledge. In society, this has implications for privacy and

safety by blocking unauthorized, inaudible commands, as such attacks could compromise personal devices, affecting user trust in smart home and personal assistant technology. Industrially, defending against DolphinAttack helps tech companies fortify smart systems against ultrasonic manipulation, reducing cybersecurity risks for businesses. It also drives innovation in secure voice-recognition technology, which is crucial as voice interfaces become increasingly widespread in IoT and enterprise environments.

1.8 Organization of the Report

The report is organized as follows:
Chapter 1 introduces the problem of inaudible voice attacks and the necessity to develop a suitable defense. Chapter 2 explores the existing literature on the subject and highlights key points. Chapter 3 lists down the hardware and software resources used to develop the project. Chapter 4 provides an overview of the architecture of the software defense being undertaken. Chapter 5 details down the proposed methodology along with an elaborate description of the implementation strategies used to build the various components of the system proposed in the project. Chapter 6 reviews the contents of the report and provides a conclusion along with directions for future work.

Chapter 2

Literature Survey

2.1 DolphinAttack : Inaudible Voice Commands [1]

This paper identifies a security flaw in voice-controlled systems (VCS) that allows voice commands that are inaudible or ultrasonic to humans to control other devices covertly. The authors show that these "hidden" commands can initiate malicious actions, like visiting harmful websites, making unauthorized calls, or disabling wireless communication. By successfully testing systems like Alexa, Google Now, and Siri on a variety of devices, the study highlights the need for enhanced security in voice recognition technology and offers software and hardware defenses to lessen such attacks.

2.1.1 Background Model

Voice Controlled System

Voice-controlled systems (VCS) typically consist of three components:

1. Voice Capture
2. Speech Recognition
3. Command Execution

Sounds are recorded by the voice capture subsystem, which then filters and amplifies the signals before processing them in two stages. First, an activation phase, triggered by a wake word or key, initializes the recording. Once activated, the recognition phase converts spoken commands into text, usually using cloud-based machine-learning algorithms. Commands are then executed by system-defined actions. These VCS systems are vulnerable to inaudible ultrasonic attacks.

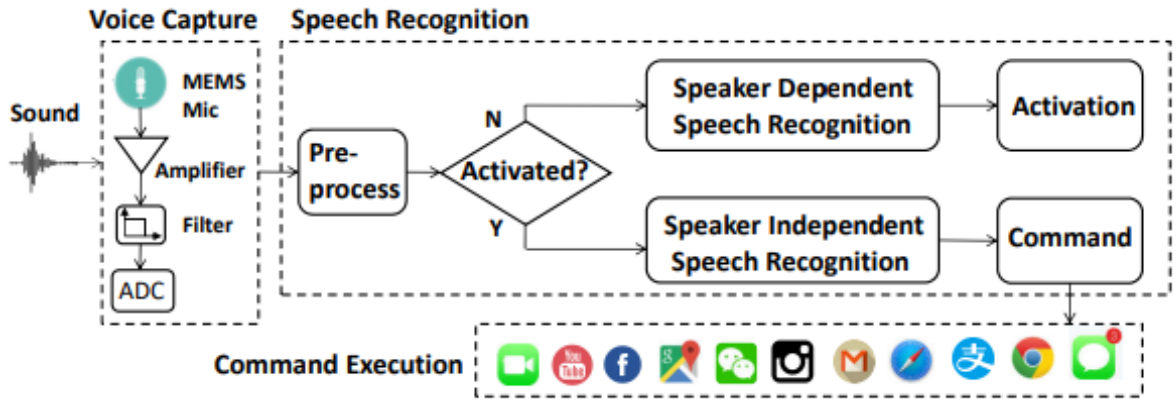


Figure 2.1: Architecture of Voice Controlled Systems

Microphone

The voice capture subsystem uses microphones, especially Micro Electro Mechanical Systems (MEMS) microphones, which convert sound waves into electrical signals. MEMS microphones dominate voice-controlled technology due to their small size and efficiency. These microphones, including the low-pass filters (LPFs) and analog-to-digital converters (ADC), are designed to capture sounds within the human hearing range (20 Hz to 20 kHz). Ultrasonic frequencies and other signals above this range are usually filtered out, but inaudible attacks are still possible.

Threat Model

The adversary's objective is to discretely inject unauthenticated commands into voice-controlled systems without user awareness. They assume no physical access to the device, no owner interaction, and rely solely on ultrasonic (inaudible) commands above 20 kHz to avoid detection. Attackers can use equipment such as ultrasonic capable speakers placed near the target device, allowing commands to be executed remotely and covertly.

2.1.2 Nonlinearity

The concept of DolphinAttack relies on modulating low-frequency commands onto an ultrasonic carrier and allowing the device's nonlinear microphone components to demodulate the signal. Because microphones and amplifiers introduce nonlinear effects, they can create new frequencies when ultrasonic signals are introduced. By using amplitude

modulation and carefully selecting frequencies, DolphinAttack downconverts the ultrasonic command signal, which remains after the low-pass filter, making it interpretable by voice-controlled systems while staying inaudible to humans.

2.1.3 Attack Design

DolphinAttack enables covert control of voice-controlled systems (VCSs) by injecting inaudible voice commands via ultrasonic modulation. To activate a device like Siri, it generates activation commands like "Hey Siri" through Text-to-Speech (TTS) brute force or by concatenating phonemes from recorded speech. This had a success rate of 39% in testing. Control commands like "call 911" follow activation without authentication checks, so they are injected more easily. Two transmitter types facilitate the attack: a powerful lab-based setup for robust testing and a portable smartphone-based design to expand attack feasibility in real-world scenarios. The portable setup, using a smartphone and ultrasonic transducer, has limited range than the powerful one, but demonstrates the attack's adaptability for everyday devices. Key parameters like modulation frequency and depth are carefully selected to ensure inaudibility and effective VCS response. This attack model shows the potential for widespread VCS vulnerability to silent, remote control.

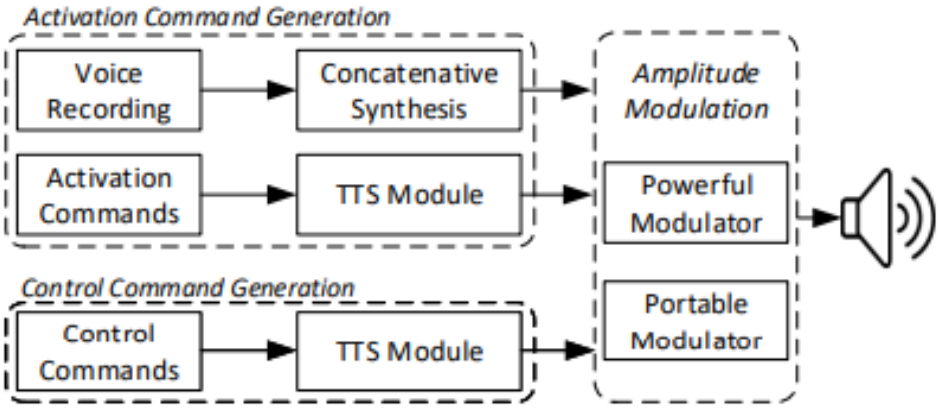


Figure 2.2: Design of DolphinAttack

2.1.4 Impact Quantification

DolphinAttack's effectiveness varies across languages, background noise levels, sound pressure levels (SPLs), and distances. Tested commands in five languages showed high

recognition rates, with English and Spanish achieving 100%. Noise affects control commands more than activation commands, with recognition rates dropping in louder settings. Higher SPLs improve signal-to-noise ratios and recognition accuracy, especially for the Apple Watch. Based on attack range, Apple devices respond from further distances than Samsung devices. Portable device attacks succeeded at close range but were limited by speaker frequency constraints.

2.1.5 Defenses

Two defense strategies proposed to counter DolphinAttack are:

1. **Hardware-Based Defense:** This involves enhancing microphones to block signals above 20 kHz and adding modules to detect and cancel inaudible voice commands before they reach the device.

2. **Software-Based Defense:** This involves distinguishing between legitimate and malicious modulated commands using machine learning classifiers, like SVM. This approach detects frequency characteristics that are unique (500–1000 Hz) and that are present in modulated attacks. Testing showed a 100% accuracy rate in distinguishing the signals.

These defenses together aim to prevent unauthorized, inaudible command injections.

The paper introduces an inaudible attack known as "DolphinAttack," on speech recognition (SR) systems like Siri, Google Now, and Alexa. It embeds the commands in ultrasonic frequencies, making them undetectable by humans using amplitude modulation (AM). To counter potential misuse, the authors proposed two defense methods, addressing both hardware and software aspects to protect SR systems from these stealthy attacks.

2.2 Inaudible Voice Commands: Long-Range Attack and Defense [2]

In this paper, the authors aim to address two primary objectives concerning inaudible voice command attacks on voice-controlled devices. The first goal is to extend the attack range of these inaudible commands, which prior research had limited to a maximum of 5 ft. due to audio leakage that could expose the attack. By innovating on the transmitter

design, the authors successfully achieve a longer attack range of up to 25 ft. . The second goal is to propose a software-based defense mechanism against these ultrasonic attacks. This defense utilizes an elliptical classifier to detect nonlinear artifacts introduced by these inaudible signals, this classifying software based mechanism was named LipRead. LipRead enables the identification and blocking of malicious voice commands without requiring hardware changes to the devices.

2.2.1 Acoustic NonLinearity

The concept of acoustic nonlinearity is central to the effectiveness of ultrasonic attacks. In theory, microphones and speakers are designed to operate linearly, meaning their output should be a direct, scaled version of their input signal. However, when ultrasonic frequencies (above 20 kHz) are introduced, many consumer-grade microphones exhibit nonlinear behavior. This nonlinearity causes high-frequency signals to leak into lower, audible frequencies due to a process known as frequency mixing. For example, two ultrasonic tones at frequencies (f1) and (f2) can combine to produce a signal at the difference frequency $\mid f2 - f1 \mid$, which may fall within the audible range, thus allowing ultrasonic commands to be "heard" by the microphone but remain inaudible to humans.

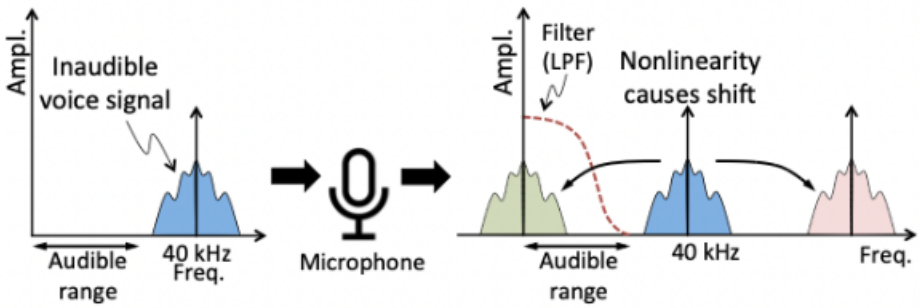


Figure 2.3: Acoustic nonlinearity

Short-Range vs. Long-Range Attacks

Short-range attacks, such as the DolphinAttack, are limited to a distance of about 5 ft. because increasing the transmission power to extend the range results in detectable audio leakage, making the attack audible. These attacks are typically feasible only when the attacker is in close proximity to the target device, such as inside the same room.

In contrast, LipRead's long-range attack achieves a significant increase in distance by employing an array of multiple speakers, each transmitting carefully segmented portions of the voice command spectrum. This multi-speaker approach reduces the audible leakage from any single speaker, allowing the aggregate signal to remain below the human hearing threshold. The results show that LipRead can successfully activate VCDs from distances up to 25 ft. for Amazon Echo and 30 ft. for smartphones, thus proving the viability of longer-range inaudible attacks.

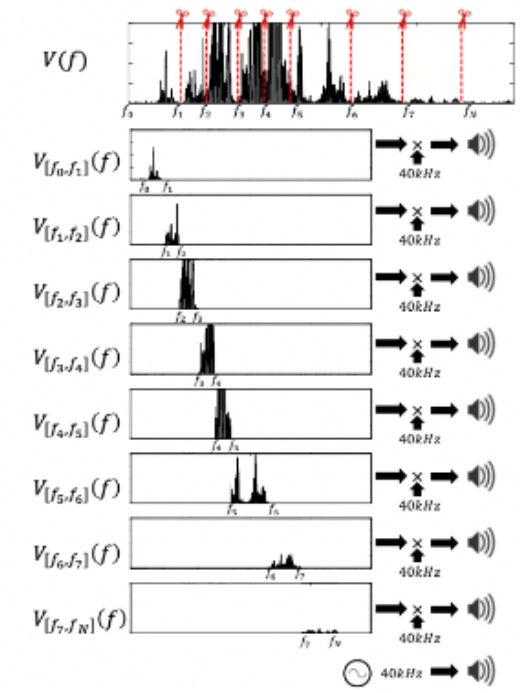


Figure 2.4: Audio segmenting based on frequencies

2.2.2 Results of the Long-Range Attack Methodology

- **Extended Range:** The attack was effective at distances of up to 25 ft. for Amazon Echo and 30 ft. for smartphones.
- **Inaudibility:** The system maintained inaudibility, with the audio leakage remaining 5-10 dB below the human hearing threshold.
- **High Success Rate:** Out of 984 commands sent to Amazon Echo, the attack achieved a high success rate, demonstrating reliable activation and command recognition.

2.2.3 LipRead Defense Design

The LipRead defense utilises key features identified in the natural voice signals to differentiate between malicious and benign audios. There are three main parameters that are closely monitored while building the defense:

1. **V^2 Correlation:** The defense system calculates a correlation coefficient between the fundamental voice frequency and its harmonics in lower frequencies. An ultrasonic signal would contain a strong correlation between the fundamental frequency and such harmonics in lower frequencies. Thus a higher correlation indicates higher possibility of ultrasonic attack.

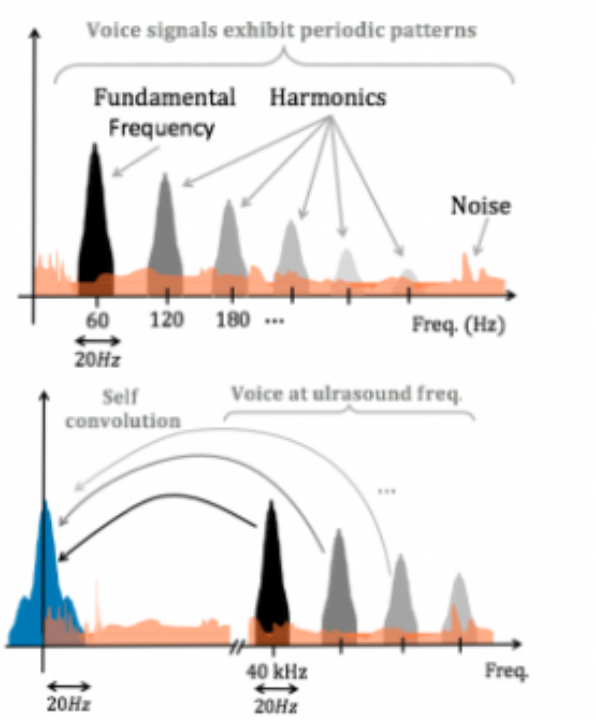


Figure 2.5: Legitimate voice in upper half, inaudible attack in lower half.

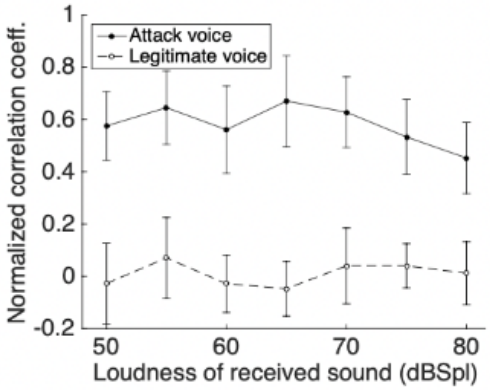


Figure 2.6: Correlation coefficient of attack (solid) vs. legitimate (dotted) signal.

2. **Amplitude Skew:** Often nonlinearity results in a skew in the amplitude distribution of the signal. Normal voice signals have an almost balanced distribution of positive and negative amplitudes, whereas ultrasonic signals that undergo nonlinear distortion due to nonlinearity show an amplitude skew towards positive amplitudes. The defense studies the ratio of the maximum to minimum amplitude to detect this imbalance.

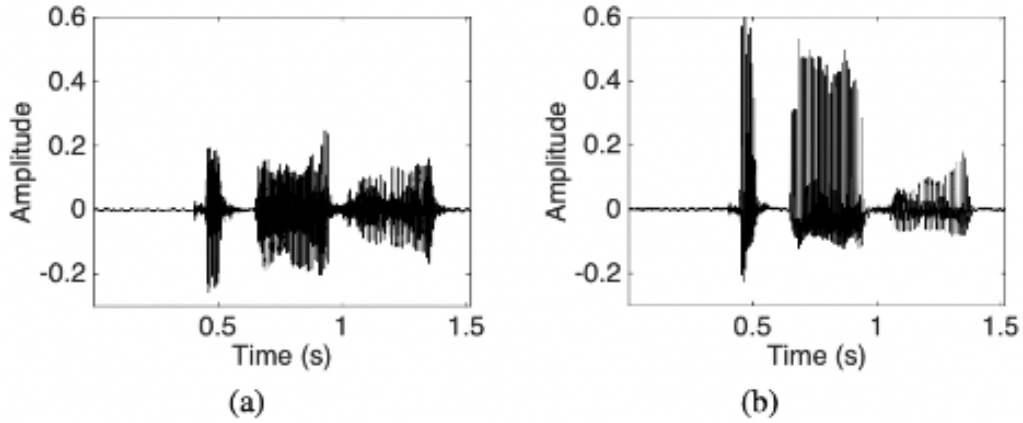


Figure 2.7: Amplitude skew of (a) legitimate voice signal and
(b) inaudible attack signal

3. **Power in the Sub-50Hz Band:** Legitimate human speech generally does not contain significant energy in frequencies below 50 Hz. However, nonlinear effects

can introduce new low-frequency components in the output signal of the microphone. The defense checks the power levels in this sub-50 Hz band, to detect anomalies that may indicate nonlinear effects.

2.2.4 Elliptical Classifier

The final stage of the LipRead defense employs an elliptical classifier that uses the features mentioned previously - V^2 correlation, amplitude skew, and power in the sub-50 Hz band, to distinguish between legitimate voice commands and inaudible attacks. The classifier was designed to minimize both the false acceptance rate (FAR) and false rejection rate (FRR), achieving 97% precision and 98% recall across various test scenarios. This software-only solution is particularly advantageous because it does not require additional hardware, making it deployable on existing VCDs.

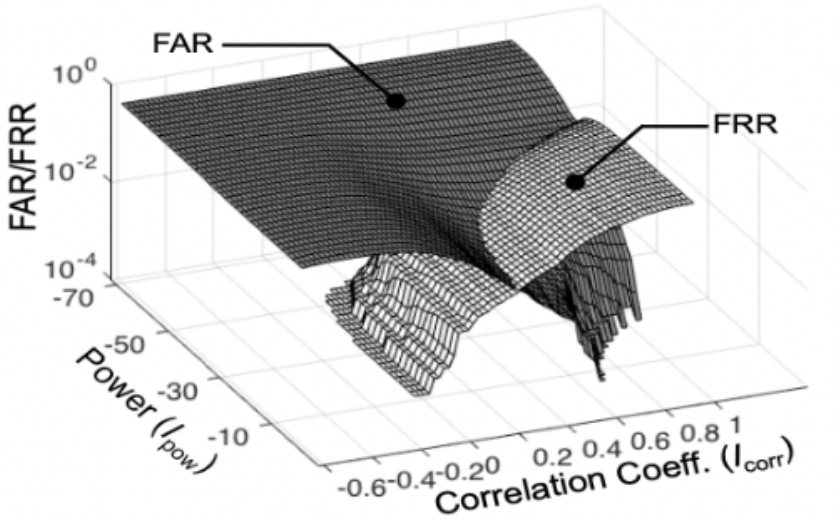


Figure 2.8: Elliptical Classifier

2.2.5 Key Results of the LipRead Defense

- **High Detection Accuracy:** The defense system demonstrated 98% precision and 99% recall, even in the presence of background noise and signal manipulation attempts by attackers.
- **Robustness:** The system maintained high detection rates across different environments and sound pressure levels, ensuring reliable defense against ultrasonic attacks.

- **No Hardware Changes:** The software-based solution is compatible with existing VCDs, making it a practical approach for enhancing device security.

2.3 Learning Normality is Enough : A Software-based Mitigation against Inaudible Voice Attacks [3]

To protect the existing and future devices from the threats caused by inaudible voice attacks, the paper proposes NormDetect, a software-based mitigation that can be instantly applied to a wide range of devices without requiring any hardware modification. Unlike the approaches devised by existing studies, which are supervised in nature, this paper adopts an unsupervised learning method to detect such attacks. It involves training the model on benign audio samples rather than on both kinds of samples.

