A SOUND MINIMIZATION SYSTEM FOR ENHANCED INDOOR ACOUSTICS

by

Syed Labib Al-Barr
21321024
Mehdi Sarker
21321081
Mansiv Hamid Shangram
21321032
Md. Talha Bin Imran
21221018

A Final Year Design Project (FYDP) submitted to the Department of Electrical and Electronic Engineering in partial fulfillment of the requirements for the degree of B.Sc. in Electrical and Electronic Engineering (EEE)

Academic Technical Committee (ATC) Panel Member:

Dr. Rony Kumer Saha (Chair)
Associate Professor, Department of EEE, BRAC University

Bristy Das
Lecturer, Department of EEE, BRAC University

Md. Muhiul Islam Muhiuddun
Lecturer, Department of EEE, BRAC University

Department of Electrical and Electronic Engineering
Brac University
September 2025

Declaration

It is hereby declared that

1. The FYDP submitted is our own original work while completing degree at Brac University.
2. The FYDP does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The FYDP does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

Syed Labib Al-Barr 21321024	**Mehdi Sarker** 21321081
Mansiv Hamid Shangram 21321032	**Talha Bin Imran** 21221018

Approval

The FYDP titled "A SOUND MINIMIZATION SYSTEM FOR ENHANCED INDOOR ACOUSTICS Final Year Design Project Title" submitted by

1. Syed Labib Al-Barr (21321024)
2. Mehdi Sarker (21321081)
3. Mansiv Hamid Shangram (21321032)
4. Talha Bin Imran (21221018)

in the summer of 2025, has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in EEE on 11/09/2025.

Examining Committee:

Academic Technical Committee (ATC):
(Chair)

_____
Dr. Rony Kumer Saha, PhD
Associate Professor, Department of EEE
Brac University

Final Year Design Project Coordination Committee:
(Chair)

_____
Mirza Rasheduzzaman, PhD
Associate Professor, Department of EEE
Brac University

Department Chair:

_____
Md. Mosaddequr Rahman, PhD
Chairperson, Department of EEE
Brac University

Ethics Statement

We declare that the following in relation to the preparation of this work:

1. The material is solely the original work of the named authors prepared under the supervision of the mentioned ATC panel members.
2. All sources used as reference have been appropriately credited and cited.
3. We have strictly adhered to the Brac University Plagiarism Policy.
4. The reported similarity index of this work is 15%.

Abstract

Indoor noise pollution has become a critical public health concern particularly due to rapid urbanization. However, existing Passive Noise Control (PNC) solutions inherently compromise Indoor Environmental Quality (IEQ) by impeding air flow and blocking sunlight, while the traditional Active Noise Control (ANC) methods remain cost-prohibitive for widespread deployment. This project introduces an indoor sound minimization system that utilizes a Machine Learning-based ANC framework. The system integrates microphones, speakers and microcontrollers with a Convolutional Recurrent Network (CRN) algorithm to predict and generate effective real-time anti-noise signals. Performance evaluation of a working prototype in diverse acoustic environments demonstrates an average noise reduction of 9.4 decibel (dB) in the frequency range of 20 to 2000 Hertz (Hz), achieved within 80 milliseconds (ms). The system prioritizes scalability and low power consumption, positioning it as a potentially viable and sustainable acoustic solution for residential homes and healthcare facilities.

Keywords: Active Noise Control; Convolutional Recurrent Network; Real-time Predictive Anti-Noise Generation; Indoor Environmental Quality.

Acknowledgement

We would like to express our sincere gratitude to our supervisors — Dr. Rony Kumer Saha (Chair), Ms. Bristy Das, and Mr. Muhiul Islam Muhiuddin — for guiding us with constructive criticism and encouragement. Their supervision and inspiration have made this work possible. We would also like to extend our heartfelt gratitude to FYDP committee members, whose invaluable comments and suggestions have contributed to the refinement and successful completion of this project.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AC	Alternating Current
ADC	Analog-to-Digital Converter
ANC	Active Noise Control
ARN	Attention Recurrent Network
ATC	Academic Technical Committee
B/C	Benefit-Cost Ratio
BDT	Bangladeshi Taka
BoM	Bill of Materials
CO	Course Outcome
CRN	Convolutional Recurrent Network
DAC	Digital-to-Analog Converter
dB	Decibel
DSP	Digital Signal Processor
EEE	Electrical and Electronic Engineering
FAM	Fused Add-Multiply
FxLMS	Filtered-x Least Mean Square
FYDP	Final Year Design Project
Hz	Hertz
I2C	Inter-Integrated Circuit
I2S	Inter-IC Sound
IEQ	Indoor Environmental Quality
IRR	Internal Rate of Return
ISTFT	Inverse Short-Time Fourier Transform
LMS	Least Mean Square
MDF	Medium-Density Fibreboard
ML	Machine Learning
ms	Milliseconds
PNC	Passive Noise Control
PNW	Present Net Worth
PO	Program Outcome
RAM	Random Access Memory
RIR	Room Impulse Response
RoHS	Restriction of Hazardous Substances
SFANC	Selective Fixed-Filter Active Noise Control
STFT	Short-Time Fourier Transform
THD	Total Harmonic Distortion
WHO	World Health Organization

Chapter 1: Introduction [CO1, CO2, CO10]

1.1 Introduction

Noise pollution remains a pervasive yet often overlooked threat to public health and well-being, particularly in densely populated urban environments. The World Health Organization identifies environmental noise as a significant contributor to a range of adverse health outcomes, including sleep disturbance, cardiovascular disease, cognitive impairment, and increased stress levels. In rapidly urbanizing regions such as South Asia, where cities like Dhaka face unregulated construction, dense traffic, and industrial activity, noise levels frequently exceed safe limits, placing millions at risk. Existing noise control solutions are either costly, ineffective in dynamic environments, or too resource-intensive for everyday use. Therefore, there is a pressing need for an affordable, real-time, and efficient noise mitigation system suitable for indoor environments.

1.1.1 Problem Statement

Noise pollution is a growing concern worldwide, with low-frequency components (200–1000 Hz) of urban noise being particularly disruptive to indoor environments. The expansion of infrastructure, such as the metro-rail, airports, and highways on top of rising population densities have led to an increase in noise pollution. Traditional passive noise control measures such as the use of dense soundproofing materials and specialized acoustic windows, disrupt the interior environment by obstructing air flow and natural light. They have also proven inadequate for mitigating low-frequency noise, which penetrate indoor environments more easily. This disruption negatively affects productivity, concentration, and well-being in homes, classrooms, and offices. Therefore, there is an urgent need for scalable, real-time ANC solutions to address this challenge.

1.1.2 Background Study

Noise pollution is increasingly recognized as a critical public health concern, particularly in densely populated urban environments like Dhaka city. According to Jahan, different zones of Dhaka City constantly exceed the maximum noise level deemed safe by the World Health Organization (WHO) and highlights a wide range of adverse effects including hearing loss, sleep disruption, and cardiovascular stress that occurs as result of long exposure to high noise levels from traffic and urban activity [1].

Ma J. et al. [2] attempted to examine the correlation between diversified noise pollution and urban residents' mental health symptoms using Bayesian Multilevel Logistic models, establishing a significant connection between long time noise exposure to increased probability of chronic anxiety, stress, headache, and fatigue.

Peng et al. [3] highlights the ineffectiveness of the traditional passive noise control method (PNC) in dealing with low-frequency noises and propose a membrane-type acoustic metamaterial optimized using a Kriging surrogate model, offering improved control over low-frequency vibrations.

Lam et al. [4] suggests Active Noise Control (ANC) as a leading solution to mitigating noise pollution in urban environments, highlighting its improved performance in low-frequency regions over passive noise control (PNC) techniques. The paper also points out that due to ANC making use of destructive interference generated by anti-noise signals to cancel out unwanted sound waves, ANC systems can theoretically perform cancellation even with windows and other apertures kept open.

Elliot and Nelson [5], in 1990, generalized the LMS adaptive algorithm to produce a multichannel active noise cancellation zone, which successfully mitigated low-frequency noise in aircraft cabins and inside cars.

With the advent of increased processing power, some novel approaches have been proposed to better utilize and adapt the ANC approach against a larger range of frequencies in real-time. One such approach attempted by Li and Zhang [6] was to employ an adaptive parametric array of loudspeakers to counter environmental noise, offering more precision and directional control.

The most recent trend in the active noise cancellation approach has been to evaluate the performance of pre-trained Machine Learning (ML) models to generate effective cancellation signals in real time. Zhang and Wang [7], proposes a model for multichannel implementation, analyzing its theoretical performance against a range of frequencies and noise types.

Collectively, this background research indicates that there is indeed a demand for an affordable and effective low-frequency noise mitigation system and continuous research across multiple fields over the last few decades have prepared the ground for such an endeavor.