2.3.1 System Overview

The overview of the system proposed by the paper is as shown in Figure 2.9.
The key stages of the system are:

1. Speech Preprocessing
2. Spectrum Augmentation
3. NormDetect Model.

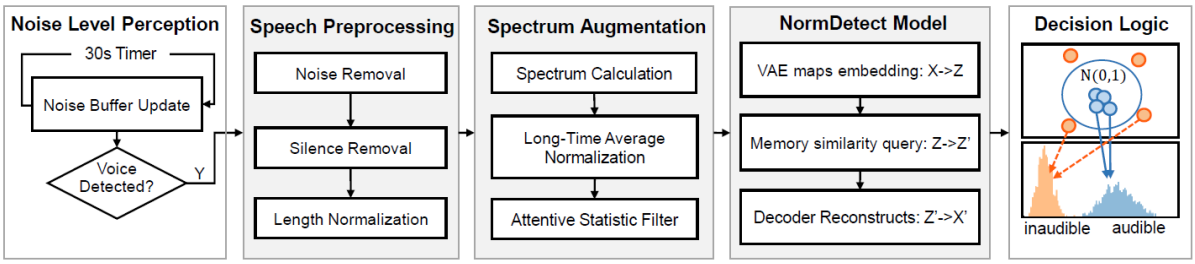


Figure 2.9: System Overview

Prior to the key stages, the system consists of a module called Noise Level Perception, which is a method that makes the system continuously monitor the ambient noise in its environment. The system maintains a buffer of the last five noise samples, which it continuously updates. Each time a new noise sample is recorded, it enters this queue,

replacing the oldest one. The system calculates the average noise level from these five samples, to obtain a more stable estimate of the overall ambient noise level. To account for changes in the environment, periodic updation of the noise buffer takes place using a timer. Once the system has a stable understanding of it's ambient noise, a distinct sound such as a voice command can be detected on comparing the detected sound with the calculated noise estimate value. If a voice is detected, the system moves to the next stage, speech preprocessing.

1. Speech Preprocessing

1. **Noise Removal :** Ambient noise is found to have an abnormal response similar to that of inaudible commands. Hence it is essential to remove ambient noise to isolate the response of inaudible commands.

2. **Silence Removal :** It is vital to eliminate the influence of different unvoiced segments, which are caused by speakers' habits, such as speaking speed, and semantic pauses, to ensure that the model can focus on learning voiced features.

3. **Length Normalization :** It is performed to ensure that all voice commands, regardless of their original duration, have a consistent input length when fed into the machine learning model.

2. Spectrum Augmentation

This module transforms normalized audio into augmented spectrums.

1. **Spectrum Calculation/Generation :** Using the Short-Time Fourier Transform(STFT), the audio signal is converted into it's frequency components over time, called a spectrogram. This is done for the purpose of feature extraction.

2. **Long-Time Average Normalization :** This method takes the average of the log of STFT spectrums to generate the Long-Time Average Spectrum (LTAS). This helps to minimize the variations due to different speakers and speech content, so that the system can focus more on the underlying patterns.

3. **Attentive Statistical Filter :** The Attentive Statistical Filter helps to reduce irrelevant linguistic variations to enhance the discriminative information between audible and inaudible voice commands using the F-ratio method.

3. NormDetect Model

The main principle behind the model is that the model reconstructs the normal data well based on standard training data, while it reconstructs poorly for abnormal data. The model is a Variational Autoencoder (VAE), to which the augmented spectrum is fed as input. The VAE then performs mapping, reconstruction, and judgement based on the anomaly score. The diagram of the proposed model consisting of 3 stages is shown in Figure 2.10.

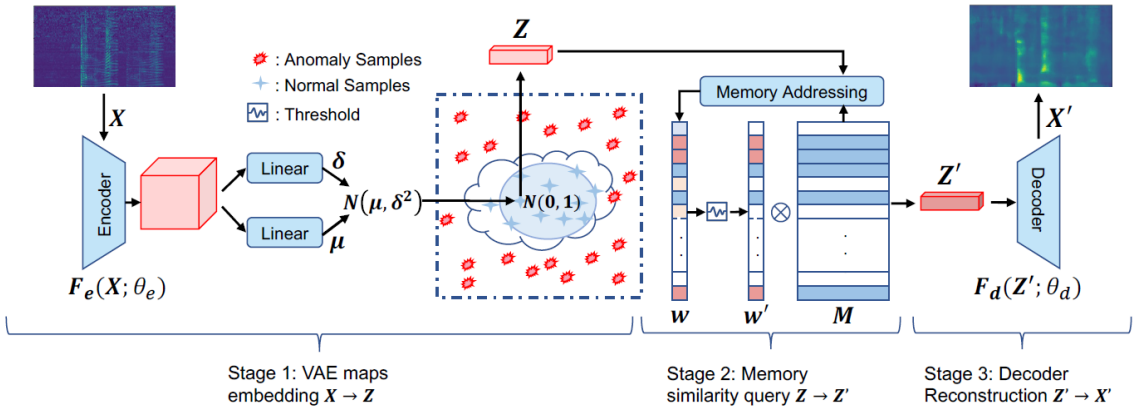


Figure 2.10: NormDetect Model

STAGE 1: VAE maps embedding $X \rightarrow Z$

In this stage, the encoder produces a latent embedding Z of the input spectrum X. The embedding Z follows a Gaussian distribution which captures the essential features of the input.

STAGE 2: Memory Similarity Query $Z \rightarrow Z'$

This stage involves a memory network structure, also called the memory module, that consists of multiple items M_i that represent prototypical patterns of normal audible voice commands. The mapped embedding Z is compared to each memory item M_i to compute a similarity score. This generates a similarity weights vector w, using which low-similarity items in w are removed, resulting in a refined weights vector w'. The refined weights vector w' is then used to compute Z' based on the most similar memory items, which reflects the closest match to the normal voice command patterns, helping to mitigate the

risk of false acceptance by voice command systems.

STAGE 3: Decoder Reconstruct $Z' \rightarrow X'$
The input to this stage is the embedding Z' using which the decoder attempts to reconstruct the input X'. An anomaly score is computed between X and X' by taking the negative log-likelihood.

A dataset containing 383320 samples was used to evaluate the NormDetect model, which included both audible and inaudible voice commands. The model performed an average AUC of 99.48% and EER of 2.23%, which implies that the model has a strong discriminatory power and can be adapted across various devices.

2.4 Hidden Voice Commands: Attacks and Defenses on the Voice-Controlled Systems of Autonomous Vehicles [4]

As autonomous vehicles increasingly rely on Artificial Intelligence for more advanced and reliable features, voice-controlled systems (VCS) have become critical for human- vehicle interaction. However, recent studies reveal that these systems are vulnerable to inaudible voice command or hidden voice command (HVC) attacks, where ultrasonic attack signals manipulate vehicle behavior without user knowledge. This paper explores the mechanisms of such attacks on autonomous vehicles, examining the potential risks and impacts on vehicle control. The paper also proposes a pop-noise detection based defense solution. This defense provides an essential layer of security. Hidden voice command attacks pose various potential dangers to vehicle operations and, by extension, public safety. Potential attack scenarios include:

1. **Media Manipulation:** An attacker embeds inaudible voice attacks into a popular video or song, that can be distributed through social media platforms. When a driver plays such media in the vehicle, the voice-controlled systems interpret the embedded commands, potentially causing dangerous consequences.

2. **Proximity-Based Attacks:** An adversary equipped with an ultrasound transmitter within close range of the target vehicle can issue commands directly to the VCS,

manipulating navigation, speed, or system settings.

The risks of such attacks highlight the urgent need for robust defenses, as they could lead to severe accidents, property damage, or loss of life. Existing defenses against hidden voice command attacks include several techniques, though many are specific to either inaudible commands or adversarial audio perturbations:

- **Audio Filtering and Sampling Rate Adjustment:** Some defenses attempt to mitigate adversarial audio by reducing the sampling rate, limiting the VCS's ability to capture hidden commands. While this method may hinder adversarial examples, it is ineffective against ultrasound attacks, as high-frequency sounds bypass these filters.

- **Source Verification:** One proposed method involves detecting inconsistencies in audio signals originating from synthetic sources versus live human speakers. However, this approach is still limited in practical application, as distinguishing between recorded and live audio remains challenging for Automatic Speech Recognition Systems (ASR).

Despite these strategies, the limitations of existing defenses point to the need for a comprehensive approach that can safeguard against various types of HVC attacks.

2.4.1 Pop Noise Detection as a Defense Strategy

This paper proposes a general defense method based on detecting "pop noise"—the distinctive low-frequency sound produced when a person speaks near a microphone. Human speech inherently generates pop noise through natural breathing, which is absent in synthetic or pre-recorded audio. By leveraging this pop noise, the proposed defense can reliably distinguish between genuine voice commands and hidden voice commands from synthetic sources. The defense method involves three main steps:

1. **Phoneme Segmentation and Non-Speech Component Removal:** The VCS processes the audio signal to identify and isolate phonemes, removing non-speech components that may interfere with accurate detection.

2. **Pop Noise Detection Using Short-Time Fourier Transform (STFT):** Each phoneme segment undergoes STFT to measure low-frequency energy, isolating frames that exhibit the unique frequency and duration characteristics of pop noise.

3. **Classification and Verification:** Once pop noise is identified, the VCS uses Gammatone Frequency Cepstral Coefficients (GFCC) to classify the audio source as either human or synthetic, applying a two-class support vector machine (SVM) model to increase detection accuracy.

4. **Experimental Validation :** Tests conducted on the proposed pop-noise-based defense show a high true acceptance rate (TAR) and low false acceptance rate (FAR), effectively identifying and rejecting hidden commands under both inaudible voice command and adversarial audio scenarios. Command length and replay device quality slightly affect detection rates, with longer commands yielding higher accuracy. This defense demonstrated resilience against attacks across various replay devices and command lengths, validating its robustness for practical use.

The study underscores the vulnerabilities of voice-controlled systems in autonomous vehicles to hidden voice command attacks, emphasizing the urgency for advanced defenses. Current defense strategies are often limited to specific types of attacks, making them inadequate for real-world applications. The proposed pop-noise-based detection method addresses these limitations, offering a general solution that leverages unique human speech characteristics to filter out synthetic audio commands. This defense approach represents a significant step toward safer autonomous driving technologies, potentially setting a new standard for VCS security in future vehicle designs.

2.5 Summary and Gaps Identified

2.5.1 Gaps Identified

The papers address the security vulnerabilities of voice-controlled systems (VCS) to inaudible voice attacks. The papers cover a variety of instances where such attacks are seen, the first study shows how an attacker can manipulate autonomous vehicle behavior through hidden voice commands and develops a pop-noise-based defense mechanism. Another paper puts forward NormDetect, a software-based solution that uses unsupervised

learning to detect such inaudible attacks in devices without any hardware changes. A third study extends the attack range of inaudible commands from 5 ft. to 25 ft. and builds a software defense that utilises an elliptical classifier. Finally the last paper identifies vulnerabilities in VCS where ultrasonic commands are used to perform malicious actions, and suggests both hardware and software solutions to address these risks. The gaps identified from these studies are:

1. **Range Limitations:** By using a different transmitter design, the attack range has been increased to 25 ft. However, further research is required to adapt to longer distances or noisy environments.

2. **Compatibility:** The effectiveness of software defenses across multiple devices remains unclear, even though they can be built without any large hardware modifications.

3. **Unsupervised Learning Challenges:** Any unsupervised detection model requires optimization to reduce false positives and negatives and ensure scalability for different devices.

4. **Nonlinear Feature Detection:** All classifiers built need to be trained on a variety of different speakers, languages, and environments to ensure that the features of malicious signals can be captured and tested in different scenarios.

5. **Security:**To integrate both hardware and software solutions for the detection of inaudible voice attacks, development of a new security framework is needed.

6. **Testing:** To assess the effectiveness and robustness of developed solutions, extensive real-world testing and evaluation is needed.

2.5.2 Summary of Literature Survey

Paper Title	Advantages	Disadvantages	Methodology
DolphinAttack: Inaudible Voice Commands [1]	• Implements software and hardware defense • Hardware independent	• Limited by distance of 5ft • Attack sample dependent	• Support Vector Machine (SVM) • Hardware
Inaudible Voice Commands: The Long-Range Attack and Defense [2]	• Extends attack range up to 25ft • Accurate classification for malicious and benign audio • Hardware independent	• Additional hardware needed to increase range • Attack sample dependent	Harmonics
Learning Normality is Enough: A Software-based Mitigation against Inaudible Voice Attacks [3]	• Universal • Lightweight • Hardware independent • Attack sample independent	• Ineffective for attack range of only 30 cm	Convolutional Variational AutoEncoder (Conv VAE) with a memory network module

Table 2.1: Summary of literature survey on inaudible voice attacks

Paper Title	Advantages	Disadvantages	Methodology
Hidden Voice Commands: Attacks and Defenses on the VCS of Autonomous Driving Cars [4]	• Can defend against different types of attacks • Novel defense strategy	• Dependence on Pop Noise Detection • Physical proximity required for some attacks	Noise produced by human speech (breathing)

Table 2.2: Summary of literature survey on inaudible voice attacks cont.

2.5.3 Strategies for Bridging Identified Gaps

Voice-controlled systems are becoming highly susceptible to ultrasonic attacks, resulting in risks to user privacy and system integrity. Despite the development of technological defenses, critical gaps remain in these defenses. Addressing these gaps and bridging them is key to building defenses that are more optimised.

1. Improve ultrasonic detection through advanced sensors and analysis to improve range and effectiveness of the defense in several different environments.

2. Build software-based defenses that are adaptable across different devices by testing on a larger set of devices and hardware components.

3. Integrating supervised and unsupervised learning, optimizing models, and testing in real-world scenarios to increase the accuracy of detection.

4. Improve software models to detect ultrasonic features in noisy environments and use noise-augmented data in training.

5. Create a defense architecture that follows the industry standards found in VCS and incorporates both software and hardware defenses.

6. Test proposed solutions in varied real-world settings to check its effectiveness.

2.5.4 Proposed Solution

The rising use of voice-controlled systems has increased the number of targets for attacks, allowing attackers to gain control over these systems without the user's awareness, thus leading to serious security breaches, such as unauthorized access to sensitive information and financial manipulation. These reasons have served as a motivation to dig deeper into this area to develop countermeasures to such emerging threats. So, the solution proposed is to create a software detection system that will classify audible (benign) and inaudible (malicious) audio samples, generate and send alerts to users when malicious commands pass through, and decode those commands. The mentioned gaps are resolved by developing this solution.

Chapter 3

Requirements

This chapter states the hardware and software resources required to develop the project.

3.1 Hardware Requirements

1. **Intel Core i7/i9 CPU:** provide the computational power necessary for real-time processing of audio signals, generation of ultrasonic commands, and running simulations of attack scenarios and their effects on devices.

2. **NVIDIA GeForce RTX GPU:** essential for handling parallel processing tasks. It is used for accelerating the analysis of high-frequency ultrasonic signals for machine learning purposes and in real-time analysis.

3. **16GB/32GB RAM** : for dealing with large datasets or continuous real-time processing, ensuring smooth execution of complex programs for audio generation, analysis, and attack testing, and the running of multiple tools or scripts simultaneously without any performance drops.

4. **Windows 11, 64-bit OS:** provides better compatibility with current tools and libraries required for the project, and its 64-bit architecture allows efficient handling of large audio data files and supports high-performance applications.

3.2 Software Requirements

1. **Python (Version 3.9 or Higher):** used for this project due to its versatility, ease of use, and extensive library ecosystem. Version 3.9 or higher is preferred because of its compatibility with the latest libraries and tools and improved features like flexible function annotations and enhanced dictionary operations, useful in large-scale data processing and simulations.

2. **Development Environment: Google Colab and Jupyter Notebook:** Google Colab provides free GPU access, essential for computationally intensive tasks like real-time ultrasonic signal processing or machine learning tasks. Jupyter Notebook is an open-source interactive environment for running Python code that is ideal for prototyping and visualizing data, such as analyzing audio waveforms or plotting spectrograms.

3. **Python Libraries:** used for audio and signal processing, machine learning, visualization, data handling, and analysis purposes. Some of them include Librosa, Matplotlib, NumPy, etc.

The requirements stated in this chapter were used to develop the project and would be necessary in order to run the developed code in an other environment.

Chapter 4

System Architecture

4.1 System Overview

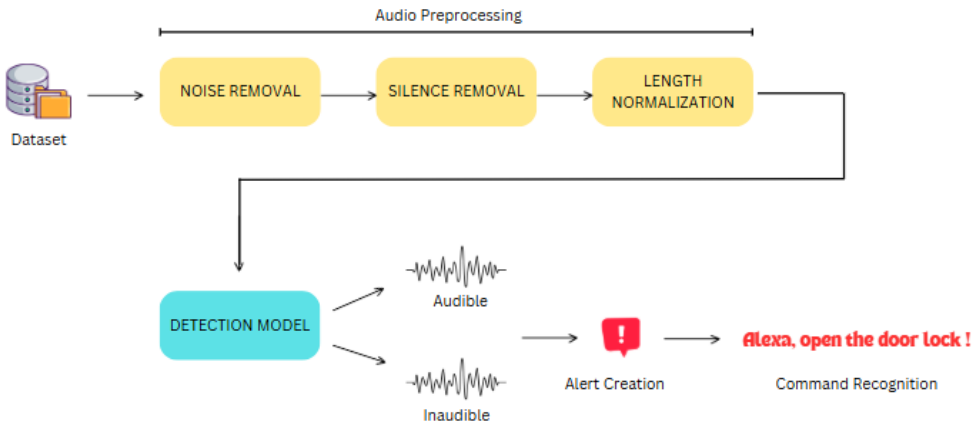


Figure 4.1: System Architecture

The entire system of detection begins with an input from the microphone. The audio input passes through several preprocessing steps which are: Noise Removal, Silence Removal, Length Normalization. Following which, it is passed to a detection model which classifies the input into one of the two output classes - audible(benign) or inaudible(malicious). If the classification result obtained is audible, the audio input passes freely through the system without hindering the working of the Voice-Controlled System and enabling it to execute the actions as directed by the command in the input received. Whereas, if the classification result is determined as inaudible(ie. an input in the ultrasonic frequency range), the original input gets demodulated in to the audible range and passes through a low pass filter which filters out the high frequency components as a result of which only the audible component of the input remains. An alert is then generated to indicate the presence of an inaudible command and the audible component passes through a speech

recognition system to identify the message in the audio and is displayed to the user along with the alert.

4.2 Architectural Design

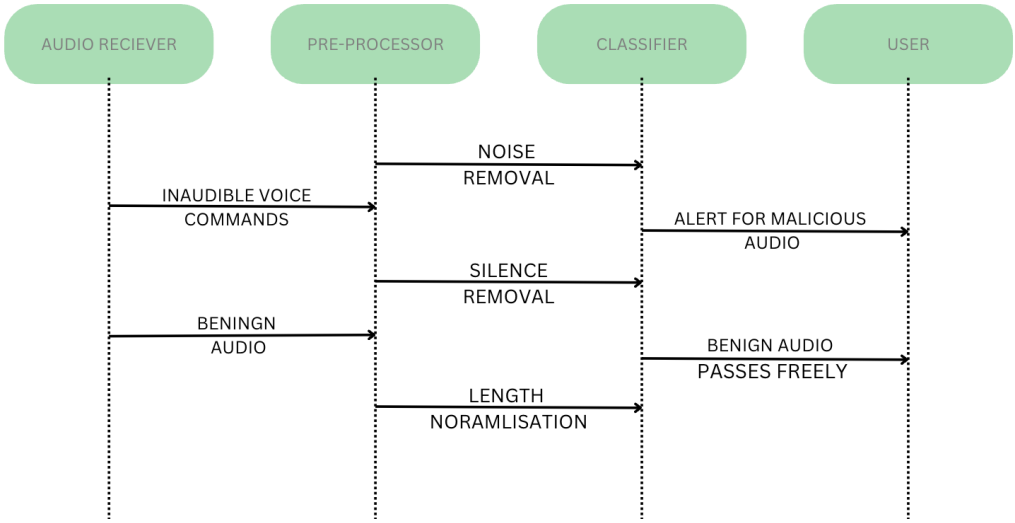


Figure 4.2: Data Flow Diagram

The simulation of the flow of an audio within a microphone is simulated using Python script in Jupyter Notebook. The different layers of the system being developed are:

1. **Input Layer**

   This layer receives the input audio which has to be classified into one of the two target classes: benign and malicious.

2. **Processing Layer**

   The audio preprocessing of the input audio is performed in this layer. Following audio preprocessing, feature enhancement is performed on the input, which is includes Z-score normalization. The audio preprocessing consists of:

   (a) Noise Removal

   (b) Silence Removal

   (c) Length Normalization

3. **Model Layer**

   The preprocessed result is fed as input to this layer, where the model attempts to predict the class which the input belongs to. The layer consists of a detection model which uses the features learned during training to perform prediction.

4. **Output Layer**

   The output layer consists of two components: alert generation and command recognition. This layer produces output only if the predicted class is malicious, whereas benign audio passes freely and does not enter this layer.

   - **Demodulation & Low Pass Filter:** The input is demodulated into it's baseband signal containing the audible command and the carrier wave of high frequency. The low pass filter eliminates the high frequency carrier and only the audible baseband component remains which is passed to the subsequent modules.
   - **Alert Generation:** An alert is generated to indicate to the users that the smart device is under attack as it received an ultrasound-modulated audio as input, which can make the user more alert by controlling their voice assistant to receive only genuine commands.
   - **Command Recognition:** Along with the alert, the command embedded in the ultrasound-modulated audio input(ie. the audible baseband obtained after demodulation) is also displayed, so as to let the user know what the attacker is trying to access or get control over, which will then allow users to protect that asset.

4.3 Project Timeline

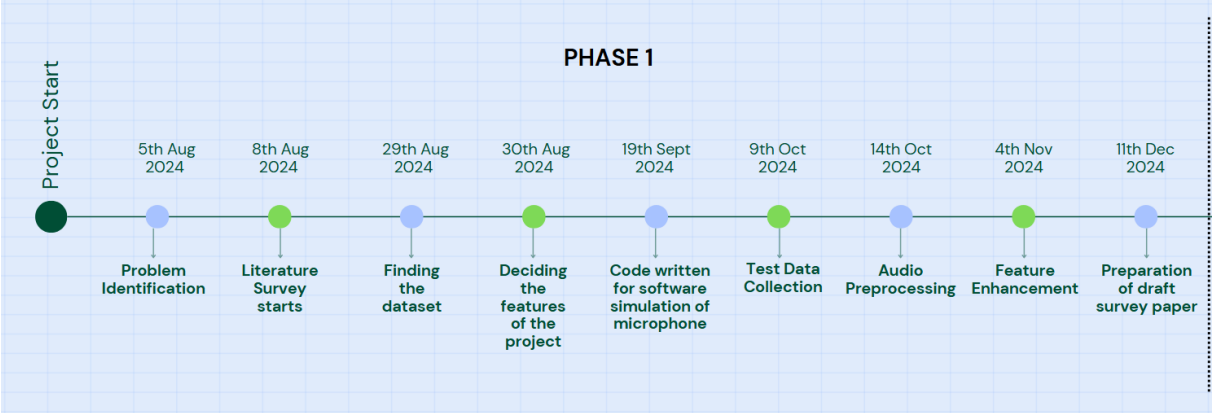


Figure 4.3: Gantt Chart - Project Phase 1

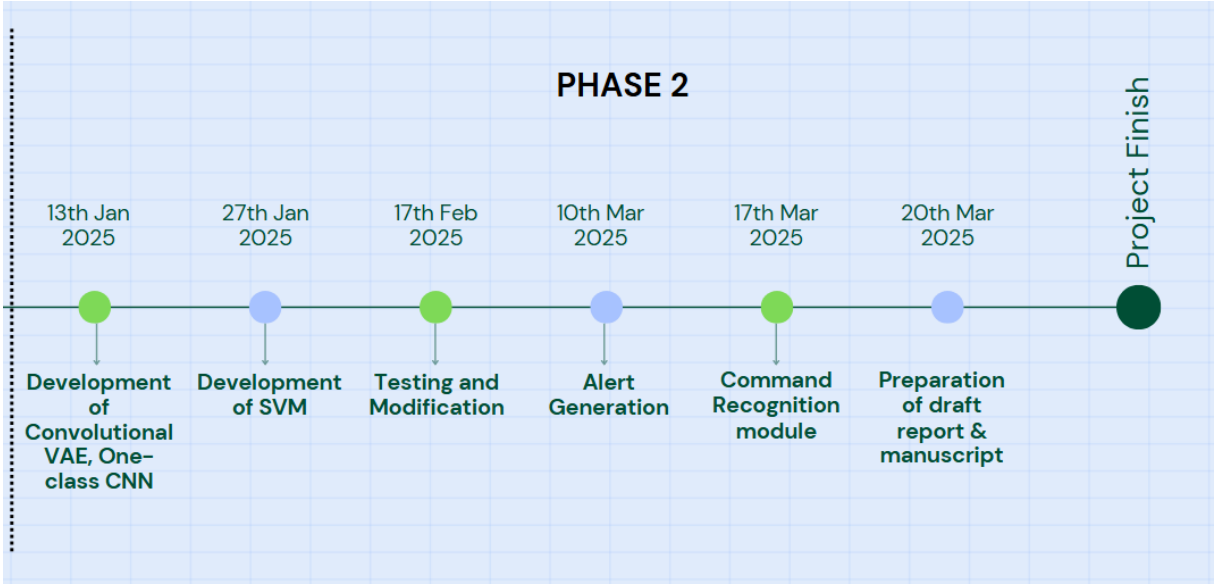


Figure 4.4: Gantt Chart - Project Phase 2

Chapter 5

System Implementation

This chapter describes in detail the methodology used in this project and how it is implemented.

5.1 Proposed Solution

Voice-controlled systems, such as smart assistants and smartphones, have become increasingly common in our daily lives. However, they are vulnerable to a class of inaudible voice command attack known as DolphinAttack, which exploits ultrasonic frequencies to send malicious commands that are inaudible to humans but can be interpreted by microphones because of the microphones' nonlinearity. This poses serious privacy and security risks, as unauthorized actions can be triggered without the user's awareness.

So, the solution proposed to such emerging threats is to create a software detection system that will classify audible (benign) and inaudible (malicious) audio samples, generate and send alerts to users when malicious commands pass through, and decode those commands.

5.2 Proposed Methodology

The goal is to develop an effective detection method which will classify benign and malicious voice commands, alert the users when those malicious ones are found, and decode the corresponding commands. This involves analyzing the characteristics of both audible and inaudible audio inputs using machine learning techniques. Accurate detection of this attack is important to get a safe and reliable voice-controlled system.

For the detection method, three machine learning models are used:

1. Convolutional Variational AutoEncoder (CVAE)

2. One-Class Convolutional Neural Networks (CNN)

3. Support Vector Machines (SVM)

This is done to compare and understand the working of each model with the goal of identifying the most suitable one for alert generation and command recognition.

Audio preprocessing is the most important and common method that is used in all three models, as the raw audio samples are transformed into cleaner, standardized, and consistent versions by reducing noise, removing silence, and normalizing the audio samples into equal lengths.

Depending on the inputs used, there are some preprocessing steps that are specific to each model after the audio preprocessing. CVAE and One-Class CNN have the same preprocessing steps since the inputs to both of them are spectrogram images of the audio, whereas the SVM has a different set of steps since the input to the model is an audio.

5.2.1 CVAE & One-Class CNN

1. **Audio Slicing:** Slices the original audio of 1.5sec duration to consider only 0.5sec of the audio.

2. **Spectrogram Generation:** To obtain a visual representation corresponding to the 0.5sec of the audio, as desired by the model

3. **Resizing:** To further reduce the dimensions of the spectrogram input

4. **Z-Score Normalization:** To standardize the spectrogram preventing certain features from dominating the learning process of the model and ensuring that different features contribute equally to the model's performance.

5.2.2 SVM

1. **Framing:** To divide the audio signals into small and manageable time frames for localized analysis of signal properties.

2. **Windowing:** Applies a window function to each frame to minimize spectral leakage before transformation.

3. **Feature Extraction:** To convert the windowed audio frames to representative numerical features suitable for classification.

4. **Feature Selection:** To identify and retain the most relevant features that contribute to the classification of audible and inaudible classes, improving model performance.

5. **Feature Scaling:** To normalize or standardize the selected feature values to ensure uniform input ranges for the training.

The subsequent processes include training, testing, and validation of each of the models, alert generation, and command recognition.

5.2.3 Algorithms Used

The following are the algorithms used in the implementation:

Audio Preprocessing

STEP 1: Start

STEP 2: Import Python modules librosa, noisereduce as nr, os, numpy as np, and soundfile as sf

STEP 3: Define the path for the input as `input_folder_path` and the path for the output files as `output_folder_path`.

STEP 4: Create output directory if not present using the makedirs() method of the `os` module

STEP 5: Set `target_duration` as 1.5

STEP 6: Repeat steps 6.1 to 6.4 for each folder in `input_folder_path`

STEP 6.1: Define `input_path` by joining the folder to the `input_folder_path`

STEP 6.2: Define `output_path` by joining the folder to the `output_folder_path`

STEP 6.3: Create output directory if not present

STEP 6.4: Repeat steps 6.4.1 to 6.4.10 for each file in `input_path`

STEP 6.4.1: Define the `input_file_path` by joining the filename to the `input_path`

STEP 6.4.2: Load the input audio file from `input_file_path` into audio

STEP 6.4.3: Store the noise reduced audio into `reduced_noise_audio` by passing audio to reduce_noise() method of `nr` module

STEP 6.4.4: Use librosa.effects.split() to remove the silent parts by passing `reduced_noise_audio` as input

STEP 6.4.5: Concatenate the non-silent intervals into `non_silent_audio`

STEP 6.4.6: Compute the `target_length` as the product of `target_duration` and sampling rate

STEP 6.4.7: If the length of the `non_silent_audio` is greater than `target_length`, then

STEP 6.4.7.1: Truncate the audio to the first `target_length` samples and store it in `normalized_audio`

STEP 6.4.8: Otherwise

STEP 6.4.8.1: Compute the amount of padding as difference of length of `non_silent_audio` from `target_length`

STEP 6.4.9: Define `output_file_path` by joining filename to `output_path`

STEP 6.4.10: Save the `normalized_audio` to `output_file_path`

STEP 7: Stop

Spectrogram Generation

STEP 1: Start

STEP 2: Import Python modules librosa, noisereduce as nr, os, numpy as np, soundfile as sf, matplotlib.pyplot as plt

STEP 3: Define the path for the input as `input_folder_path` and the path for the output files as `output_folder_path`.

STEP 4: Create output directory if not present using the makedirs() method of the `os` module

STEP 5: Set `target_duration` as 1.5

STEP 6: Repeat steps 6.1 to 6.4 for each folder in `input_folder_path`

STEP 6.1: Define `input_path` by joining the folder to the `input_folder_path`

STEP 6.2: Define `output_path` by joining the folder to the `output_folder_path`

STEP 6.3: Create output directory if not present

STEP 6.4: Repeat steps 6.4.1 to 6.4.10 for each file in `input_path`

STEP 6.4.1: Define the `input_file_path` by joining the filename to the `input_path`

STEP 6.4.2: Load the input audio file from `input_file_path` into audio

STEP 6.4.3: Set `window_size` to 1024, `hop_length` to 512 and `window` as "Hann"

STEP 6.4.4: Generate the Short-Time Fourier Transform (STFT) using librosa.stft() by passing `audio`, `window_size` and `hop_length` as arguments

STEP 6.4.5: Generate the spectrogram by taking the magnitude of the STFT and storing it into `spectrogram`

STEP 6.4.6: Apply color map to the `filtered_normalized_spectrogram` to get `colored_spectrogram`

STEP 6.4.7: Define `output_file_path` by joining filename to `output_path`

STEP 6.4.8: Save the `colored_spectrogram` to `output_file_path`

STEP 7: Stop

5.2.4 Framing

STEP 1: Start

STEP 2: Define the `frame_signal` function

STEP 2.1: Take `y`, `sr`, `frame_duration`, and `overlap` as the input parameters where: y → audio signal (numpy array), sr → Sampling rate of the signal, frame_duration → Frame length in seconds (default = 25ms), overlap → Overlap percentage (default = 50%).

STEP 2.2: Convert frame duration to sample size: `frame_size` = `sr` × `frame_duration` (convert seconds to samples).

STEP 2.3: Compute hop size: `hop_size` = `frame_size` × (1 `overlap`).

STEP 2.4: Use librosa.util.frame() to segment the signal into overlapping frames.

STEP 2.5: Return the framed 2D numpy array.

STEP 3: Define the `process_audio_folder` function

STEP 3.1: Take `folder_path`, `target_sr`, and `output_folder` as the input parameters where: folder_path → Directory containing .wav audio files, target_sr → Target sampling rate, output_folder → Directory where framed .npy files will be stored.

STEP 3.2: Create `output_folder` if it does not exist.

STEP 3.3: Retrieve all .wav files from folder_path.

STEP 3.4: For each .wav file in folder_path:

STEP 3.4.1: Load the audio file using librosa.load(file_path, sr=target_sr).

STEP 3.4.2 Apply frame_signal(y, sr) to frame the signal.

STEP 3.4.3 Save the framed signal as a .npy file in output_folder.

STEP 3.4.4: Print the filename and the shape of the framed data.

STEP 4: Define folder paths

STEP 4.1: Set `audible_folder` for storing 16kHz audio files.

STEP 4.2: Set `inaudible_folder` for storing 96kHz audio files.

STEP 4.3: Set the `audible_output` folder for framed audible signals.

STEP 4.4: Set the `inaudible_output` folder for framed inaudible signals.

STEP 5: Process Audio Files

STEP 5.1: Call `process_audio_folder`(audible_folder, 16000, audible_output).

STEP 5.2: Call `process_audio_folder`(inaudible_folder, 96000, inaudible_output).

STEP 6: Stop

5.2.5 Windowing

STEP 1: Start

STEP 2: Define Input and Output Paths

STEP 2.1: Set `audible_input_path` as the folder containing framed audible signals.

STEP 2.2: Set `inaudible_input_path` as the folder containing framed inaudible signals.

STEP 2.3: Set `audible_output_path` as the folder to save windowed audible signals.

STEP 2.4: Set `inaudible_output_path` as the folder to save windowed inaudible signals.

STEP 3: Create Output Directories

STEP 3.1: Ensure `audible_output_path` exists (create if not).

STEP 3.2: Ensure `inaudible_output_path` exists (create if not).

STEP 4: Define the `apply_hamming_window` Function

STEP 4.1: Take `input_folder` and `output_folder` as parameters.

STEP 4.2: Retrieve all .npy files from `input_folder`.

STEP 4.3: Define `batch_size` = 1000 (to process large datasets in chunks).

STEP 4.4: Iterate through files in batches:

STEP 4.4.1: Extract `batch_files` from `file_list`.

STEP 4.4.2: For each `file_name` in `batch_files`:

STEP 4.4.2.1: Construct `input_path` and `output_path`.

STEP 4.4.2.2: Load .npy file using np.load() with `mmap_mode`='r' for memory efficiency.

STEP 4.4.2.3: Generate a Hamming window of size equal to the number of columns in `framed_data`.

STEP 4.4.2.4: Apply Hamming window: `windowed_data` = `framed_data` * `window`.

STEP 4.4.2.5: Save `windowed_data` to `output_folder`.

STEP 5: Apply Hamming Windowing

STEP 5.1: Call `apply_hamming_window`(audible_input_path, audible_output_path).

STEP 5.2: Call `apply_hamming_window`(inaudible_input_path, inaudible_output_path).

STEP 6: Stop

5.2.6 Feature Extraction

STEP 1: Start

STEP 2: Define the `extract_features` function

- STEP 2.1: Take `file_path` and `sr` (sampling rate) as inputs.
- STEP 2.2: Load .npy file (framed & windowed audio).
- STEP 2.3: Convert the 2D framed signal to 1D if necessary.
- STEP 2.4: Compute spectral features using librosa: MFCC (n_mfcc=13), Spectral Centroid, Spectral Bandwidth, Spectral Contrast, Spectral Flatness, Spectral Rolloff.
- STEP 2.5: Compute temporal features: Zero Crossing Rate, Energy (mean(y^2)), Entropy ($-\sum(y^2 \log(y^2))$)
- STEP 2.6: Compute statistical features (mean, variance, skewness, and kurtosis).
- STEP 2.7: Return all extracted features as a NumPy array.

STEP 3: Define the `process_dataset` function

- STEP 3.1: Take folder, label, and sr as inputs.
- STEP 3.2: Initialize empty lists for data and labels.
- STEP 3.3: Iterate over all .npy files in the folder:
  - STEP 3.3.1: Load each file and extract features using `extract_features()`.
  - STEP 3.3.2: Append extracted features to data.
  - STEP 3.3.3: Append label to labels.
- STEP 3.4: Convert data and labels into numpy arrays and return them.

STEP 4: Set directories for audio data

- STEP 4.1: Define `audible_dir` (16 kHz audio).
- STEP 4.2: Define `inaudible_dir` (96 kHz audio).

STEP 5: Extract features from both classes

- STEP 5.1: Call `process_dataset`(audible_dir, label=1, sr=16000).

STEP 5.2: Call `process_dataset`(inaudible_dir, label=0, sr=96000).

STEP 6: Combine Datasets

STEP 6.1: Stack feature matrices from both classes into X.

STEP 6.2: Concatenate labels into y.

STEP 7: Apply Feature Selection Using SVM-Based Recursive Feature Elimination (RFE)

STEP 7.1: Define an SVC model with a linear kernel.

STEP 7.2: Apply RFE with `n_features_to_select`=30.

STEP 7.3: Fit RFE on X and y to select the best 30 features.

STEP 7.4: Store selected feature indices.

STEP 8: Normalize Selected Features

STEP 8.1: Standardize features using StandardScaler().

STEP 8.2: Fit and transform `x_selected`.

STEP 8.3: Save the scaler.pkl model using joblib.dump().

STEP 9: Save Extracted & Selected Features

STEP 9.1: Save `x_scaled` and `y` as an .npz file.

STEP 10: Stop

5.2.7 Alert Generation

STEP 1: Start

STEP 2: Import Python modules librosa, noisereduce as nr, os, numpy as np, soundfile as sf, matplotlib.pyplot as plt

STEP 3: Calculate the error rate for all three models and select a suitable one

STEP 4: Once a model has been selected, determine a suitable threshold value that factors in the error rate of the model

STEP 5: If, on detecting a voice signal, the error rate lies above the threshold value, then the signal is classified as malicious and an alert is generated for the user.

STEP 6: The signal is passed through a speech transcription tool that will then recognize the command embedded in the signal; this command is also provided to the user.

STEP 7: Stop

5.3 Dataset Identified

The dataset used for the project is the "Fluent Speech Commands Dataset"[5], which is an open-source audio dataset used for spoken language understanding experiments. The audio is recorded as 16 kHz single-channel .wav files each containing a single utterance for controlling smart-home appliances or virtual assistants, with a total of 30,043 utterances from 97 speakers. Three slots: action, object and location, have been used as labels for each audio. The combination of the slot values is referred to as the intent of the utterance. This dataset contains 248 phrases mapped to 31 unique intents. The dataset is divided into 3 parts: 23,132 utterances for training, 3,118 utterances for validation and 3,793 utterances for testing, such that no speaker appears more than once in a split. The dataset can be found at https://fluent.ai/fluent-speech-commands-a-dataset-for-spoken-language-understanding-research/.

5.4 Description of Implementation Strategies

5.4.1 Detection Models

CONVOLUTIONAL VARIATIONAL AUTOENCODER

The VAE is a generative model designed for anomaly detection where it is trained only on normal data. It learns compact latent representations of the audio patterns during training. The VAE reconstructs the input test data and measures the performance using reconstruction error. But when ultrasonic commands (the anomaly) are given as input, a higher reconstruction error is obtained, indicating an attack. This model is particularly effective for uncovering subtle deviations that conventional models such as K-Nearest Neighbours (KNN) and Isolation Forests may miss, making it a robust choice for de-

tecting covert commands in complex audio environments. When convolutional layers are employed in both the encoder and decoder, the VAE is called a Convolutional Variational AutoEncoder (CVAE), which is the implementation used for the project.

Preliminaries

1. **Latent Space Representation**

   The VAE consists of 2 parts - encoder and decoder. The encoder encodes the given input to a lower dimensional representation called the latent space representation. The latent space is a vector that captures the most essential features extracted from the input data. The decoder uses this latent space to reconstruct the input the data. The latent space is generated such that it contains only the important features which are sufficient enough for the decoder to reconstruct the input using it. The dimension of the latent space is a factor that influences the model's performance, capacity, and generative abilities. A higher dimension enables the model to capture more complex variations whereas a lower dimension causes the model to learn a more generalized representation.

2. **Reparameterization Trick**

   The reconstruction of the input is done by the decoder by sampling a point from the latent space. The latent space is a probability distribution and sampling from such a distribution is non-differentiable, meaning computing gradients for backpropagation becomes difficult. To overcome this, the reparameterization trick is used, hence making it possible to train the VAE based on gradient based optimization algorithms and also enabling the model to simulate various normal audio being disturbed with noise making the modeling of normal audio patterns more robust.

3. **Loss Function**

   To guide the model during training, a loss function is used to quantify how well the model's predictions match the actual data. The loss function for a VAE is a combination of the reconstruction error and the KL Divergence loss. The reconstruction error measures how well the VAE reconstructs the original input from the latent space. the KL divergence loss quantifies how much the learned latent space distribution deviates from the desired Gaussian distribution. During training, this

loss pushes the encoder to adjust its parameters so that the output distribution approximates the target Gaussian distribution.

Training

This model was trained completely on audible audio samples.After the audio preprocessing steps, the input audio in the training dataset are converted to corresponding spectrograms using a Hanning window function with a window size of 1024. The Hanning window helps reduce spectral leakage and ensures clarity and precision of the frequency distribution in the spectrogram. Due to limitations of computational resources, before the spectrogram is generated, the audio is sliced to contain only 0.5 seconds of the total duration. These 0.5 seconds are taken from the central portion of the audio recording. Even though the length of the audio is reduced and a part of the command of the audio is lost, the training process in a negative way since the model's focus is on learning the features that representative of the audible class, which is later used to detect a deviation leading to detection of inaudible class. The spectrogram obtained for 0.5 seconds of a single audio is of dimensions 496px x 369px (width x height). To further reduce computational load, the size was reduced to 128px x 95px. Following resizing, Z-Score normalization is performed to ensure that the spectrograms are standardized, which maintains numerical stability in computations, reduces large variations in features and helps reduce any bias that the model may develop. Z-Score normalization rescales the features to a standard normal distribution with a mean of 0 and a standard deviation of 1. The model was trained on 14208 audible samples.

Model Layers

Both the encoder and decoder are designed using convolutional layers. A spectrogram represents the frequency content of an audio over time, and convolutional layers are designed to learn local patterns efficiently. These layers are also translation-invariant, meaning, regardless of the position in the spectrogram, it can recognize patterns. The encoder involves 3 consecutive convolutional layers with a stride of 2, which allows for reduction in spatial dimensions resulting in the generation of a lower dimensional latent space, following which, is the flatten layer that converts the 3D features maps generated by the convolutional layers to a 1D vector hence preparing data for the dense layer.

The decoder starts with the dense layer that accepts the 1D vector which is then reshaped back to a 3D map. It uses 3 consecutive transposed convolutional layers that enables reconstruction of a spectrogram from a lower dimension to a spectrogram of original dimensions. The individual layers of the encoder and decoder are as shown in Figure 5.1 and Figure 5.2 respectively:

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 48, 64, 32)	320
conv2d_1 (Conv2D)	(None, 24, 32, 64)	18,496
conv2d_2 (Conv2D)	(None, 12, 16, 128)	73,856
flatten (Flatten)	(None, 24576)	0
dense (Dense)	(None, 256)	6,291,712
dense_1 (Dense)	(None, 32)	8,224

Figure 5.1: Encoder Layer

Layer (type)	Output Shape	Param #
dense_2 (Dense)	(None, 24576)	417,792
reshape (Reshape)	(None, 12, 16, 128)	0
conv2d_transpose (Conv2DTranspose)	(None, 24, 32, 64)	73,792
conv2d_transpose_1 (Conv2DTranspose)	(None, 48, 64, 32)	18,464
conv2d_transpose_2 (Conv2DTranspose)	(None, 96, 128, 1)	289

Figure 5.2: Decoder Layer

Loss Function

The loss function used for the model consists of 2 components - Reconstruction loss and KL Divergence loss. The decoder aims to minimize the reconstruction loss as much as possible which indicates how much the reconstructed data differs from the original input data. The KL Divergence loss measures how much the learned latent representation differs from a standard Gaussian distribution with a mean of 0 and a standard deviation of 1. This component of the loss function ensures that the data in the learned latent representation does not significantly deviate due to noise or other reasons, thus constraining the latent

representation to a desired distribution. This helps the model to generate samples that are as close as possible to the real data that the model would be dealing with.