1.1.3 Literature Gap

Extensive literature review was undertaken to identify the gap that exists within literature, which is currently hindering the development of a sound cancellation device for mass consumers. A number of literature gaps were identified.

Firstly, the existing studies on noise profiles in South Asian cities lack data regarding the spectral content of the measured noise [1]. While the average loudness, measured in decibel, is a good indicator of the severity of sound pollution and subsequent impact on health, noise level data alone is not a sufficient starting point for ANC implementation. Acoustic control of urban environments requires a detailed analysis of the dominant audio frequency components that arrive through the building openings into the indoor environment.

Secondly, the ANC systems designed and prototyped by Lam et al. [4] requires a high-end processor to handle the increasing complexities posed when transitioning from single channel to multichannel systems. Although it achieved significant reduction in indoor noise level both with and without passive obstruction, the required high-end processor lowers its market-compatibility. Furthermore, the paper neither proposes further optimization to the Filtered-x Least Mean Square (FxLMS) algorithm nor offers an implementation procedure compatible with low-end processors. In addition, the system lacks adaptability to high frequency noises, compromising reliability in loud, changing environments.

Finally, in order to bypass the high computational complexity of adaptive filters in multichannel systems, a pre-trained fixed filter method leveraging Machine Learning techniques is proposed by Zhan and Wang [7]. They suggested the CRN model due to its capability to detect both spatial patterns and handle sequential data. The convolutional recurrent network, despite showing promising results in offline tests when compared with outputs from FxLMS, Selective Fixed-Filter Active Noise Control (SFANC) or Least Mean Square (LMS) - based systems, suffers from slow convergence and high latency, hurting real-time performance. The paper evaluates the model's performance but does not advance to hardware implementation.

To summarise, three major gaps exist in current research. Firstly, existing noise profile research performed for South Asian cities lacks spectral information, limiting the extent to which they can be useful in noise reduction strategies and research work. Secondly, the multi-channel adaptive filter system proposed by Lam et. al relies on powerful processors for practical use despite achieving high noise reduction, which undermines its reach in the consumer-market. Finally, the machine learning based noise mitigation strategies currently suffer from slow convergence hurting real-time performance, necessitating modifications of proposed algorithms for future hardware implementation.

1.1.4 Relevance to Current and Future Industries

The development of an affordable, real-time indoor noise mitigation system holds strong relevance for both current and future industries, particularly in the context of growing urbanization and rising concerns over environmental noise pollution. As cities become denser and more industrialized, the demand for efficient noise control solutions in homes, offices, hospitals, and educational institutions continues to grow. However, existing ANC systems are either expensive or not optimized for real-time performance, limiting their market accessibility.

By leveraging machine learning to reduce hardware requirements and enhance responsiveness, this project introduces a scalable alternative suitable for mass-market adoption. This aligns with the needs of consumer electronics, smart home technology, and healthcare sectors that increasingly prioritize wellness-focused innovations. Additionally, the solution has potential for integration into future smart building infrastructure, contributing to sustainable urban living. Its affordability also makes it particularly suitable for deployment in developing countries, addressing both public health concerns and economic constraints.

1.2 Objectives, Specifications, Requirements and Constraints

This project intends to address the identified gaps in *Section 1.1.3* by developing a cost-effective, scalable ANC system tailored for residential and commercial spaces.

1.2.1 Objectives

The aim of the project is to design a system with the following objectives:

1. Attenuate the transmission of broadband external noise into the indoor spaces
2. Perform real-time active acoustic control with minimum processing latency

1.2.2 Functional Requirements

1. Real-time detection and processing of incoming sound signals
The system must t be capable of detecting and processing incoming sound signals in real-time, with minimal latency between noise detection and anti-noise generation.

2. Anti-noise wave generation
The system must generate anti-noise signals that are equal in amplitude but opposite in phase to incoming external noise, creating destructive interference at the center of the quiet zone.

3. Low-frequency noise mitigation
The system must significantly reduce the amplitude of frequency components within the range of 200 to 2000Hz.

4. Scalability
The system must be adaptable for different infrastructures, ranging from small residential homes to large institutional setups.

5. Energy efficiency
The system must operate on minimal power, with an average consumption comparable to typical household electrical appliances.

1.2.3 Non-functional Requirements

1. **Requires minimal to no structural modification**
   The system must utilize a modular design, requiring no major structural changes to the window openings.

2. **Intuitive user interface**
   The system must be controllable with an intuitive graphical interface that allows adjustment to the preferred noise reduction levels.