ONE-CLASS CLASSIFICATION CNN

The One-Class Convolutional Neural Network (CNN) is a classification model used to detect anomalies. In the context of ultrasonic audio clasification, it is trained exclusively on normal audio spectrograms to extract features and learn their distributions and significance. At the time of inference, the model assigns an anomaly score to each sample based on its similarity to the training data. If an input produces a score above a defined threshold, it is considered anomalous—such as in the case of ultrasonic commands. This approach is particularly effective for identifying covert signals in noisy or complex audio environments, especially when only one class of training data is available.

Preliminaries

1. **One-Class Classification CNN**

   A One-Class Classification Convolutional Neural Network (CNN) is a neural network model designed for anomaly detection tasks. Only data from a single class, usually normal samples are used during training. The model learns different features of these inputs through its layers. At the time of inference, the model evaluates an input by comparing how closely it matches the features learned from the normal data. If the input deviates significantly, it is tagged as an anomaly. One-Class CNN is most effective when collecting data for different classes is difficult.

2. **Mean Square Error - Loss Function**

   Mean Squared Error (MSE) is a loss function that calculates how closely an input sample matches the labelled output. It is calculated by taking the average of the differences between the predicted value and the corresponding true value and squaring it. This identifies larger errors and penalises them more than smaller errors, allowing the model to make better predictions. In the context of a one-class CNN, MSE helps the model learn to predict values close to the desired label as normal data, while higher deviations indicate anomalies.

3. **Anomaly Score Thresholding**

Anomaly score thresholding is used to determine whether an input is normal or anomalous based on an anomaly score and its difference from a predetermined threshold. After training a model on normal data, each input is assigned an anomaly score during testing. This indicates how much it deviates from the features that the model has learned. A threshold is then used to understand this score: if the anomaly score exceeds the threshold, the input is classified as anomalous. The threshold is often fine-tuned using validation data.

Training

The model was trained on four batches of normal spectrograms, each containing 1000 to 1200 samples of audible audio. During the preprocessing, each audio clip was trimmed to retain only the central 0.5 seconds, to capture relevant acoustic features and reduce computational load. Spectrograms were then generated sized 496px $\times$ 369px, these were then resized to 128px $\times$ 95px to reduce computational overhead.

Before feeding them into the model, the spectrograms were standardized using Z-Score normalization, ensuring each feature contributes equally during training and enhancing numerical stability. The model was trained to minimize the Mean Squared Error (MSE) between the predicted and expected outputs for the audible class. A threshold anomaly score is determined through empirical testing. Inputs resulting in a score above this threshold, such as ultrasonic commands, are flagged as anomalies.

Model Layers

The model has an input layer that accepts spectrograms of shape (128, 95, 1). This is followed by three 2D convolutional layers. Each convolutional layer uses a (3$\times$3) kernel with the activation function as ReLU activation. These layers help in extracting features from the spectrograms. After each convolutional layer, batch normalization is applied to stabilize training. MaxPooling layers are used to downsample the data, reducing their dimensions while retaining information. Dropout layers randomly deactivate a portion of neurons during training to help in reducing overfitting.

Following these, a global average pooling layer is used to change the data into a single vector as the input to the dense layer. The fully connected dense layer then transforms and refines the extracted features. Finally, a dense output layer with a sigmoid activation function produces an anomaly score, which is used to classify the sample.

Loss Function

The model uses the Mean Squared Error (MSE) loss function, which calculates the squared difference between the model prediction and the data label. During training it calculates the anomaly score of each input and an appropriate threshold is determined. At inference, new inputs that have a score higher than the threshold are classified as anomalous.

The model effectively detects anomalous audio signals by learning the patterns of normal spectrograms and flagging deviations based on a fine-tuned anomaly threshold.

SUPPORT VECTOR MACHINE (SVM)

Support Vector Machine (SVM) is a supervised machine learning algorithm that is used primarily for classification tasks. It is simple and computationally efficient. It uses features of the input data to find the optimal hyperplane or boundary that distinguishes between audible (benign) and inaudible (malicious) audio samples in high-dimensional feature space. The main goal of the SVM is to maximize the margin between the hyperplane and the closest points, or support vectors, from each data class.

Preliminaries

1. **Decision boundary**

   It is a decision surface that separates audible and inaudible classes in the feature space. In SVM, it can be a line, a plane, or a hyperplane. A line is chosen for 2D, a plane for 3D, and a hyperplane for higher dimensions.

2. **Support Vectors**

   They are the data points that lie closest to the decision boundary. These are the most critical elements of the training set because they directly influence the position

and orientation of the hyperplane. Removing a support vector can change the model significantly.

3. **Margin**

   The margin is the distance between the decision boundary (hyperplane) and the support vectors from each of the audible and inaudible classes. A good SVM model identifies the hyperplane that maximizes this margin, which improves generalization and reduces the risk of overfitting.

4. **Linear Separability**

   Data is said to be linearly separable if it can be perfectly divided into audible and inaudible classes using a hyperplane.

5. **Linear Kernel**

   A kernel is a mathematical function that determines how the algorithm measures similarity between data points in a higher-dimensional space. It helps to handle both linear and non-linear classification problems, making the model computationally efficient in finding the hyperplane for complex datasets. A linear kernel is the simplest kernel function. It is used when the data is linearly separable and no transformation into higher dimensions is needed. It computes the dot product between two feature vectors. But if the classes cannot be separated linearly, then SVM uses kernel (non-linear) functions like Gaussian and Polynomial kernels to project the data into a higher-dimensional space where separation becomes possible.

Training and Testing

The SVM model was trained on both audible and inaudible audios (.wav format). About 9996 audio samples (4998-audible and 4998-inaudible) were used for the training. These audible and inaudible audios were stored in separate folders.

The audio samples were first preprocessed, which involved noise reduction, silence removal, and length normalization. Audible samples were sampled at a rate of 16000 Hz and inaudible ones at 96000 Hz. The preprocessed audio samples were framed using the Overlapping framing method. It is a technique that is used to divide a continuous au-

dio signal into small, overlapping frames before further analysis. This method helps to preserve temporal continuity and ensures that important features are not lost at frame boundaries. It also smoothens the transition between frames, improving model performance. After framing, the audio files were windowed using the Hamming Window method. It is a type of window function that is used to smooth the edges of a framed audio signal. It reduces spectral leakage, which occurs when sharp edges in the time domain introduce unwanted frequency components into the spectrum. Both framing and windowing output files were saved in .npy (NumPy array file) format. The windowed outputs were used to extract the features of both audible and inaudible samples separately. To build a robust classifier, the spectral, temporal, and other audio-related features were extracted. These features were saved in .npz (NumPy Zipped Archive) format. After extraction, features of both samples were combined and it underwent feature selection using the Recursive Feature Elimination (RFE) method. It is a feature selection technique that is used to improve the performance of machine learning models by systematically removing less important features. Not all the features contribute equally to classification accuracy. So, RFE helps by identifying and retaining only the most relevant features, which improves model efficiency and prevents overfitting.

Finally, these selected features were scaled using Z-Score normalization to have a mean of 0 and a standard deviation of 1. This ensures that all the features are contributed equally to the model. The saved scaler was used for training the data. And the trained model was tested and validated on 230 unseen audio samples recorded from different speakers.

5.4.2 Alert Generation and Transcription

This final step is responsible for actionable outcomes based on the detection results. It includes:

- **Alert Generation:** When the detection model detects an ultrasonic voice command, it raises an alert to notify users of a possible attack indicating a security threat . This alert can caution the users, allowing them to take appropriate actions to protect their data and device.
- **Command Transcription:** By using Google's Speech Recognition Engine, the

system aims to translate the message within the inaudible input. This functionality can help the users understand what the attacker is trying to gain control over.

Prior to this, the input command is demodulated to bring in the effect of nonlinearity in microphones as discovered by research. This feature is the main reason why Voice-Controlled Systems are able to decode the command within an ultrasound modulated input.

The process starts by applying a Butterworth low-pass filter to the input signal that removes high frequency noise by allowing only those frequencies to pass through that are below a specified cutoff frequency giving a filtered signal. The filtered signal which is a modulated signal, is multiplied by a high frequency carrier wave that shifts it back to the baseband frequency. After this, the signal is passed through a low-pass filter to eliminate residual high frequency components, leading to extraction of the message in the modulated input. This extracted part of the audio is provided to the speech recognition system for decoding.

This chapter lays down the proposed methodology along with elaborate description of the strategies and algorithms used to implement the system that achieves the objectives of this project.

Chapter 6

Results and Discussions

6.1 Overview

This section provides an overview of the results that was obtained from the three different machine learning classification models, namely - Convolutional Variational AutoEncoder (CVAE), One-Class Convolutional Neural Network (CNN), and Support Vector Machines (SVM) - for differentiating between benign and malicious inputs. All three models were trained and tested on the preprocessed input. The models were assessed based on performance metrics such as accuracy, precision, recall, and F1-score. Confusion matrix was also used to illustrate the models' performances. The CVAE showed strong anomaly classification capabilities using reconstruction errors, and efficiently identified data that deviated from normal. This model showed the best results and was subsequently selected as the final model that was used for alert generation and command recognition. The SVM also yielded good results with a well-defined decision boundary. However, it required both classes of data for training which made the model a poor fit for the intended objective of the project. The One-Class CNN demonstrated good precision in distinguishing normal and anomalous audio inputs when trained exclusively on normal data. However, results indicate that this model may need additional modifications as it shows highly specific behaviour. The results confirm the feasibility of a lightweight, software-based defense against inaudible voice attacks.

6.2 Quantitative Results

6.2.1 Convolutional Variational AutoEncoder

A dataset containing 14,208 samples was used for training and validation. The dataset was split in a ratio of 80:20, where 80% of the samples were used to train the model and

the remaining 20% were used as a validation dataset. The reconstruction errors obtained during training were observed to be in the range of 0.7 to 0.9, and the KL Divergence loss was 9551.2090 during the first epoch, which later reduced to a value close to 0 towards the 10th epoch to a value of 0.0014, which indicates that the model is working as a true variational autoencoder by sampling from a probabilistic Gaussian distribution to reconstruct instead of just reconstructing the given input. The threshold required for the classification is calculated from the reconstruction errors obtained from the validation dataset, and it was found to be 1.0070. During testing, the model uses this threshold value to classify the given input as either audible or inaudible based on the reconstruction error of the provided input. The model was tested on the dataset that was collected, which contains 230 samples. Out of 230 samples, 115 were used as audible samples, and the remaining 115 were converted to ultrasonic to be used as inaudible samples. The model achieved an accuracy of 95.65%, with a precision of 92.0% and a recall of 100%. The confusion matrix of the Convolutional VAE is shown in Figure 6.1

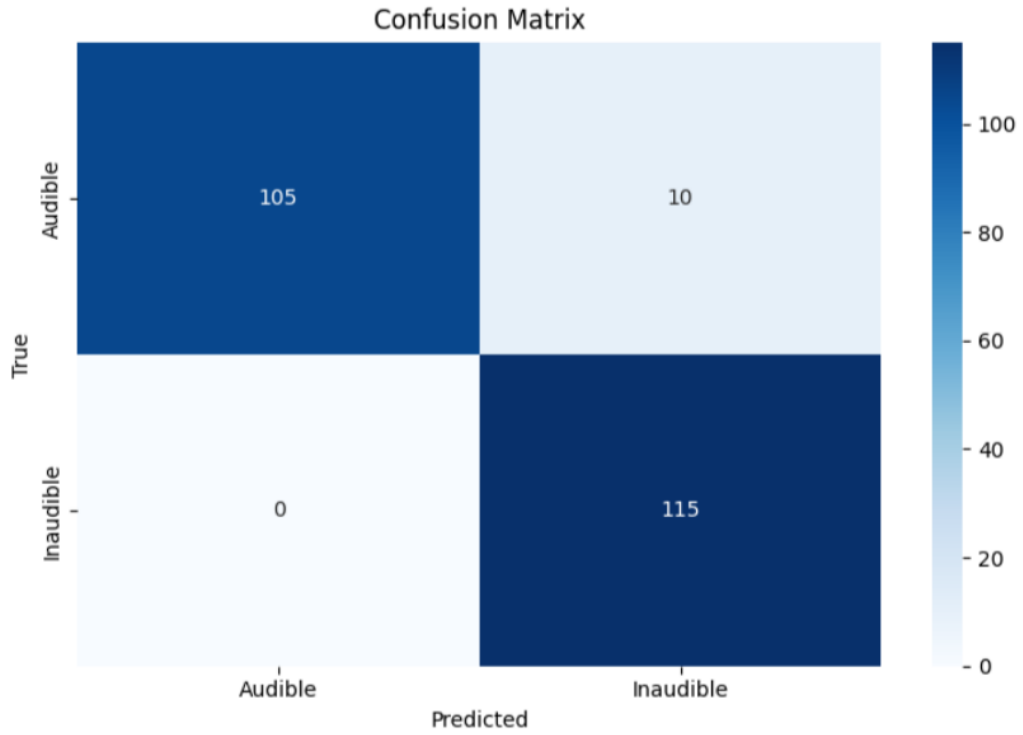


Figure 6.1: Confusion Matrix - Audible vs. Inaudible using CVAE

The latent space representation generated by the model has been visualized and is as shown in Figure 6.2:

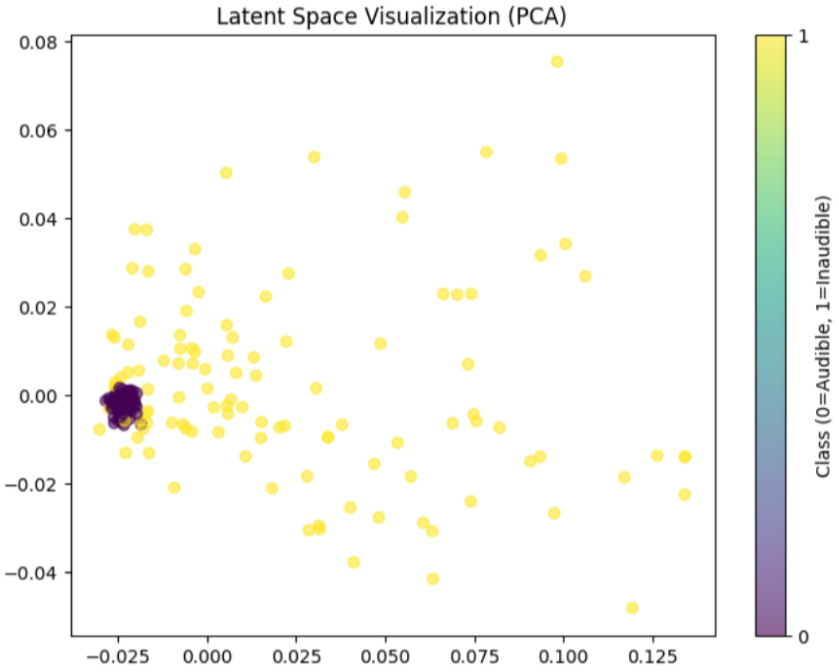


Figure 6.2: Latent Space Visualization

This model has displayed accurate results when used to simulate the working of a microphone and has been able to correctly classify any input into one of the two classes. An alert is generated, followed by the decoding of an embedded command when the classification result is "inaudible," while audible inputs pass freely.

6.2.2 One-Class Classification CNN

Strong classification performance was obtained by the one-class convolutional neural network (CNN) trained for spectrogram anomaly detection. Assuming that only normal (non-anomalous) spectrograms were available for training, the model was trained using Mean Squared Error (MSE) as the loss function. Anomalous inputs are likely to produce large anomaly scores, which serve as the foundation for anomaly detection. On evaluation, using a threshold set at 0.000002 (Figure 6.4), the model achieved an accuracy of 85.22%, a precision of 98.80%, a recall of 71.30%, and an F1-score of 82.83%.

As shown in the confusion matrix in Figure 6.3 the model correctly identified 114 out of 115 normal samples, with only 1 false positive. For the anomalous samples, 82 were correctly detected, while 33 were missed.

The model threshold was set at the 35th percentile of the normal training scores - this indicates that a considerable number of normal test samples scored higher than what the model learned to expect from its training data. This indicates that the model is highly sensitive to small changes in input data, like background noise, recording devices, or environments. While the model demonstrates good capability to identify anomalous samples, it needs to be adjusted to handle natural variations in real-world input. This can be addressed by exposing the model to a broader range of normal audio examples during training or by fine-tuning it using samples that are more specific to the conditions in which it will be deployed.

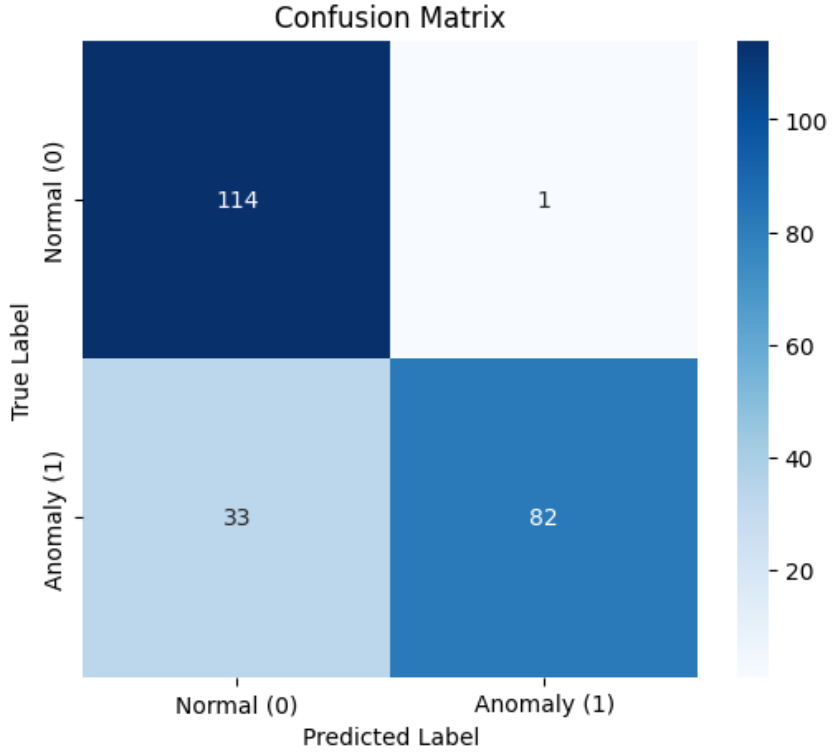


Figure 6.3: Confusion Matrix - Audible vs. Inaudible using One-Class CNN

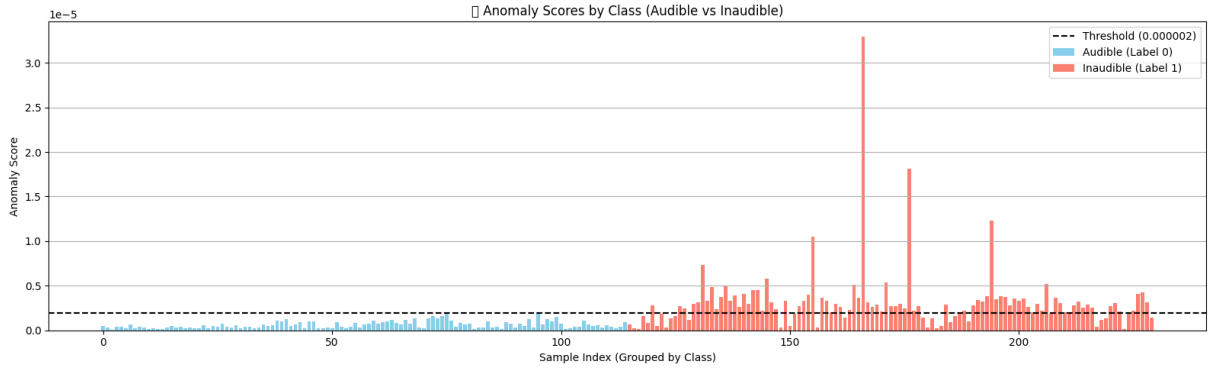


Figure 6.4: Visual representation of threshold used for classification

6.2.3 Support Vector Machines (SVM)

The SVM model was trained for classifying audible and inaudible signals based on extracted audio features. The dataset consists of two classes: audible (16 kHz sampling rate) and inaudible signals (96 kHz sampling rate, modulated). The audio signals were framed and windowed prior to training to get them ready for feature extraction. Framing divides the audio into small, overlapping segments, ensuring that transient features are not lost. For this purpose, the Hamming window was applied to each frame to smooth discontinuities at the edges and minimize spectral leakage. After the preprocessing, feature extraction was performed on each windowed segment. Features extracted were MFCCs (Mel-Frequency Cepstral Coefficients), spectral features (centroid, bandwidth, and contrast), and temporal features (zero-crossing rate, energy, and entropy). These features captured the differences between audible (benign) and inaudible (malicious) signals effectively. To reduce the computation time and to improve the accuracy of the model, Recursive Feature Elimination (RFE) was applied. This method repeatedly eliminated the least important features, leaving only the most relevant ones for training.