3. **Highly durable with minimal maintenance**
   The system must minimize maintenance through a set of dust and humidity protection features for the external components.

4. **Affordable**
   The system should be an economically viable alternative to traditional soundproofing methods, ensuring accessibility for diverse economic demographics.

1.2.4 Specifications

Table 1.1: Specifications of the project

Level	**Requirements**	**Specifications**
System Level	Real-time noise cancellation	8 dB attenuation (200 – 2000 Hz); maximum processing latency < 100ms
	Power Supply	220 V AC input; 50 W rated power (max)
Component Level	Controller	Clock speed > 1.5 GHz; 8 GB RAM; I2C and I2S protocol support
	Speakers	Frequency Response: 20 Hz to 15 kHz; Total Harmonic Distortion (THD) < 5%
	Microphones	Frequency Response: 0.5 Hz to 20 kHz; Sensitivity: 50 mV/Pa

1.2.5 Constraints in Design Process

1. **Dependency on site-specific acoustic characteristics**
   The system requires a time consuming and detailed noise profiling of the site before installation to accurately map the intensity gradients and directionality of the incoming noise. Performance dependency on site-specific calibration prevents scalability.

2. **Electro-acoustic geometric limitation causes visual hindrance**
   Loudspeakers with large diaphragm are necessary for effective cancellation of low-frequency noise components, which conflicts with the architectural requirement of minimal visual obstruction at the window aperture.

3. **Trade-off between durability and economic feasibility**
   Selection of industrial-grade, weather-resistant components to ensure durability is challenged due to the strict limit imposed on the Bill of Materials (BoM) to maintain affordability**.**

4. **Social acceptance and privacy perception**
   Integration of microphones in residential spaces present possible privacy concern from the users, leading to strictly local non-retentive signal processing.

1.3 Systemic Overview

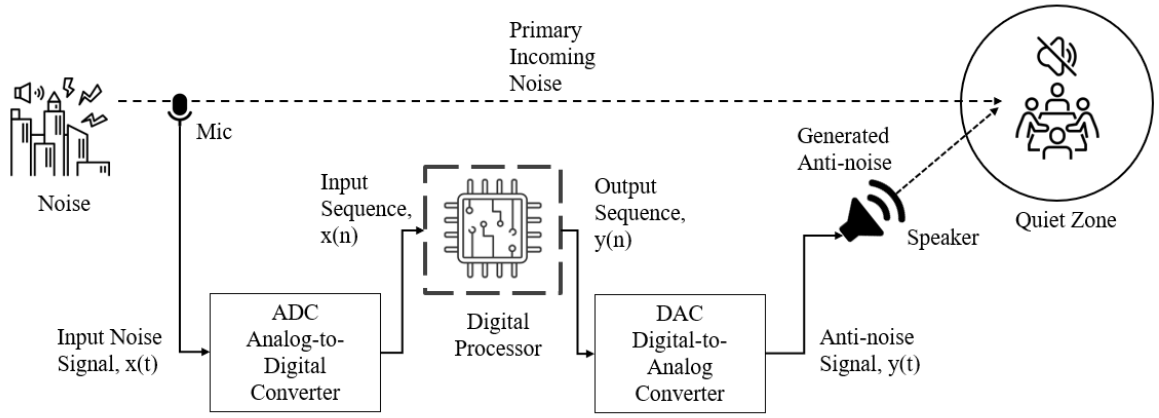


Fig 1.1: Systemic overview of the proposed ANC project

The system utilizes a reference microphone to capture incoming noise signals from the environment. The signal, x(t), detected by the mic is first converted into digital form using an Analog-to-Digital Converter (ADC), resulting in a discrete-time signal, x(n). The signal is subsequently processed by a digital processor that accounts for the change in incoming signal due to the propagation path, to generate a suitable predicted anti-noise sequence, y(n). The predicted digital anti-noise signal is converted back into analog form using a Digital-to-Analog Converter (DAC), producing the analog anti-noise signal, y(t). Finally, y(t) is played back through a speaker, which interferes destructively with the original noise to form a quiet zone, effectively reducing perceived noise in real-time.

1.4 Conclusion

This chapter highlights the growing need for a cost-effective, scalable, real-time acoustic control system to increase the IEQ of residential and commercial units in noisy urban regions.

The ineffectiveness of traditional passive soundproofing methods — due to their limitation against low-frequency components as well as obstruction to sunlight and ventilation —justifies the investigation into ANC as an alternative.