After feature extraction and selection, the output samples were scaled, or normalized, using the Z-Score Normalization approach, where the features were equally contributed to the model. The SVM model was trained against the labeled dataset of audible and inaudible signals. The training and testing sets were divided using the 80:20 splitting approach to assess the model performance. The classifier achieved high accuracy (100%), confirming the effectiveness of the selected features and its robustness in handling ultra-

sonic signals. Performance metrics such as precision, recall, and F1-score showed that the SVM model was highly reliable in classification. The model demonstrated strong generalization capabilities, making accurate predictions on unseen test samples. The model maintained strong performance, correctly classifying new instances of audible and inaudible signals. This confirms that the feature extraction and selection procedures generalized the classification process effectively. A confusion matrix (in Figure 6.5) was used to represent the prediction. An optimal hyperplane was also identified that classified both audible and inaudible classes perfectly (in Figure 6.6).

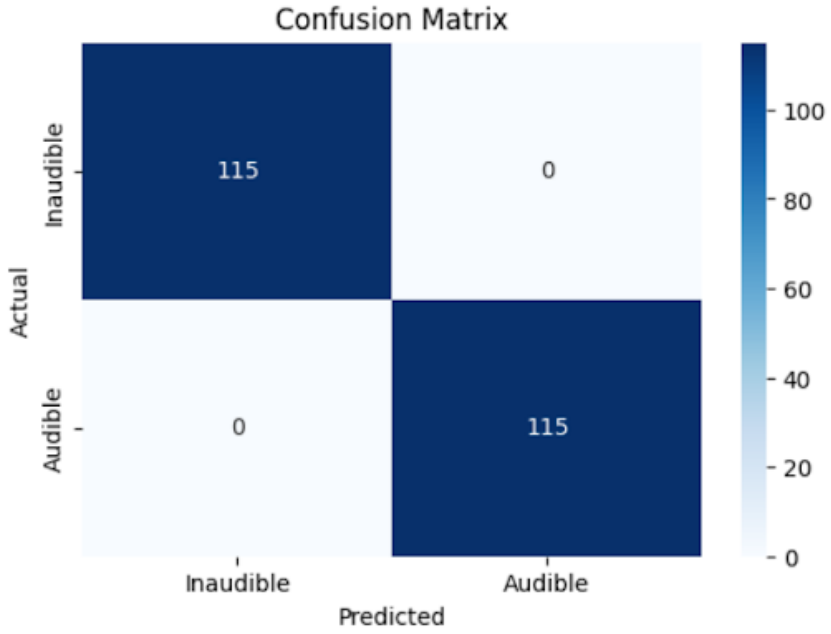


Figure 6.5: Confusion Matrix - Audible vs. Inaudible using SVM

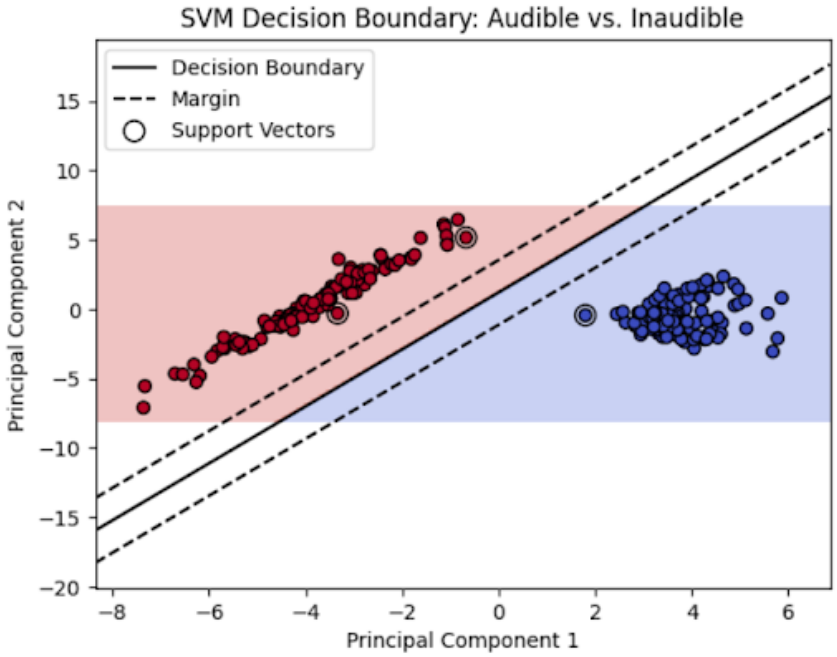


Figure 6.6: SVM Decision Boundary and margins used in classification

6.3 Testing

The entire system of microphone simulation was tested for both audible and inaudible commands using the trained and tested Convolutional Variational AutoEncoder(CVAE) model. The output obtained for samples of the both the classes are shown in Figure 6.7 and Figure 6.8.

```
[23]: input_path = "/kaggle/input/testaudios/audio1.wav"
      #input_path = "/kaggle/working/modulated_audio6.wav"

      + Code   + Markdown

[26]: microphone_simulation(input_path)

Reconstruction Error: 0.8393
Threshold: 1.0070

CLASSIFICATION: Audible (Normal)
Everything is good!
```

Figure 6.7: Audible Input

```
[27]: #input_path = "/kaggle/input/testaudios/audio1.wav"
      input_path = "/kaggle/working/modulated_audio6.wav"

[28]: microphone_simulation(input_path)

Reconstruction Error: 1.6239
Threshold: 1.0070

CLASSIFICATION: Inaudible (Anomaly)
ALERT - Inaudible command detected!!
Recognized Command: OK Google restart my phone now
```

Figure 6.8: Inaudible Input

The Figure 6.7 shows the output obtained when an audible sample is provided as input to the system developed. From the output, it can be seen that the model has acheived a reconstruction error of 0.8407. In the output of Figure 6.8, the reconstruction error is high with a value of 1.6305 for inaudible input. Since this value is greater than the threshold value, the classification result obtained is "Inaudible".

6.4 Discussion

The objective of this project was to develop a software based defense that could be used to classify between normal and ultrasonic audio commands, thus serving as a reliable defense against inaudible voice attacks. In the project a comparison was undertaken to select the most approrpiate model for classification between anomalous and normal audio commands. This could then be used to subsequently create an alert notifying the user of a potential attack. The model that was chosen was the Convolutional Variational Auto-Encoder (CVAE). This model provides an adaptable solution and can be trained on only normal spectrograms, making it a viable method that can be installed and used in voice-enabled devices to detect inaudible voice commands.

- The CVAE has better accuracy of 95.65%, performing better than the One-Class CNN, which achieved only 82.5%. These two models are trained exclusively on

normal audio samples.

- The CVAE was trained only on normal data, which is easier to collect in practical scenarios making it a more viable solution. Although the SVM achieved 100% accuracy, it required data from both normal and anomalous classes, which make training more difficult.
- The CVAE provided a more adaptable solution. While the One-Class CNN also delivered good results, it was highly sensitive to changes between the training and testing datasets, resulting in the need for careful fine-tuning for each use case.

6.5 Summary of the Detection Models

Models	**CVAE**	**CNN (One-Class)**	**SVM**
Accuracy	95.65%	82.5%	100%
Training	Only on audible audios	Only on audible audios	On both audible and inaudible audios
Classification Basis	Reconstruction error threshold	Anomaly Score Threshold	Decision Boundary (Hyperplane)

Figure 6.9: Summary of the models used

Chapter 7

Conclusion & Future Scope

The use and growing popularity of speech recognition systems increases convenience but brings about their own set of risks. Although these devices are widely used, attacks exploiting inaudible voices have been observed. A survey of the literature covering the nature type and defenses of inaudible voice attacks was conducted, as well as an examination of the nonlinearity of the microphone which is the primary source of this attack. A comparison of the surveyed research papers was carried out at the conclusion of the survey outlining the benefits and drawbacks of each paper. The survey also revealed gaps in the literature. The system architecture and implementation chapter described the proposed methodology and it's architecture, description of each component of the system and implementation strategies, the datasets, data flow diagrams, algorithms used and the results obtained were then presented in the subsequent chapters.

Three models were considered for the implementation of the software defense. Convolutional Variational Auto-Encoder (CVAE), One-Class Convolution Neural Network (One-Class CNN) and Support Vector Machine (SVM). The comparison yielded the best results for the CVAE and SVM, with 95.65% and 100% accuracy respectively. The One-Class CNN had an accuracy of 82.5%. However, the CVAE was selected for the implementation of the final defense as it was trained only on normal models, making it more viable and adaptable. With the intention of preventing such inaudible voice attacks this report is concluded with the information presented above.

The objective of this project can be further expanded by improving the existing model. A detailed study of the exact features that the model learns and their corresponding significance in the accuracy of the prediction can be studied using ExplainableAI (XAI). Improving the accuracy and optimising model functionality along with integration of the

developed system with a user-friendly application that delivers alerts in realtime to the users is also within the scope of the project. With the rising influence of voice-enabled devices for convenience and accessibility it is vital to provide a safe and reliable environment in which the technology can flourish. The objective has been achieved through the execution of this project but can be further adapted as newer advancements are made.

References

[1] G. Zhang, C. Yan, X. Ji, T. Zhang, T. Zhang, and W. Xu, "Dolphinattack: Inaudible voice commands," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. Association for Computing Machinery, 2017, p. 103–117. [Online]. Available: https://doi.org/10.1145/3133956.3134052

[2] N. Roy, S. Shen, H. Hassanieh, and R. R. Choudhury, "Inaudible voice commands: the long-range attack and defense," in *Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'18. USENIX Association, 2018, p. 547–560.

[3] X. Li, X. Ji, C. Yan, C. Li, Y. Li, Z. Zhang, and W. Xu, "Learning normality is enough: a software-based mitigation against inaudible voice attacks," in *Proceedings of the 32nd USENIX Conference on Security Symposium*, ser. SEC '23. USENIX Association, 2023.

[4] M. Zhou, Z. Qin, X. Lin, S. Hu, Q. Wang, and K. Ren, "Hidden voice commands: Attacks and defenses on the vcs of autonomous driving cars," *IEEE Wireless Communications*, vol. 26, pp. 128–133, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:149866388

[5] L. Lugosch, M. Ravanelli, P. Ignoto, V. S. Tomar, and Y. Bengio, "Speech model pre-training for end-to-end spoken language understanding," in *Proc. of Interspeech*, G. Kubin and Z. Kacic, Eds., 2019, pp. 814–818.

Appendix A: Presentation

DOLPHINATTACK DETECTION

FINAL PROJECT PRESENTATION

Guided by
Dr. Anita John

Group 3
Nandana S Pillai
Ritu Maria
Shaun Mammen John
Shreya Veeraraghav

CONTENTS

INTRODUCTION

- Speech recognition systems reveal a vulnerability to an attack where inaudible voice commands are transmitted via ultrasonic carriers.
- The inaudible commands can covertly control devices, highlighting the need for effective defenses that prevent such attacks.
- The project aims at detecting such attacks.

PROBLEM DEFINITION

- Voice recognition systems vulnerable to inaudible voice commands transmitted via ultrasonic carriers
- The inaudible commands can covertly control these devices without user awareness.
- Need for a defensive solution that can detect, classify, and alert users about ultrasonic attacks, ensuring the safety and proper functioning of voice-controlled devices.

PURPOSE AND NEED

- In today’s digital era, where the use of voice control for device connectivity is on the rise, attackers are developing sophisticated techniques to gain unauthorized control over voice-controlled devices.
- This poses significant risks, including data theft, financial manipulation, and unauthorized access to sensitive information.
- This creates an urgent need for a robust defense system capable of detecting malicious commands and alerting users to potential attacks, ensuring safety.

PROJECT OBJECTIVE

- To develop a machine learning model for detecting and classifying ultrasonic audio from benign audio in the scene of an attack on voice recognition systems, indicated by creating alerts to the user.

MODULES

1. TEST DATASET CREATION
2. AUDIO PREPROCESSING
3. FEATURE ENHANCEMENT (specific to each model)
4. DETECTION MODEL
   - 4.1 CONVOLUTIONAL VAE (VARIATIONAL AUTOENCODERS)
   - 4.2 ONE-CLASS CLASSIFICATION CNN (CONVOLUTIONAL NEURAL NETWORKS)
   - 4.3 SVM (SUPPORT VECTOR MACHINES)
5. ALERT GENERATION
6. COMMAND RECOGNITION

LITERATURE SURVEY

PAPER	ADVANTAGES	DISADVANTAGES	METHODOLOGY
***DolphinAttack: Inaudible Voice Commands*. [1]**	• Implements software + hardware defenses • Hardware independent	• Limited by distance of 5ft • Attack sample dependent	Support Vector Machine (SVM)
***Hidden Voice Commands: Attacks and Defenses on the VCS of Autonomous Driving Cars*. [5]**	• Can defend against different types of attacks • Novel Defence Strategy	• Dependence on Pop Noise Detection • Physical Proximity for Some Attacks	Two-class Support Vector Machine Classifier

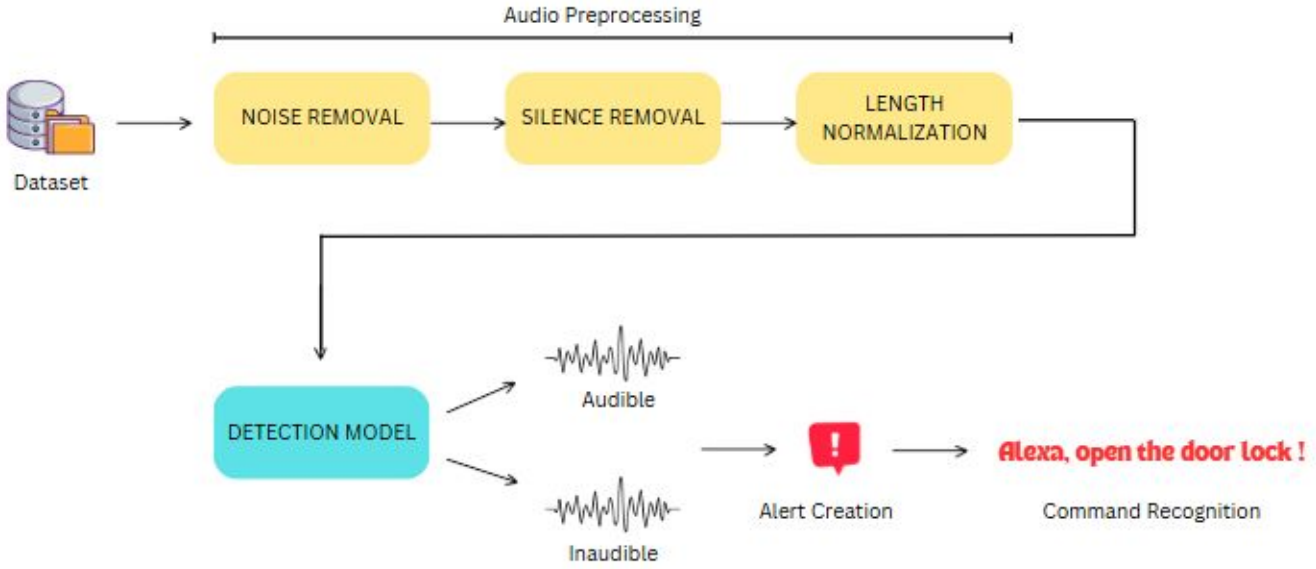
PAPER	ADVANTAGES	DISADVANTAGES	METHODOLOGY
***Learning Normality is Enough: A Software-based Mitigation against Inaudible Voice Attacks*. [2]**	• Universal • Lightweight • Hardware Independent • Attack sample independent	• Does not work for attack range of only 30 cm	Variational AutoEncoder
***Inaudible Voice Commands: The Long-Range Attack and Defense*. [4]**	• Extends attack range to upto 25ft • Accurate classification for malicious and benign audios • Hardware independent	• Requires more hardware to increase range • Attack sample dependent	Elliptical Classifier

DATASET

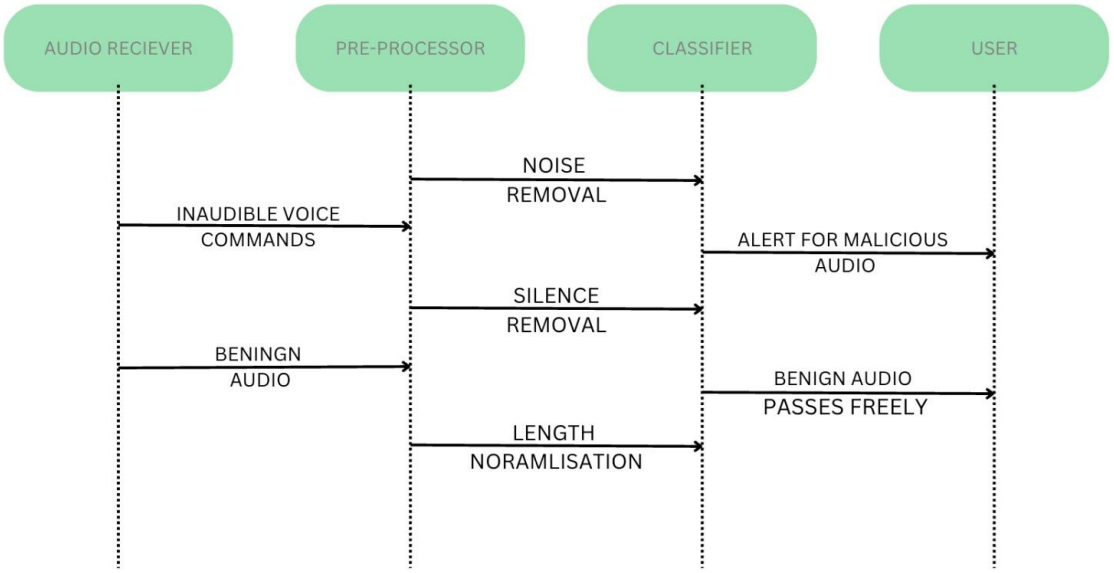
- **DATASET (Training):** "Fluent Speech Commands Dataset"
- Open-source audio dataset for spoken language understanding experiments.
- The audio is recorded as 16 kHz single-channel .wav files, with a total of 30,043 utterances from 97 speakers.
- https://fluent.ai/fluent-speech-commands-a-dataset-for-spoken-language-understanding-research/

- **DATASET (Testing):** 230 audio samples recorded from different speakers.

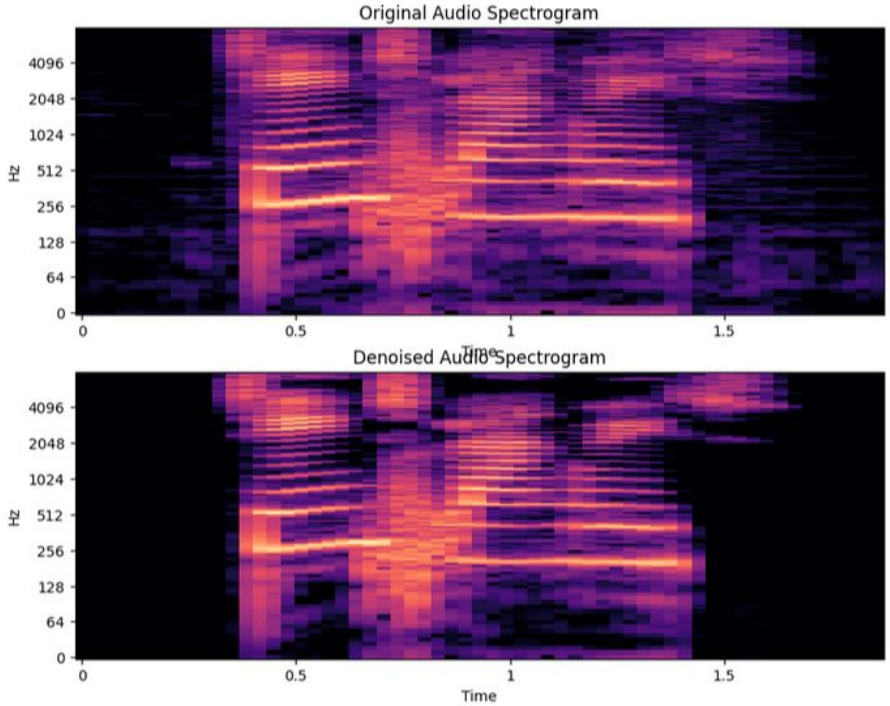
ARCHITECTURE DIAGRAM



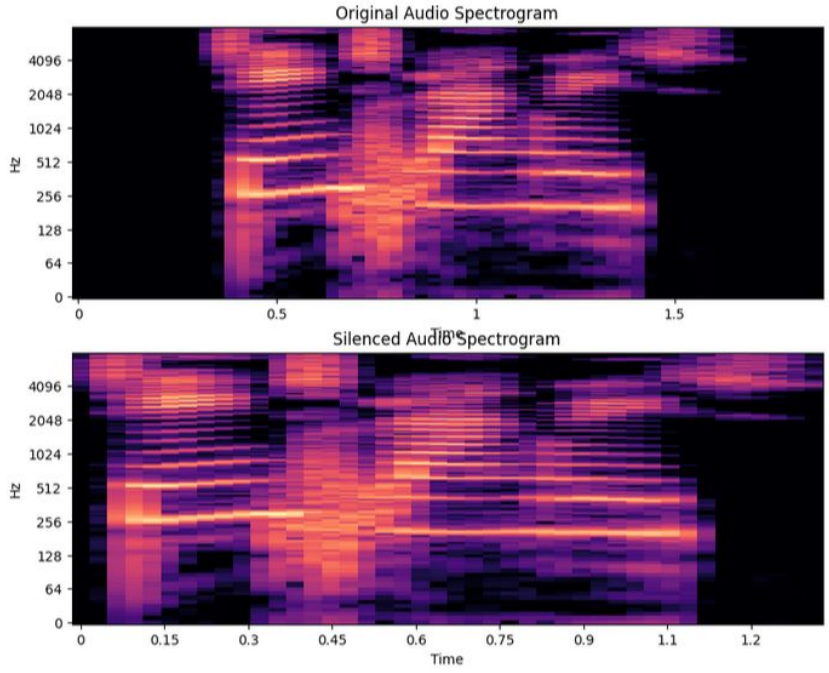
SEQUENCE DIAGRAM



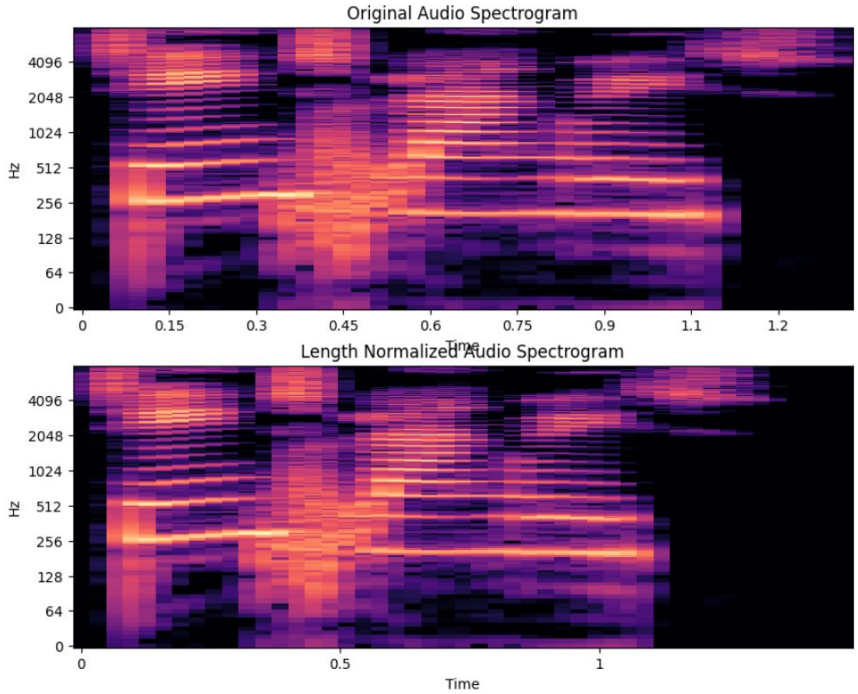
AUDIO PREPROCESSING - NOISE REDUCTION



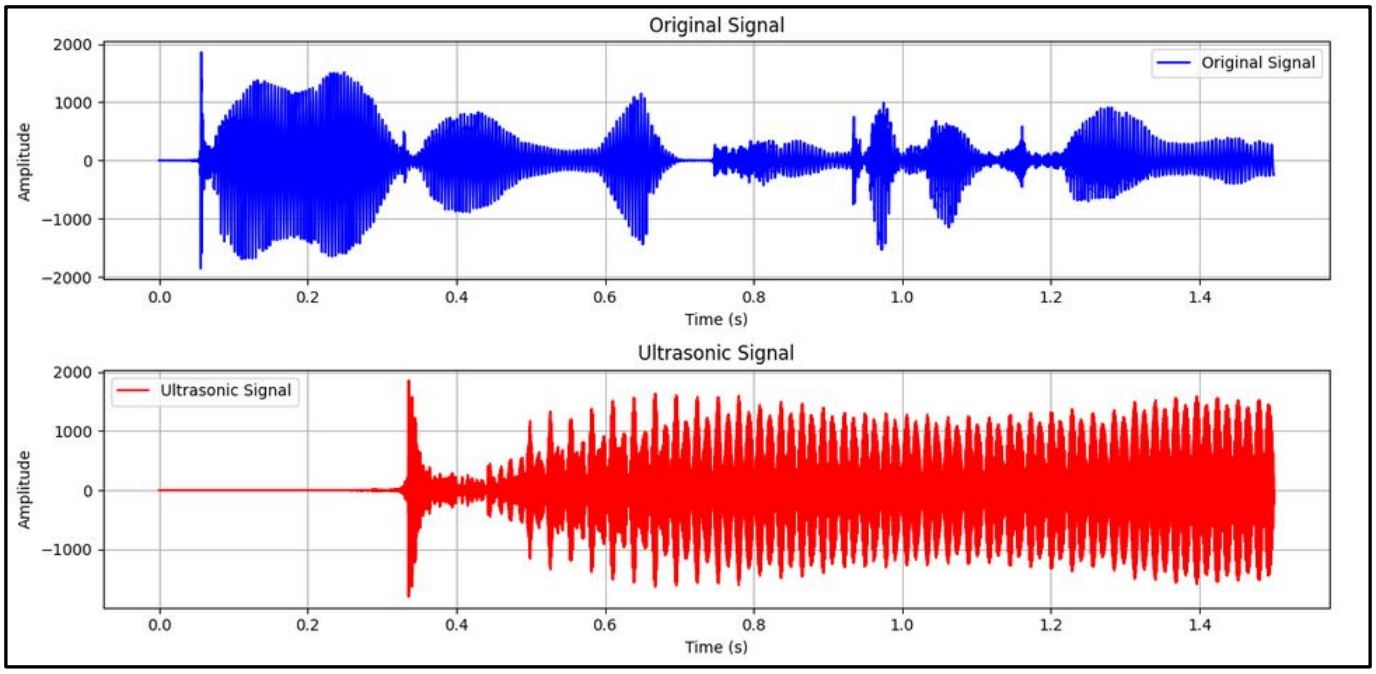
AUDIO PREPROCESSING - SILENCE REMOVAL



AUDIO PREPROCESSING - LENGTH NORMALIZATION



ULTRASONIC CONVERSION



SPECTROGRAM (Whole duration: 1.5 secs)

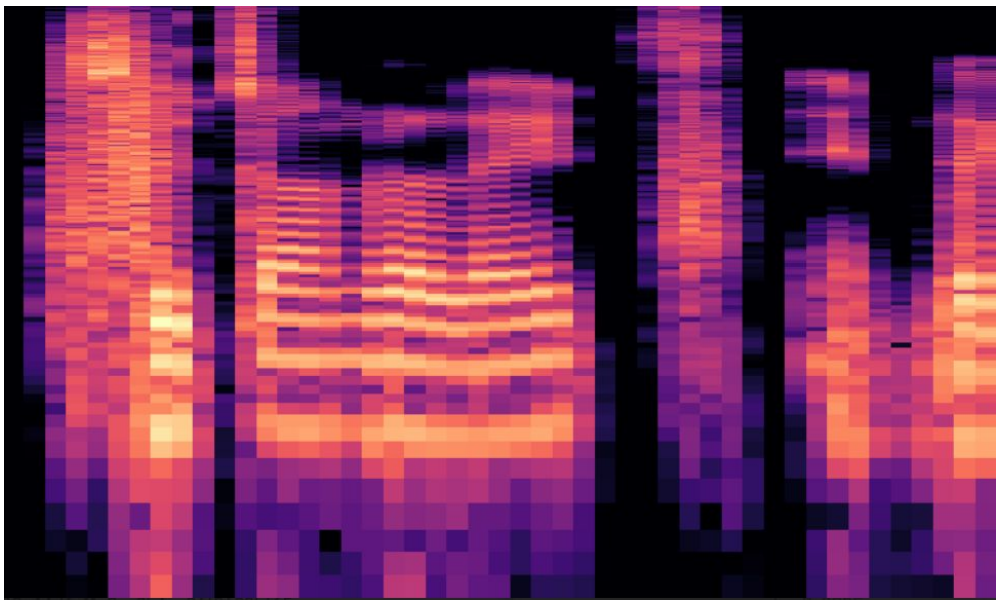


Fig: Original Spectrogram (775x462)

SPECTROGRAM (0.5 secs of the audio)

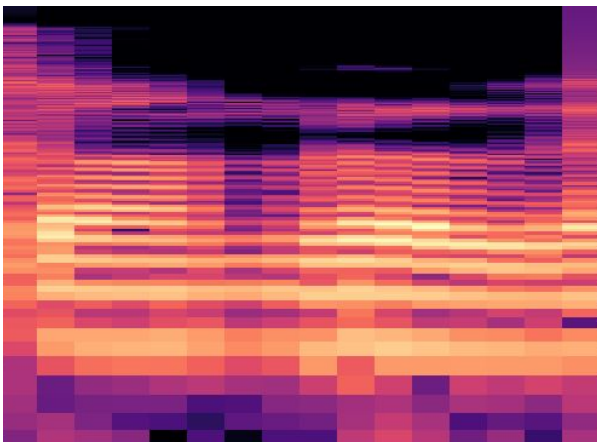


Fig: Spectrogram before resizing (496x369)

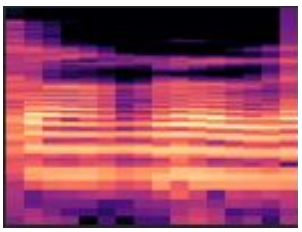


Fig: Spectrogram after resizing (128x95)

CVAE (Convolutional Variational Autoencoders)

CVAE

- A model used for unsupervised learning.
- The convolutional layers of the VAE extract essential features, capturing the intricate patterns of normal voice signals.
- **Encoder:** Compresses high-dimensional spectrums into a lower-dimensional latent space.
- **Decoder:** Reconstructs the input from this compressed representation.
- Detects inaudible attacks by evaluating the reconstruction error or latent space deviation of incoming spectrums.

CVAE

- **Input**: resized spectrogram of 0.5s of the audio - (95, 128, 1).
- **Encoder** : consists of convolutional , flatten and dense layers with ReLU activation function.
- **Decoder** : consists of dense, reshape and convolutional transpose layers with ReLU in intermediate and Linear activation in the final layer.
- Decoder uses “Reparameterization technique” to sample from latent space.
- **Loss Function** : combination of Reconstruction error and KL Divergence loss.

CVAE - Layers

```
model.encoder.summary()
```

Model: "sequential"

Layer (type)	Output Shape
conv2d (Conv2D)	(None, 48, 64, 32)
conv2d_1 (Conv2D)	(None, 24, 32, 64)
conv2d_2 (Conv2D)	(None, 12, 16, 128)
flatten (Flatten)	(None, 24576)
dense (Dense)	(None, 256)
dense_1 (Dense)	(None, 32)

```
model.decoder.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape
dense_2 (Dense)	(None, 24576)
reshape (Reshape)	(None, 12, 16, 128)
conv2d_transpose (Conv2DTranspose)	(None, 24, 32, 64)
conv2d_transpose_1 (Conv2DTranspose)	(None, 48, 64, 32)
conv2d_transpose_2 (Conv2DTranspose)	(None, 96, 128, 1)

CVAE

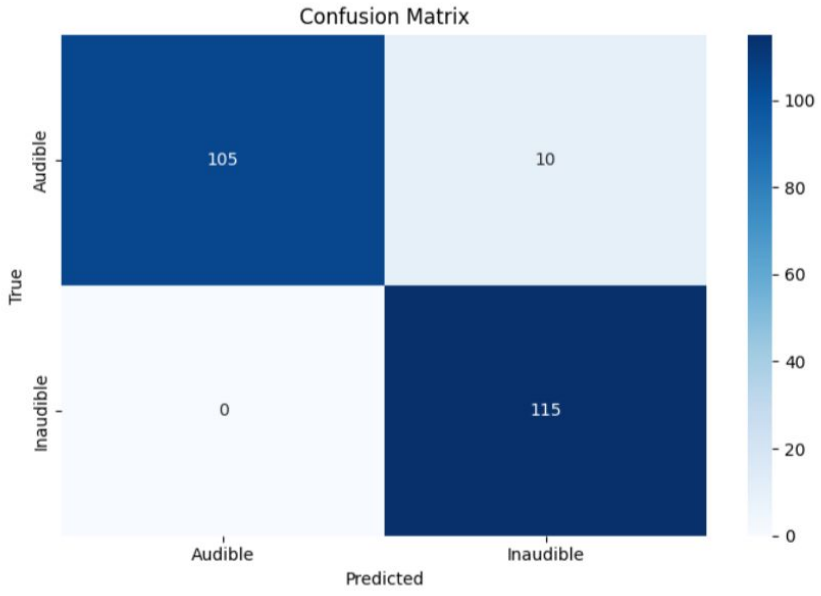
Training

- Dataset containing 14208 used for training and validation
- CVAE trained only on audible samples
- **Optimiser** : Adam
- **Learning Rate** : 1e-4
- Threshold calculated from reconstruction errors of the validation dataset
- **Obtained threshold** : 1.0070

Testing

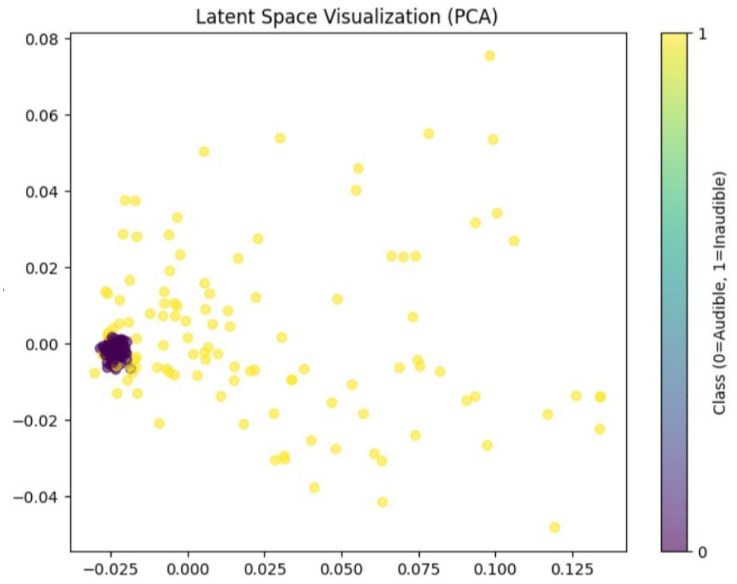
- Tested on 230 samples (115 - audible, 115 - inaudible)
- **Accuracy** : 95.65%

CVAE - Confusion Matrix