An overview of the foundational work in ANC and an analysis of the recent advances outline the potential scope of this project, specifically the addition of spectral information to existing noise profile data of South Asian cities, reduction in computational complexity of multi-channel adaptive ANC approach, and modification of the proposed CRN algorithm to address slow convergence. The objective of this project is to design, develop, and validate a prototype based on the ANC principle, that addresses the identified research gaps.

The proposed ANC system prioritizes both affordability and functionality to overcome the limitations of existing solutions. The functional and non-functional requirements, as well as the constraints were carefully selected in the context of the socio-economic constraints and acoustic requirements of the target demographic.

Chapter 2: Project Design Approach [CO5, CO6]

2.1 Introduction

The development of an effective ANC approach requires a rigorous evaluation of the existing control strategies that addresses the effect of the propagation paths on the incoming signal. Since the control strategies are being implemented at the receiver end rather than the source, each distinct propagation path and its corresponding transfer function must be considered to ensure the generated signal received at the center of the quiet zone is equal in magnitude and opposite in phase to the incoming signal at the same point.

Implementing he process presents several key challenges. Firstly, the incoming environmental noise is stochastic and time-variant, due to time-varying angles of arrival and ranging across a large frequency band. Secondly, the path between the primary microphone and the region where interference is set to occur, causes the incoming input signal to alter its characteristics such that combining it with the generated anti-noise does not readily results in destructive interference. Thirdly, the delay introduced due to the conversion between digital and analog, as well as the path between secondary loudspeaker and the target zone prevents effective cancellation. Lastly, the geometry of the room enclosure, placement of the objects within, and the nature of its surfaces significantly impact the extent and stability of the quiet zone.

An effective ANC approach must address the stated challenges, while being computationally efficient enough to be implemented on resource-constrained hardware.

2.2 Identify Multiple Design Approach

We propose two design approaches that seek to address these challenges while attaining system objectives.

The first approach is based on the idea of adaptive active noise control where the anti-noise is generated using a control filter. The coefficients of the control filter are adjusted using a feedforward algorithm [5]. The control filter output superposes with the incoming noise and results in destructive interference. The residual signal, which is the result of the combination of incoming noise and filter output, is used to update the filter coefficients such that the residual signal is minimized. This approach requires two sets of microphones for each audio channel and becomes computationally complex once multiple channels are introduced. This leads to higher power consumption. Moreover, the processing latency of this approach is high due to its reliance on error correction.

To reduce the real-time computational complexity during operation, as well as the number of audio peripherals used per channel, an alternate design approach is considered. In this second approach, a pre-trained ML model is implemented using a microcontroller to predict the anti-noise for a variety of noise types that the system is likely to encounter. The ML model matches

the input noise spectrogram to its corresponding anti-noise spectrogram accounting for the path effects. Subsequently, the model output is converted to time domain and played back through the speaker. This approach requires a large dataset consisting of a variety of urban and environmental sound during model training as well as substantial processing power during inference, resulting in reduced latency.

Although the objectives of both the approach are the same and both primarily work on the principle of superposition to achieve ANC, they employ different means and present different technical challenges and stages towards implementation.