```
Confusion Matrix Statistics (Threshold=1.0070):
True Negatives: 105
False Positives: 10
False Negatives: 0
True Positives: 115
Accuracy: 0.9565
Precision: 0.9200
Recall: 1.0000
```

CVAE - Latent Space Visualization



ONE-CLASS CNN (One-Class Convolutional Neural Networks)

One-Class CNN

What is a One-Class CNN?

- A One-Class Convolutional Neural Network (CNN) is a deep learning model trained to recognize patterns in only one class of data (normal/expected data).
- It is used for anomaly detection, where any deviation from the learned pattern is flagged as an anomaly.
- Applications include fraud detection, medical diagnosis, and ultrasonic attack detection.

One-Class CNN

Input Shape

- (95, 128, 1) → Height = 95, Width = 128, 1 Channel (Grayscale)

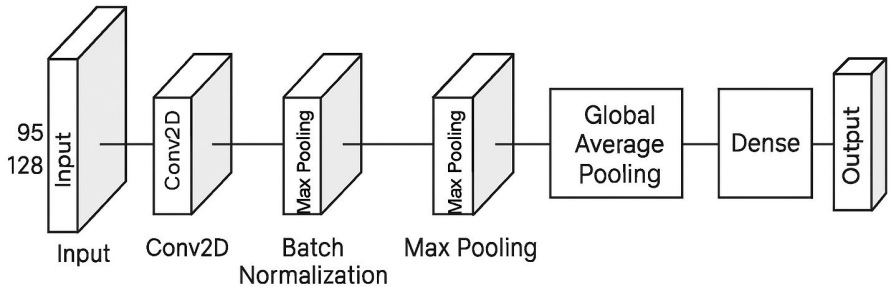
Layers in the Model

- 3 Convolutional Layers (32, 64, 128 filters, 3×3 kernel, ReLU activation)
- Batch Normalization & Dropout Layers (Prevent overfitting)
- MaxPooling Layers (Reduces spatial dimensions)

One-Class CNN

- Global Average Pooling (Flattens feature maps)
- Fully Connected Layers (128 neurons, ReLU activation)
- Output Layer: 1 neuron, Sigmoid Activation (for anomaly detection)

One-Class CNN



One-Class CNN

Loss Function - Mean Squared Error (MSE)

- Measures how far predicted values are from actual values.
- Penalizes larger errors more than smaller ones.
- Helps the CNN learn an accurate anomaly detection threshold.

One-Class CNN

Optimization - Adam Optimizer

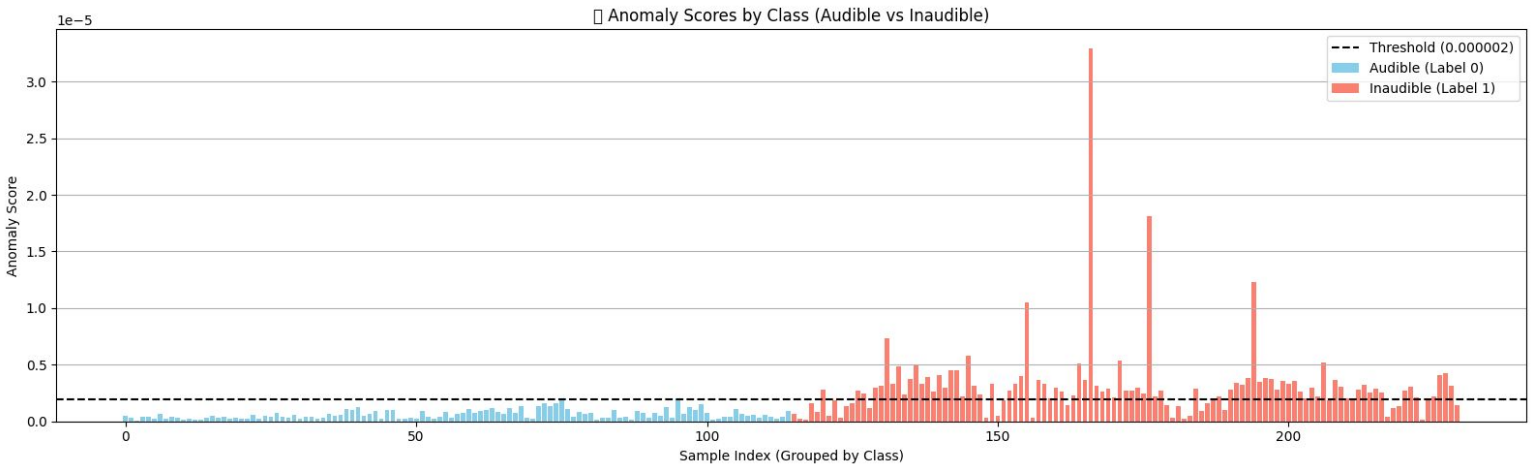
- Adaptive learning rate adjustment for faster convergence.
- Combines Momentum (smooth updates) and RMSProp (adjusts step size).
- Works well for CNNs in anomaly detection tasks.
- Computes moving averages of gradients to optimize weight updates.
- Helps CNN train efficiently with noisy data.

One-Class CNN

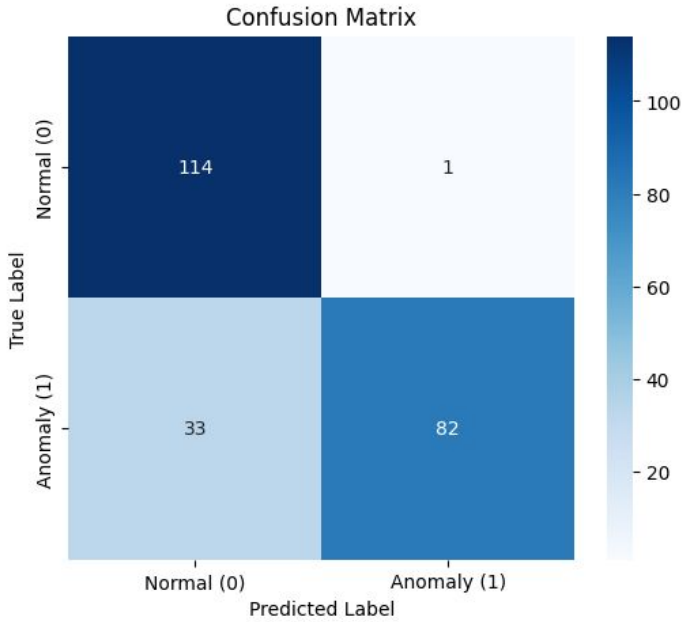
Classification Report (Approximate)

Class	Precision	Recall	F1-score	Support
Audible (0)	0.7632	0.9913	0.8644	115
Inaudible(1)	0.9880	0.7130	0.8283	115
Accuracy			0.8522	230
Macro avg	0.8756	0.8521	0.8464	230
Weighted avg	0.8756	0.8522	0.8464	230

One-Class CNN



One-Class CNN



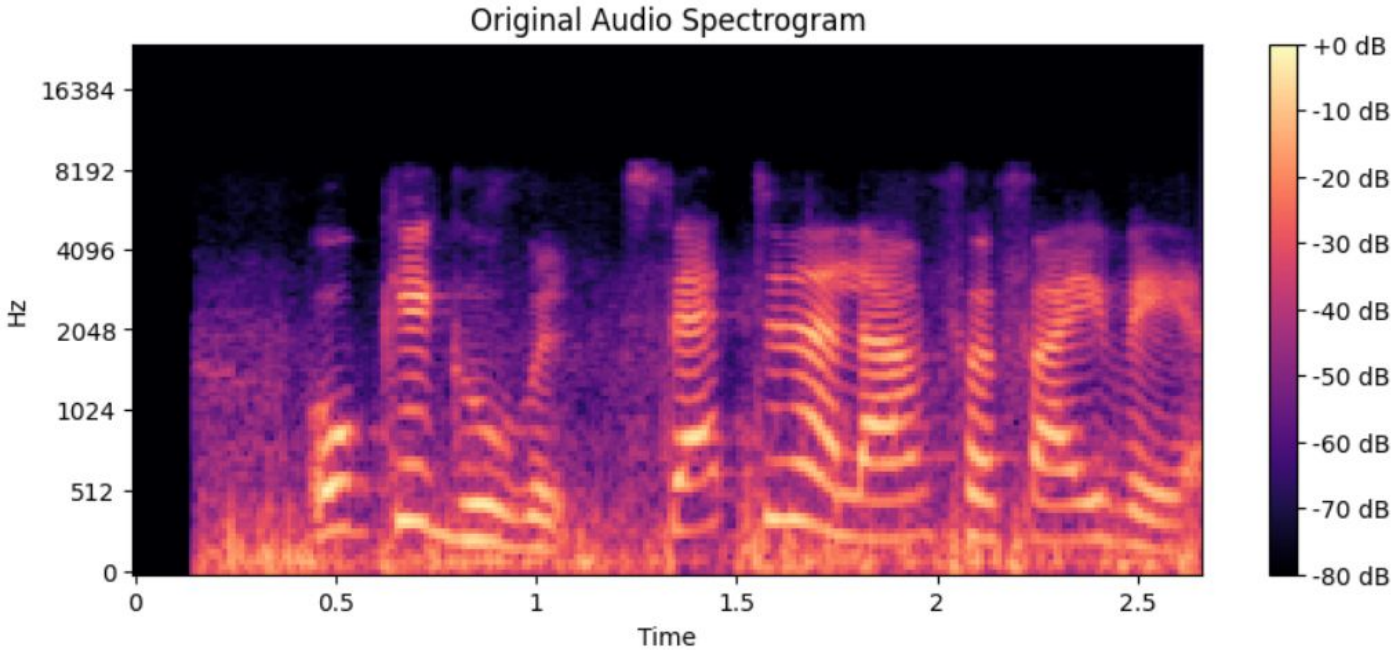
SVM (Support Vector Machines)

SVM

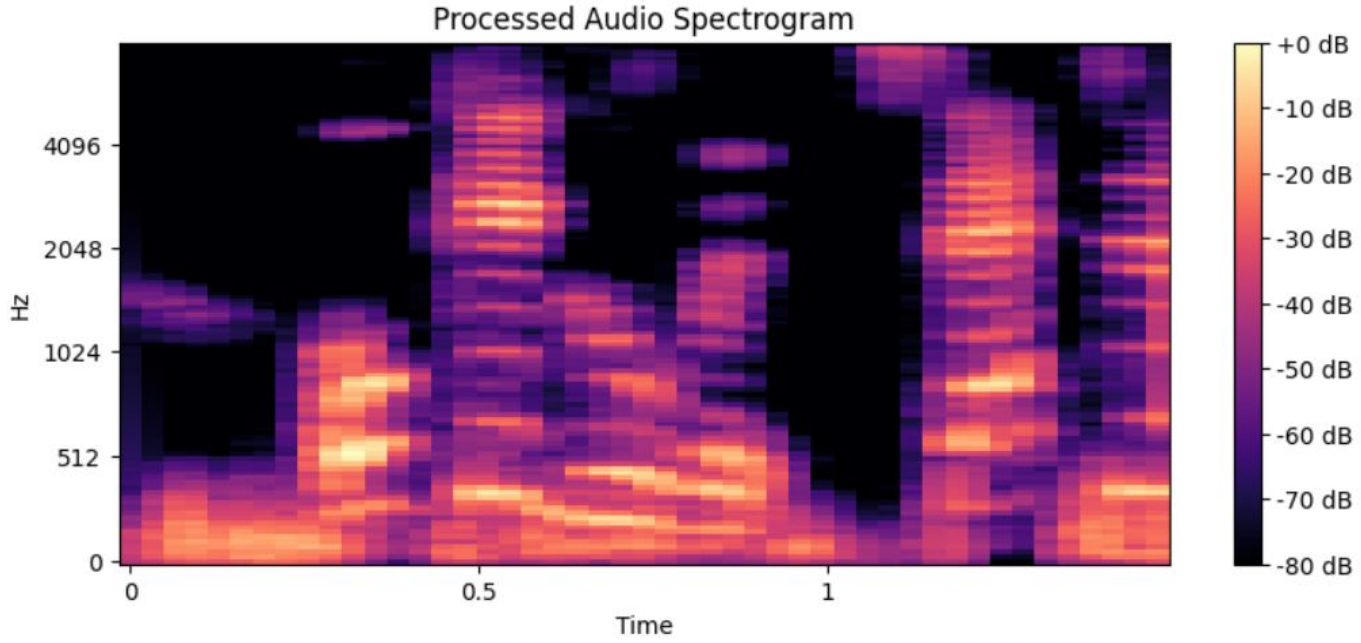
- It is a supervised learning algorithm used for classification and regression tasks. It works by finding the best hyperplane that separates different classes in a dataset.
- Methods involved:

1. Audio Preprocessing
2. Framing
3. Windowing
4. Feature Extraction
5. Feature Selection
6. Feature Scaling
7. Training, Testing, and Validation

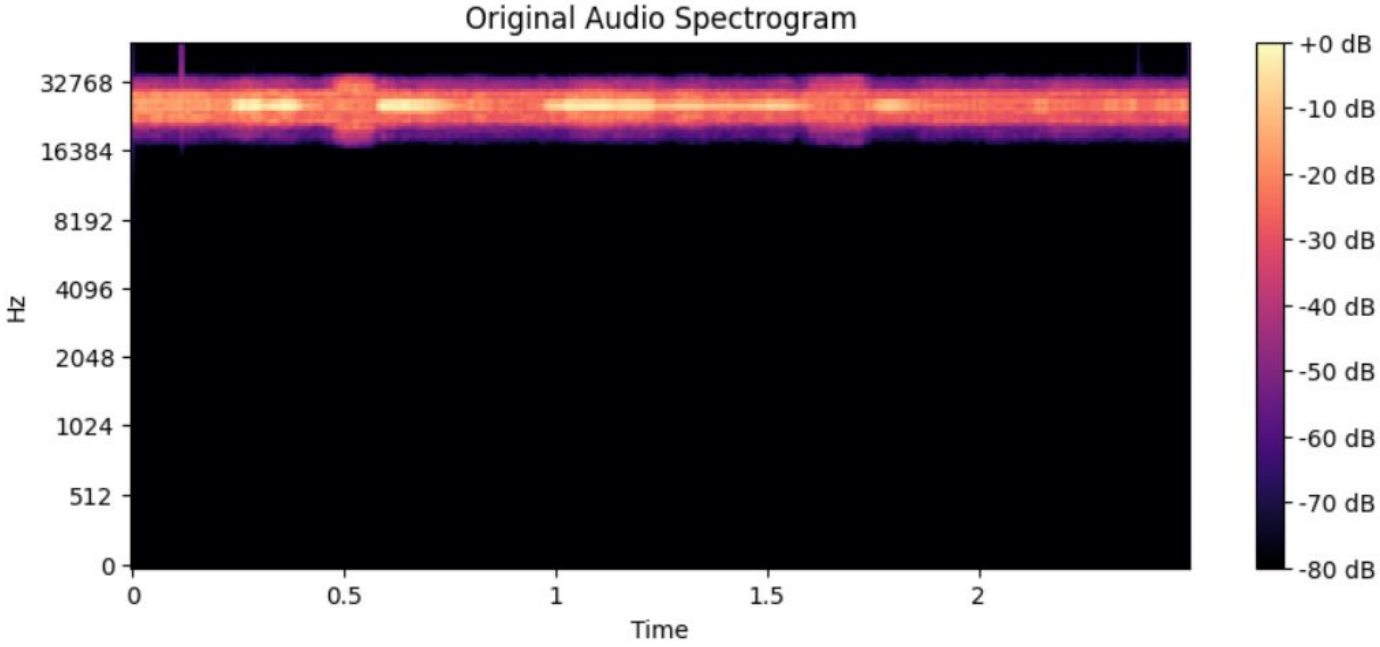
AUDIO PREPROCESSING - Audible (Before)



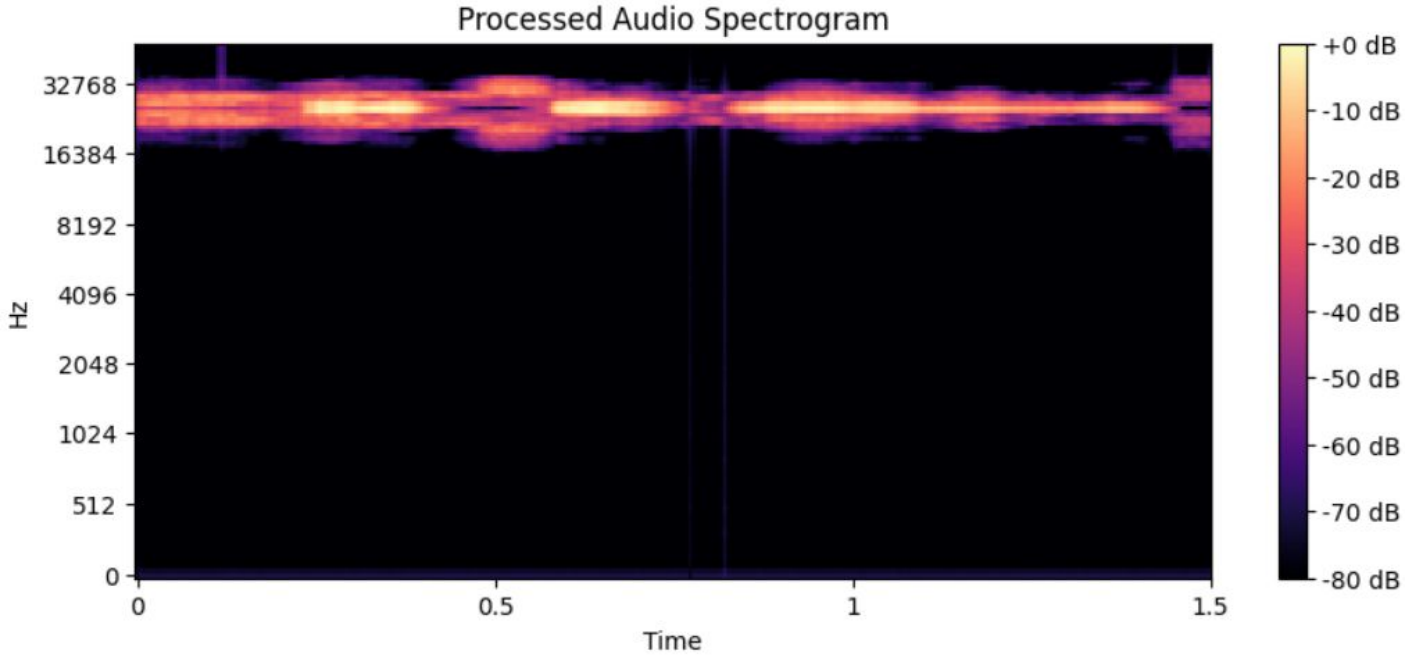
AUDIO PREPROCESSING - Audible (After)



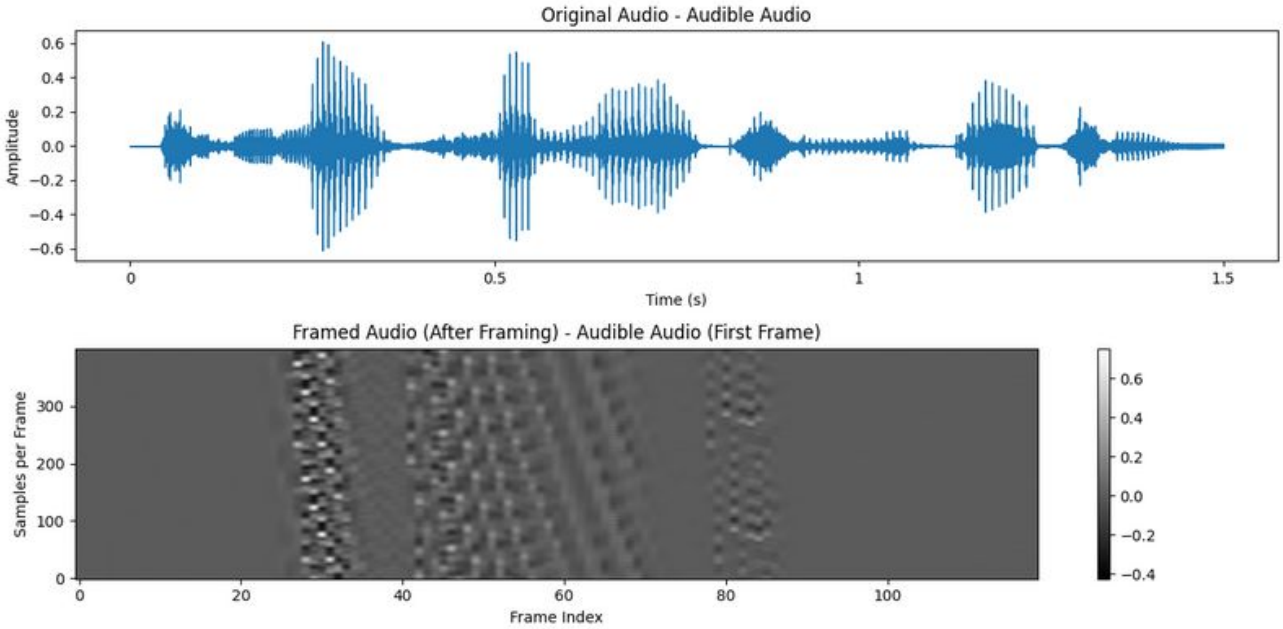
AUDIO PREPROCESSING - Inaudible (Before)



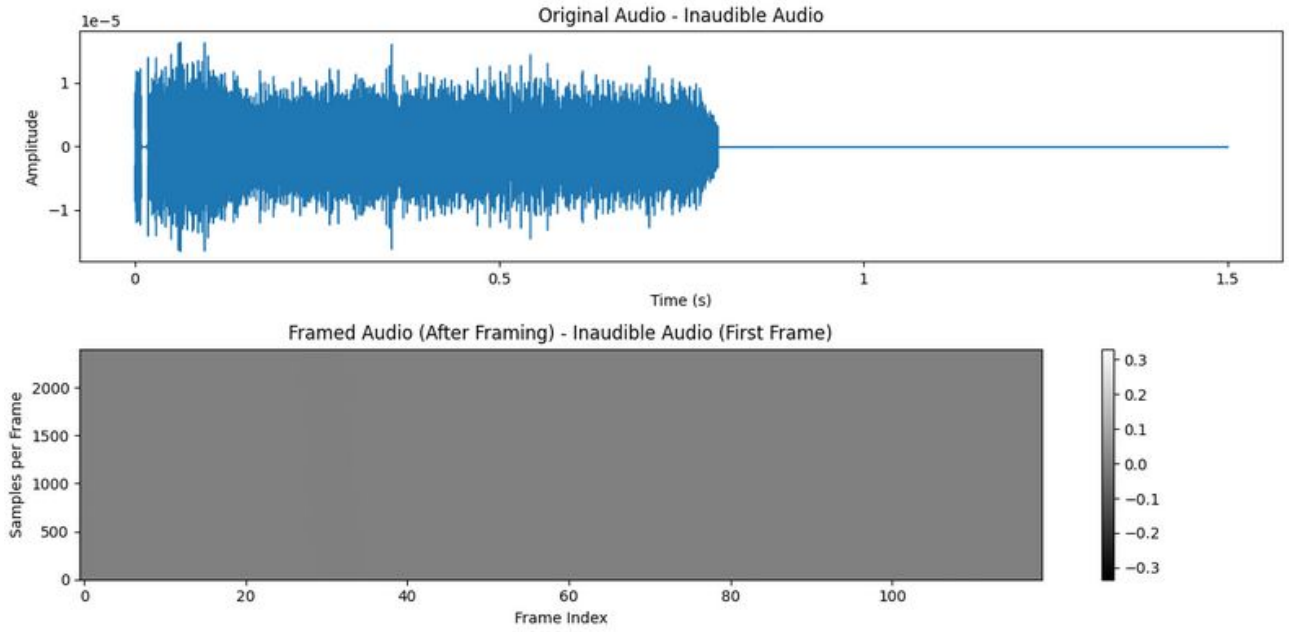
AUDIO PREPROCESSING - Inaudible (After)



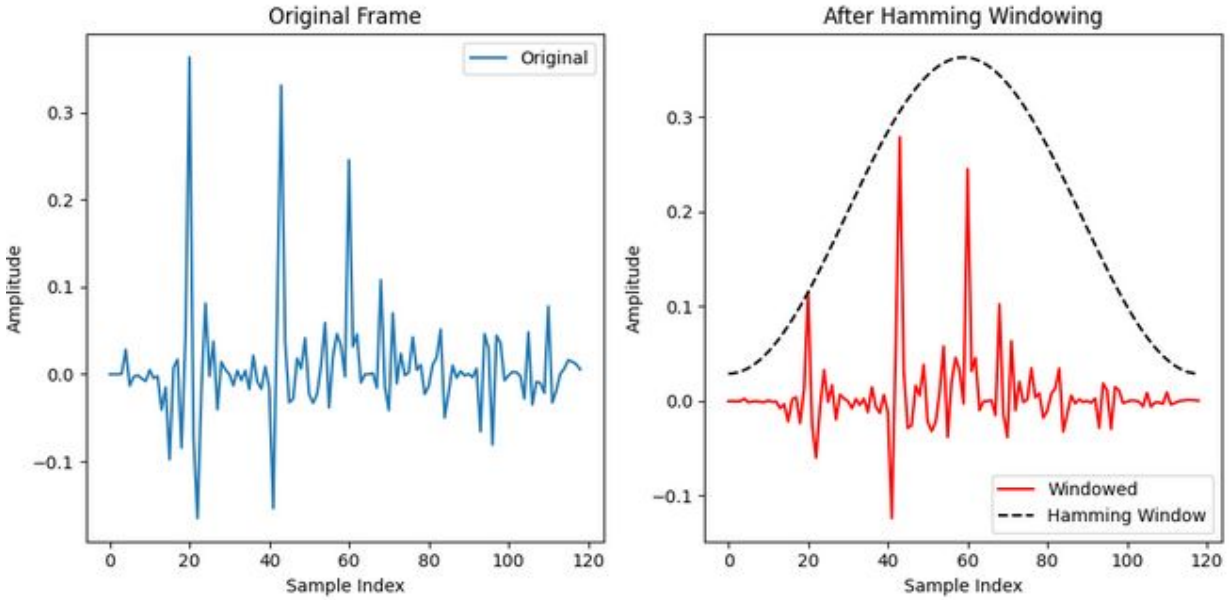
FRAMING - Audible



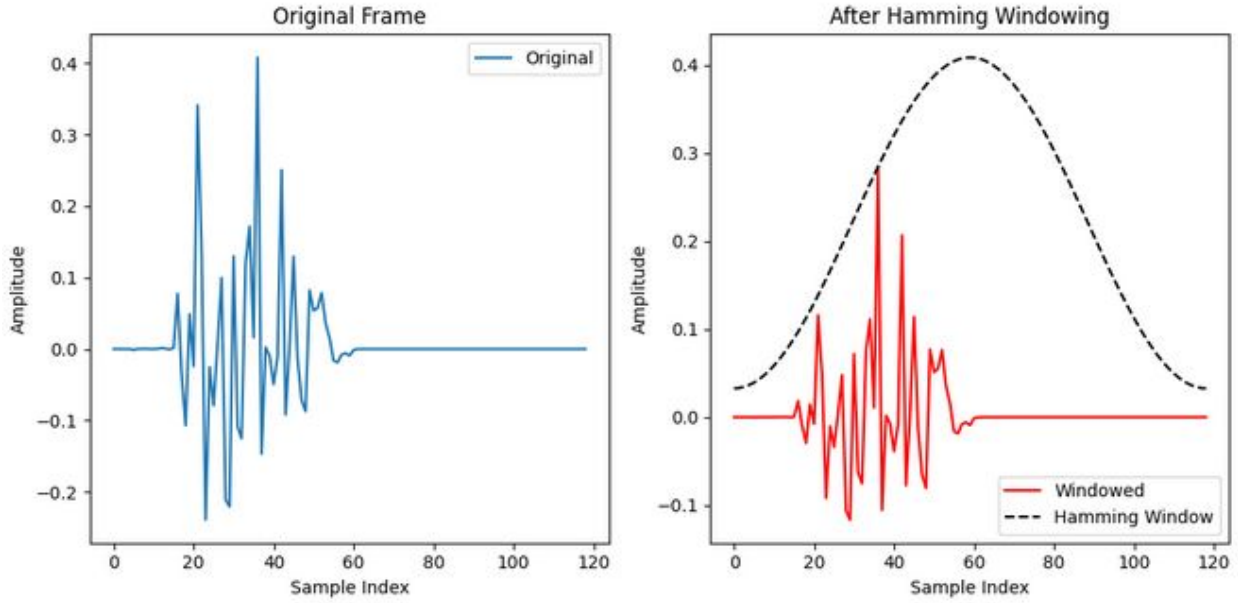
FRAMING - Inaudible



WINDOWING-Audible



WINDOWING-Inaudible



FEATURE EXTRACTION

```
Feature Matrix Shape: (9996, 30)
Labels Shape: (9996,)
```

```
Class Distribution: {0: 4998, 1: 4998}
```

- It is crucial for improving model performance, especially when dealing with high-dimensional data.
- It helps reduce complexity, improve classification accuracy, and speed up training.

FEATURE SELECTION