2.3 Describe Multiple Design Approach

2.3.1 Adaptive ANC using FxLMS

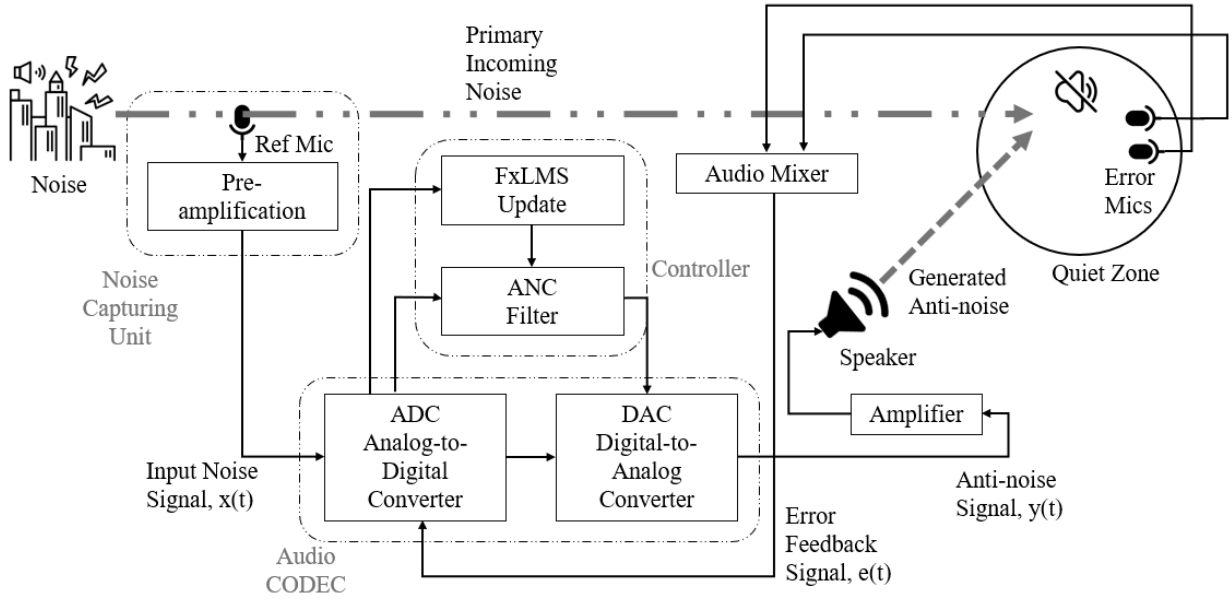


Fig 2.1: Adaptive ANC module based on FxLMS algorithm

In this design, we implement a modular single-channel ANC unit using the FxLMS algorithm, driven by an embedded microcontroller.

In the ANC module, a reference microphone placed outside the enclosure captures the incoming noise from the external environment. The resulting analog signal is sampled into a digital signal that is passed to a microcontroller capable of implementing DSP processes. The microcontroller is programmed with a control filter that outputs the anti-noise digital signal. In addition, the control filter contains an estimate of the secondary path, generated by playing white noise from the speaker and processing the received error signal during the program initiation stage. The coefficients of the control filter are adjusted at every iteration using the FxLMS algorithm, which attempts to minimize the error signal.

The output from the controller is the digital anti-noise signal, y(n), which is converted back to

analog and played through the loudspeaker. Based on the principle of superposition, the two waves destructively interfere, resulting in an attenuated noise signal.

Having multiple ANC modules ensures the system is able to detect and respond to variations in noise gradients and directionality [4].

2.3.1 Predictive ANC system based on ML model

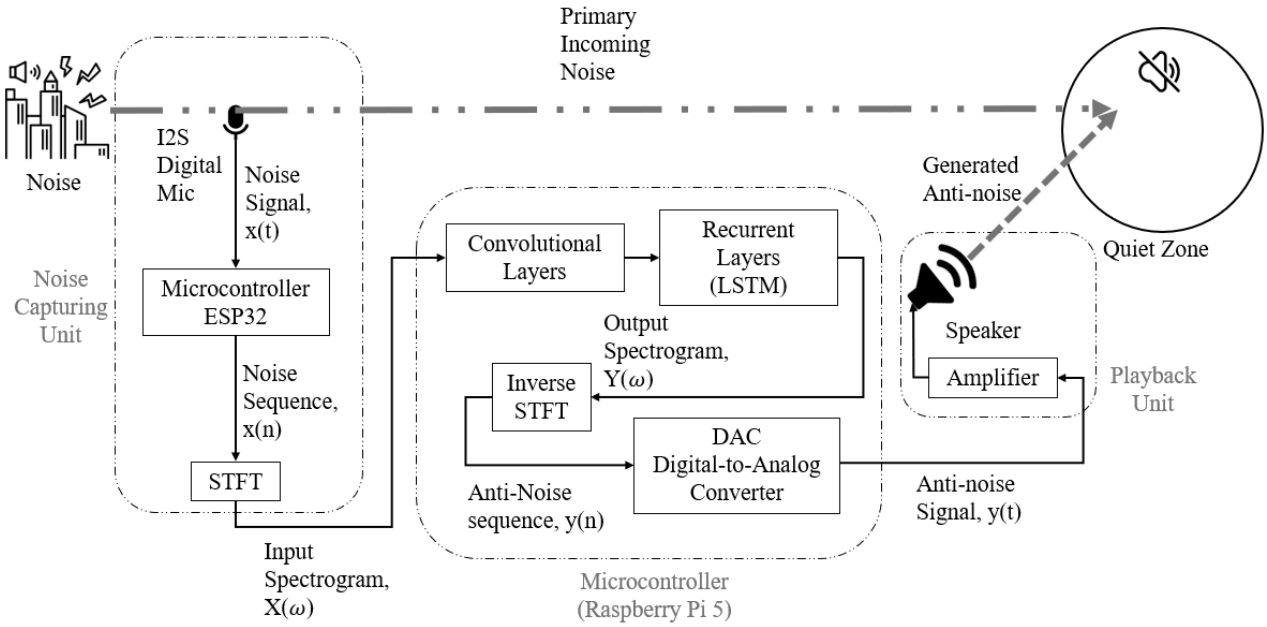


Fig 2.2: Predictive ANC system based on the CRN model

In this approach, the focus is primarily on training and implementing an ML model that can predict the anti-noise, given an input noise signal. The drawback of traditional multi-channel systems with central processing is the computational burden placed on the DSP processor, which in turn increases the cost of the system. The main computational cost comes from multiple feedback paths attempting to update the filter coefficients [6].

We bypass this computational bottleneck by implementing a pre-trained model that can accurately match the input noise frequency bins to its corresponding anti-noise frequency bins, as well as account for the effects of primary and secondary paths [7].

The model operates in the frequency domain, hence the input signal captured by the reference microphones are first converted to digital signal using an ADC, and then transformed to frequency domain by performing Short-Time Fourier Transform (STFT) on the digital sequence. The output of the STFT block is a two-dimensional array of frequency and time, which reveals the magnitude of each frequency component present in a specific time interval, along with its phase information. The model output is the frequency bins of the predicted anti-noise, which is transformed back to time-domain and played back through the loudspeaker.

This method seeks to simplify the circuitry by eliminating the error microphones from the

design. The delay-adjustment is done on the principle that the delay due to the secondary path remains fairly constant due to the speakers being a fixed distance away from the quiet zone.

2.4 Analysis of Multiple Design Approach

This section analyses the mathematical and signal processing theories behind the two design approaches discussed in *Section 2.3.*

2.4.1 Adaptive ANC using FxLMS

Implementation of the adaptive ANC module involves the integration of the control filter, W(z), and the FxLMS algorithm such that the error signal is attenuated the at the quiet zone.

Designing the Control Filter

The control filter employs a feedforward filter, where the input signal x(n) is passed through parallel delays and weights, such that each element of the x(n) sequence is altered by a certain weight [8]. The output signal, y(n), is the summation of the products of delays and weights.

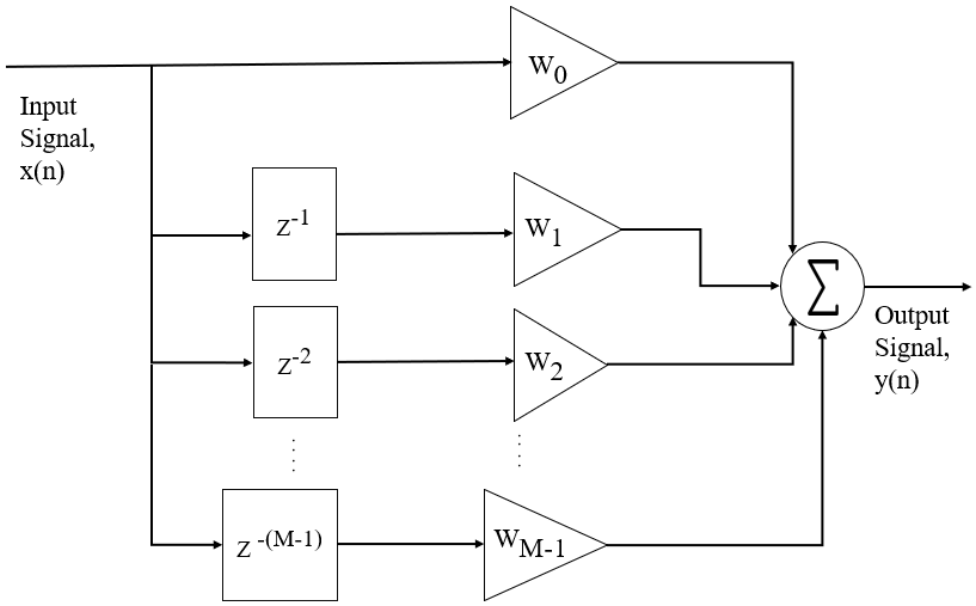


Fig. 2.3: Feedforward adaptive control filter

The output signal y(n) can be described using the following equation,

$$y(n) = \sum_{0}^{M-1} w_i(n) \cdot x(n-i) \quad (1)$$

where:

- y(n) is the generated anti-noise signal sequence
- $w_i(n)$ is the i-th coefficient weight of the adaptive filter at time n
- (n-i) is the input reference signal delayed by i samples
- M is the filter length