```
Selected Features Index: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29]
```

- It is the process of selecting a subset of relevant features for use in model construction.
- The purpose of this method is to improve the performance of a model by reducing overfitting, improving accuracy, and decreasing the computational cost.
- Used the Recursive Feature Elimination (RFE) method for feature selection.

FEATURE SCALING

- Technique used to standardize or normalize the range of independent variables or features in a dataset.
- Ensures that features have a comparable scale, preventing certain features from dominating the learning process due to differences in their magnitude.
- Centers features around zero with unit variance.
- Less sensitive to outliers.
- Used Z-Score Normalization for scaling the features.

$$X' = \frac{X - \mu}{\sigma}$$

TRAINING AND TESTING

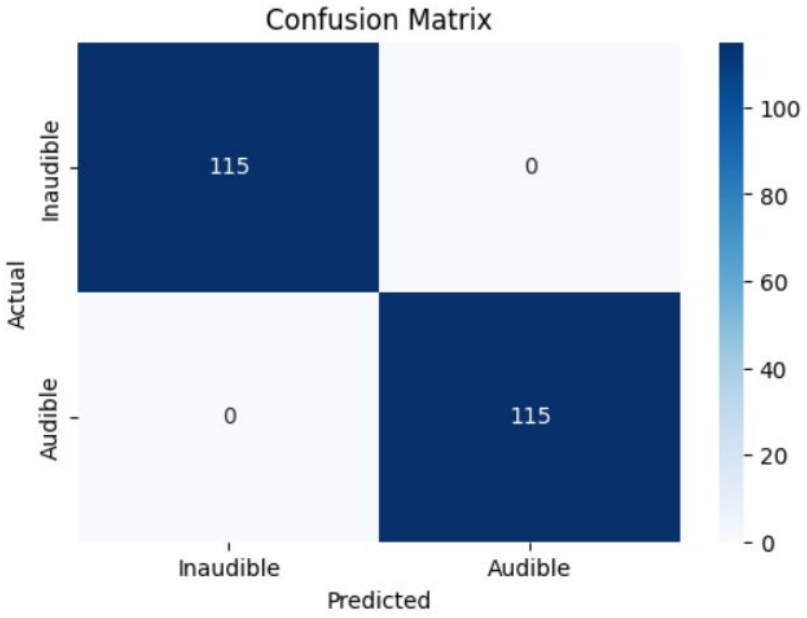
```
Test Accuracy: 100.00%
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1000
           1       1.00      1.00      1.00      1000

    accuracy                           1.00      2000
   macro avg       1.00      1.00      1.00      2000
weighted avg       1.00      1.00      1.00      2000

SVM model saved successfully!
```

```
Training Accuracy: 100.00%
Testing Accuracy: 100.00%
```

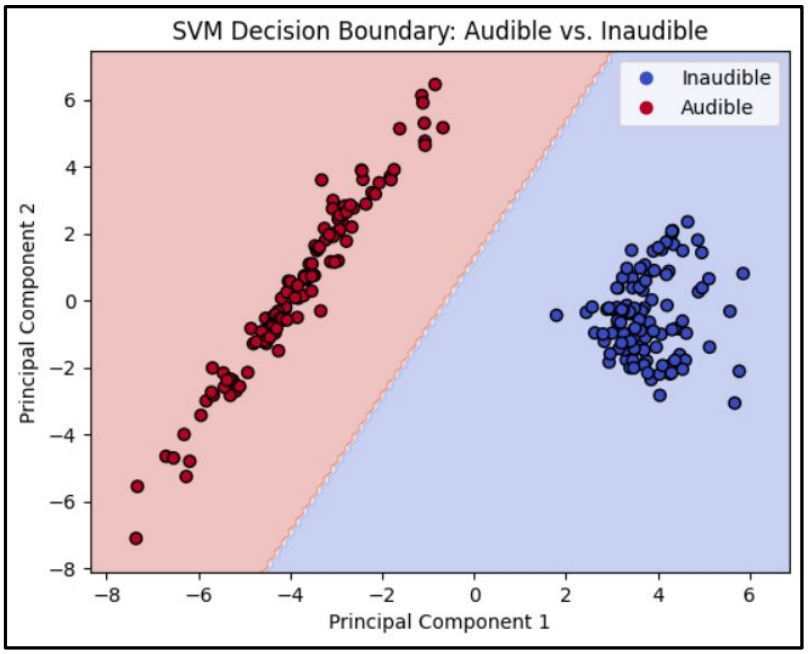

VALIDATION

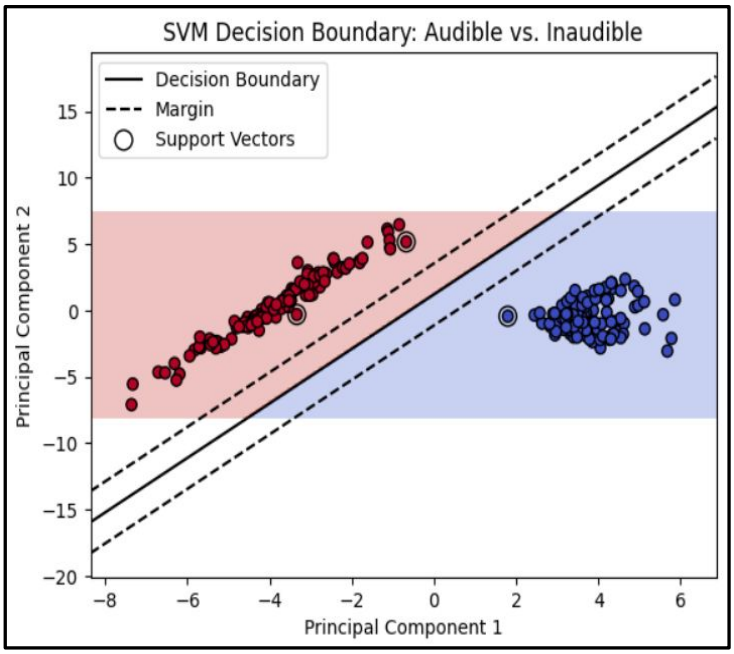
```
Model Accuracy: 100.00%

Confusion Matrix:
[[115   0]
 [  0 115]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       115
           1       1.00      1.00      1.00       115

    accuracy                           1.00       230
   macro avg       1.00      1.00      1.00       230
weighted avg       1.00      1.00      1.00       230
```

CONFUSION MATRIX



CLASSIFICATION





SUMMARY

Models	CVAE	CNN (One-Class)	SVM
Accuracy	95.65%	82.5%	100%
Training	Only on audible audios	Only on audible audios	On both audible and inaudible audios
Classification Basis	Reconstruction error threshold	Anomaly Score Threshold	Decision Boundary (Hyperplane)

MICROPHONE SIMULATION - Audible

[23]:
```
input_path = "/kaggle/input/testaudios/audio1.wav"
#input_path = "/kaggle/working/modulated_audio6.wav"
```

+ Code + Markdown

[26]:
```
microphone_simulation(input_path)
```

```
Reconstruction Error: 0.8393
Threshold: 1.0070

CLASSIFICATION: Audible (Normal)
Everything is good!
```

MICROPHONE SIMULATION - Inaudible

[27]:
```
#input_path = "/kaggle/input/testaudios/audio1.wav"
input_path = "/kaggle/working/modulated_audio6.wav"
```

[28]:
```
microphone_simulation(input_path)
```

```
Reconstruction Error: 1.6239
Threshold: 1.0070

CLASSIFICATION: Inaudible (Anomaly)
ALERT - Inaudible command detected!!
Recognized Command: OK Google restart my phone now
```


HARDWARE AND SOFTWARE REQUIREMENTS

HARDWARE:

- Intel Core i7/i9 CPU
- NVIDIA GPU P100
- 16GB/32GB RAM
- Windows 11, 64-bit OS

SOFTWARE:

- Python (Version 3.9 or Higher)
- Development Environment: Google Colab and Kaggle Notebook
- Python Libraries

SURVEY PAPER

SURVEY PAPER

Survey on Inaudible Voice Attacks : A Threat to Voice-Controlled Systems

Dr. Anita John[1†], Nandana S Pillai[1†], Ritu Maria[1†], Shaun Mammen John[1†], Shreya Veeraraghav[1†]

[1]Computer Science and Engineering, Rajagiri School of Engineering and Technology, Ernakulam, 682039, Kerala, India.

Contributing authors: anitaj@rajagiritech.edu.in; u2103147@rajagiri.edu.in; u2103176@rajagiri.edu.in; u2103194@rajagiri.edu.in; u2103198@rajagiri.edu.in;
[†]These authors contributed equally to this work.

Abstract

Inaudible voice attacks exploit vulnerabilities in voice-controlled systems (VCS) by using ultrasonic commands that humans cannot hear but devices can process. These attacks threaten user security and privacy by enabling unauthorized access and system manipulation. This survey examines the methods and impacts of attacks like DolphinAttack and SurfingAttack, showing how weaknesses in VCS hardware and software are exploited. It reviews many defense strategies such as hardware improvements, signal analysis, machine learning models, and authentication systems. While significant progress has been made, challenges remain, such as balancing security with usability and adapting to evolving attack methods. Future research, emphasizing the need for effective, practical solutions to secure VCS against these threats are the areas that are highlighted in this paper.

Keywords: Security, Privacy, Inaudible Voice Attacks, Non-Linearity

1 Introduction

The use of speech recognition software has grown in popularity recently. This has led to the emergence of voice-controllable systems, or VCS, such as Alexa and Siri. These days, people use this technology for a number of purposes, such as customer service, healthcare, and entertainment [1]. Famous author Arthur Clark popularized the idea of speech recognition systems in his 1968 book about Hal 9000, an artificial intelligence that could understand human speech. Despite the fact that few voice recognition software applications can match the HAL 9000's accuracy, speech recognition is already widely used and will only grow in popularity, as seen in hundreds of millions of smartphones and wearable technology [2].

Apart from their widespread use, these speech recognition systems are vulnerable to numerous attacks. These vulnerabilities give the attackers the chance to use voice commands without authorization, which may compromise the security and privacy of the user [3]. Attacks can be categorized as audible and inaudible. In audible voice attacks, the attackers cause security implications such as unauthorized access, data tampering, and service disruption in the form of voice commands, which can be understood by the speech recognition systems, but the humans perceive it as noise [2]. This type of attack is audible and can be caught. Nowadays, the attacks are more inaudible, so the user will not be able to hear the voice commands passed on by the attacker to the devices. Many researchers have formulated this concept and even proved it. This paper mainly focuses on the inaudible voice attacks, the defenses, and the future works that were put forward by the researchers in this domain.

2 Understanding Inaudible Voice Attacks

2.1 Definition and Characteristics

Inaudible voice attacks are a type of voice attack that primarily affect voice-activated systems like smart assistants, speakers, and autonomous systems. They exploit ultrasonic frequencies, or frequencies that are typically above the human hearing range (greater than 20 kHz), to issue malicious commands to these systems without the user's knowledge.

Characteristics of this attack include:

- The commands passed onto the systems are inaudible to humans, but they are interpreted as valid inputs.
- It allows silent execution of voice commands, making it difficult to detect by users.
- The main cause of this attack is the non-linearity of microphones. It exploits the hardware to demodulate (the process of converting a high-frequency or inaudible signal back into its original, i.e., audible form) ultrasonic signals for the system. Due to this accidental demodulation, the system considers it a genuine command and executes it.
- It does not rely on software bugs but it can exploit the hardware limitations of the devices.
- The effectiveness reduces with increasing distance or obstacles between the attack source and the target device.
- It often requires simple hardware like ultrasonic transducers and amplifiers.
- It allows silent execution of voice commands, making it difficult to detect by users.

WORK BREAKDOWN AND RESPONSIBILITIES

Task	Team Members
Data Collection	Nandana S Pillai, Ritu Maria Ajith, Shaun Mammen John and Shreya Veeraraghav
Audio Preprocessing	Shaun Mammen John and Shreya Veeraraghav
Convolutional VAE	Shreya Veeraraghav
One-Class CNN	Ritu Maria Ajith
SVM	Nandana S Pillai
Alert Generation	Nandana S Pillai and Shreya Veeraraghav
Command Recognition	Shaun Mammen John and Ritu Maria Ajith

GANTT CHART

RISKS AND CHALLENGES

- The working of the microphone is simulated using Python code due to obstacles in hardware design. The quality and processing of hardware equipment are not considered within the present model.

- As the model training is executed upon spectrogram images of audio samples, it is not possible to account for the speaker's distance from the device.

- Collecting diverse and sufficient data to train and validate the model, covering various environments, devices, and command variations to ensure robustness.

FUTURE WORK

- Integration of the developed system with a user-friendly application that delivers alerts to the users on their handheld devices in real-time along with the recognized commands.

- Implementation of Explainable AI (XAI) to determine the features learned by the model based on which it performs the classification process and subsequently using these specific features to refine the model training, focusing on the most relevant data.

CONCLUSION

- Detects inaudible command attacks by classifying audio as audible or inaudible.
- Alerts the user if malicious activity is detected.
- Identifies the specific inaudible commands used in the attack.

REFERENCES

1. Guoming Zhang, Chen Yan, Xiaoyu Ji, Tianchen Zhang, Taimin Zhang, and Wenyuan Xu. DolphinAttack: Inaudible voice commands. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pages 103–117, 2017.

2. Xinfeng Li, Xiaoyu Ji, Chen Yan, Chaohao Li, Yichen Li, Zhenning Zhang, and Wenyuan Xu. 2023. Learning Normality is Enough: A Software-based Mitigation against Inaudible Voice Attacks. In 32nd USENIX Security Symposium, USENIX Security 2023, Anaheim. 2455–2472

REFERENCES

3. Yitao He, Junyu Bian, Xinyu Tong, Zihui Qian, Wei Zhu, Xiaohua Tian, and Xinbing Wang. Canceling inaudible voice commands against voice control systems. In Proceedings of the 25th Annual International Conference on Mobile Computing and Networking, pages 1–15, 2019.

4. Nirupam Roy, Sheng Shen, Haitham Hassanieh, and Romit Roy Choudhury. Inaudible voice commands: The long-range attack and defense. In Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), pages 547–560, 2018.

5. Man Zhou, Zhan Qin, Xiu Lin, Shengshan Hu, Qian Wang, and Kui Ren.. *Hidden Voice Commands: Attacks and Defenses on the VCS of Autonomous Driving Cars*.

THANK YOU

Appendix B: Vision, Mission, Programme Outcomes and Course Outcomes

Vision, Mission, Programme Outcomes and Course Outcomes

Institute Vision

To evolve into a premier technological institution, moulding eminent professionals with creative minds, innovative ideas and sound practical skill, and to shape a future where technology works for the enrichment of mankind.

Institute Mission

To impart state-of-the-art knowledge to individuals in various technological disciplines and to inculcate in them a high degree of social consciousness and human values, thereby enabling them to face the challenges of life with courage and conviction.

Department Vision

To become a centre of excellence in Computer Science and Engineering, moulding professionals catering to the research and professional needs of national and international organizations.

Department Mission

To inspire and nurture students, with up-to-date knowledge in Computer Science and Engineering, ethics, team spirit, leadership abilities, innovation and creativity to come out with solutions meeting societal needs.

Programme Outcomes (PO)

Engineering Graduates will be able to:

1. Engineering Knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. Conduct investigations of complex problems: Use research-based knowledge including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. Modern Tool Usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. Individual and Team work: Function effectively as an individual, and as a member or leader in teams, and in multidisciplinary settings.

10. Communication: Communicate effectively with the engineering community and with society at large. Be able to comprehend and write effective reports documentation. Make effective presentations, and give and receive clear instructions.

11. Project management and finance: Demonstrate knowledge and understanding of engineering and management principles and apply these to one's own work, as a member and leader in a team. Manage projects in multidisciplinary environments.

12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change.

Programme Specific Outcomes (PSO)

A graduate of the Computer Science and Engineering Program will demonstrate:

PSO1: Computer Science Specific Skills
The ability to identify, analyze and design solutions for complex engineering problems in multidisciplinary areas by understanding the core principles and concepts of computer science and thereby engage in national grand challenges.

PSO2: Programming and Software Development Skills
The ability to acquire programming efficiency by designing algorithms and applying standard practices in software project development to deliver quality software products meeting the demands of the industry.

PSO3: Professional Skills
The ability to apply the fundamentals of computer science in competitive research and to develop innovative products to meet the societal needs thereby evolving as an eminent researcher and entrepreneur.

Course Outcomes (CO)

After the completion of the course the student will be able to:

Course Outcome 1: Model and solve real world problems by applying knowledge across domains (Cognitive knowledge level: Apply).

Course Outcome 2: Develop products, processes or technologies for sustainable and socially relevant applications (Cognitive knowledge level: Apply).

Course Outcome 3: Function effectively as an individual and as a leader in diverse teams and to comprehend and execute designated tasks (Cognitive knowledge level: Apply).

Course Outcome 4: Plan and execute tasks utilizing available resources within timelines, following ethical and professional norms (Cognitive knowledge level: Apply).

Course Outcome 5: Identify technology/research gaps and propose innovative/creative solutions (Cognitive knowledge level: Analyze).

Course Outcome 6: Organize and communicate technical and scientific findings effectively in written and oral forms (Cognitive knowledge level: Apply).

Appendix C: CO-PO-PSO Mapping

CO - PO Mapping

CO	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12
1	3	3	3	3	2	2	2					2
2	3		3	3	2	2	2					2
3									3	3		3
4					3	3		3	3		3	
5		3	3	3								3
6					3					3		3

CO - PSO Mapping

CO	PSO 1	PSO 2	PSO 3
1	2	2	2
2	3	3	2
3			2
4	1	1	1
5			2
6			2

Justification

Mapping	Justification
CO1 - PO1	Apply suitable computer science theory and mathematical foundations to identify relevant research areas
CO1 - PO2	Analyse the different problems and select relevant literature
CO1 - PO3	Create explainable content about the concepts to be conveyed to the audience
CO1 - PO4	Apply mathematical foundations and computer science theory to solve technical problems in the design of computer-based systems
CO1 - PO5	Proficiency in utilizing modern engineering tools and techniques to analyze and address real-world problems.
CO1 - PO6	Apply multidisciplinary knowledge in projects, utilizing suitable computer science theory and mathematical foundations in team-oriented tasks
CO1 - PO7	Identify solutions from literature that can lead to a sustainable solution
CO1 - PO12	Analyze the recent and relevant works done in their area of research to understand the methodologies
CO2 - PO1	Analyze academic documents using mathematical foundations and algorithm principles relevant to the modeling and design of computer-based systems.
CO2 - PO3	Evaluate and analyse computing requirements for technical programs in relevant literature chosen and understand them
CO2 - PO4	Identify the new technologies and modern engineering tools used and understand their working
CO2 - PO5	Use of a structured approach in solving complex computational and engineering problems.
CO2 - PO6	Apply comprehension from the literature to assess health, societal and legal issues and their relation to the engineering problems.
CO2 - PO7	Read and apply the context from the literature for sustainable development
CO2 - PO12	Ability to read and summarize the developments in engineering for lifelong learning process
CO3 - PO9	Team projects encourage independent thinking and lifelong learning.
CO3 - PO10	Present academic documents effectively, demonstrating communication skills
CO3 - PO12	Present the concepts and topics under study effectively

Mapping	Justification
CO4 - PO5	Give presentations about academic documents, applying modern tools and techniques
CO4 - PO6	Ability to execute tasks using available resources within time-lines to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
CO4 - PO8	Give presentations about academic documents keep up engineering norms and ethics
CO4 - PO9	Use of research-based knowledge to analyze and interpret experimental results.
CO4 - PO11	Engineering solutions addressing societal and environmental concerns.
CO5 - PO2	Enhances problem-solving for formulating technical solutions
CO5 - PO3	Develops skills in defining computing requirements and system design
CO5 - PO4	Prepare a technical report for experiments on complex problems
CO5 - PO12	Record the academic reports and concepts for life-long learning
CO6 - PO5	Development of systematic approaches for designing and testing solutions.
CO6 - PO10	Designing engineering components and systems to meet specific requirements.
CO6 - PO12	Ethical responsibility and professionalism in engineering practices.
CO1-PSO1	Students will be able to apply their skills to identify, analyze, and formulate solutions for complex, multidisciplinary engineering problems through the strategic use of academic literature.
CO1-PSO2	Students will be able to navigate academic literature effectively while applying programming concepts, thereby preparing them for more advanced learning and practical applications in the field of computer science.
CO1-PSO3	Students will demonstrate their ability to interpret and utilize fundamental computer science concepts to conduct competitive research through academic literature.
CO2-PSO1	Students will be able to evaluate and design solutions for complex engineering problems in multidisciplinary areas by critically analyzing and interpreting academic documents from the literature.
CO2-PSO2	Students will be able to locate and utilize academic literature effectively while applying programming concepts, thereby equipping them for further learning and practical applications in the field of computer science.
CO2-PSO3	Students will have the ability to understand and apply the fundamentals of computer science in competitive research by interpreting and utilizing academic documents from the literature.

Mapping	Justification
CO4 - PSO1	Identifying, analysing and designing solutions that are ethical and professional.
CO4 - PSO2	Acquiring programming efficiency to deliver quality software utlizing available resources within the time-line.
CO4 - PSO3	Effective planning and scheduling lead to better project management.
CO5 - PSO3	Apply computer science knowledge to resolve research/technology gaps and propose innovative solutions.
CO6 - PSO3	Communication and documentation of technical findings enhance professional growth.
CO3 - PSO1	Students will be able to identify, analyze, and develop solutions for complex engineering problems across multidisciplinary areas while creating a presentation on an academic document
CO3 - PSO2	Students will demonstrate the ability to apply the fundamentals of computer science in competitive research by designing a presentation based on an academic document
CO4 - PSO1	Students will be able to to identify, analyze and design solutions for complex engineering problems in multidisciplinary areas while giving a presentation about an academic document
CO4 - PSO3	Students will have the ability to apply the fundamentals of computer science in competitive research while giving a presentation about an academic document
CO5 - PSO1	Students will be able to to identify, analyze and design solutions for complex engineering problems in multidisciplinary areas while preparing a technical report