Designing the FxLMS Algorithm

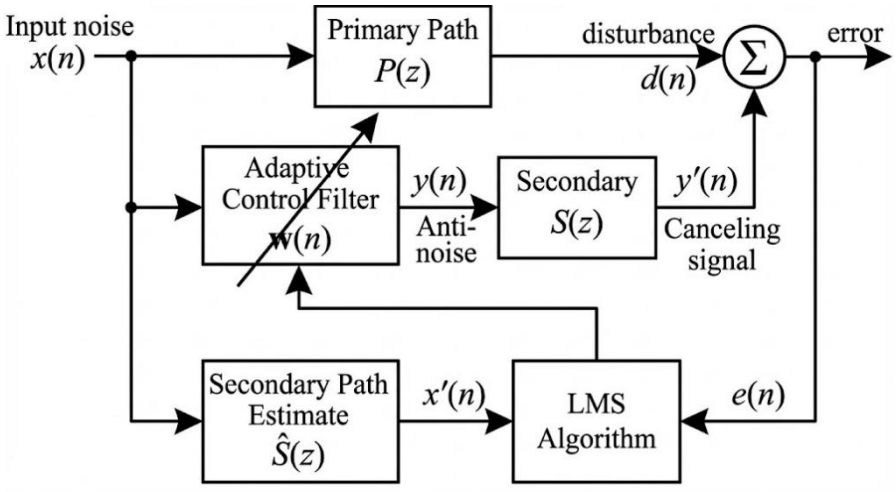


Fig. 2.4: FxLMS algorithm

While the control filter is generating the anti-noise signal from input noise, x(n), the FxLMS algorithm ensures the generated anti-noise completely cancels the input noise by measuring the error signal at each continuous iteration and updating the control filter weights, w(n) to minimize error signal, e(n).

However, the generated anti-noise signal, $y(n)$, also travels through the secondary path, S(z), which transforms it into distorted input signal $y'(n)$,

$$y'(n) = s(n) * y(n) \tag{2}$$

where:

- s(n) is the impulse response of the secondary path

To ensure the weights updating algorithm also takes into account the secondary path, the input signal, x(n) is transformed to $\hat{x}(n)$ by an estimate of the secondary path, $\hat{S}(z)$, before feeding it to the LMS algorithm responsible for minimizing error.

$$\hat{x}(n) = \hat{s}(n) * x(n) \tag{3}$$

where:

- $\hat{s}$(n) is the impulse response of the estimated secondary path

The error signal is calculated as,

$$e(n) = d(n) - y'(n) \tag{4}$$

where:

- $d(n)$ is the distorted input signal
- y'(n) is the distorted output anti-noise signal.

In the LMS algorithm, the mean squared error, $e^2(n)$, is the cost function. In order to minimize the cost function, the gradient of the squared error with respect to the filter weights is calculated,

$$\hat{\nabla}(n) = \frac{\partial\left(e^2(n)\right)}{\partial(w(n))} = -2e(n)\hat{x}(n) \quad (5)$$

The filter weight coefficients are updated iteratively using the method of steepest descent,

$$w(n+1) = w(n) - \frac{\mu}{2}\left(\hat{\nabla}(n)\right) \quad (6)$$

where:

- μ is the step size
- $w(n+1)$ and $w(n)$ are successive coefficients of filter weights

Due to the weights being continuously updated, the adaptive ANC module is capable of responding to diverse acoustic environments, making it a suitable candidate for implementation in stochastic urban soundscapes.

2.4.2 Predictive ANC System Based on ML Model

The implementation of the ML-based predictive ANC design can be achieved through three key stages:

1. Collection and preparation of training data
2. Development of the CRN model
3. Hardware Integration

The data collection stage primarily focuses on transforming the relevant urban environmental and speech audio signals into corresponding numerical arrays on which CRN model can be trained.

The training stage is dedicated to fine-tuning the algorithm based on which the CRN model is trained such that the model is compact, requires minimal processing time, and scalable to multi-channels.

In the hardware integration stage, the CRN model is implemented on low-end processors for local inference, and the data pipeline starting with the sensing of noise signal to playing pack the anti-noise for cancellation is ensured.

Training Data Collection and Preparation

The available audio collection, as well as the local field recordings are analog signals that need to be cleaned, normalized and modified into numerical dataset of input and output spectral

information with appropriate label for each category, before it can be dispatched for training.

The data preparation process is outlined in the following process flow diagram.

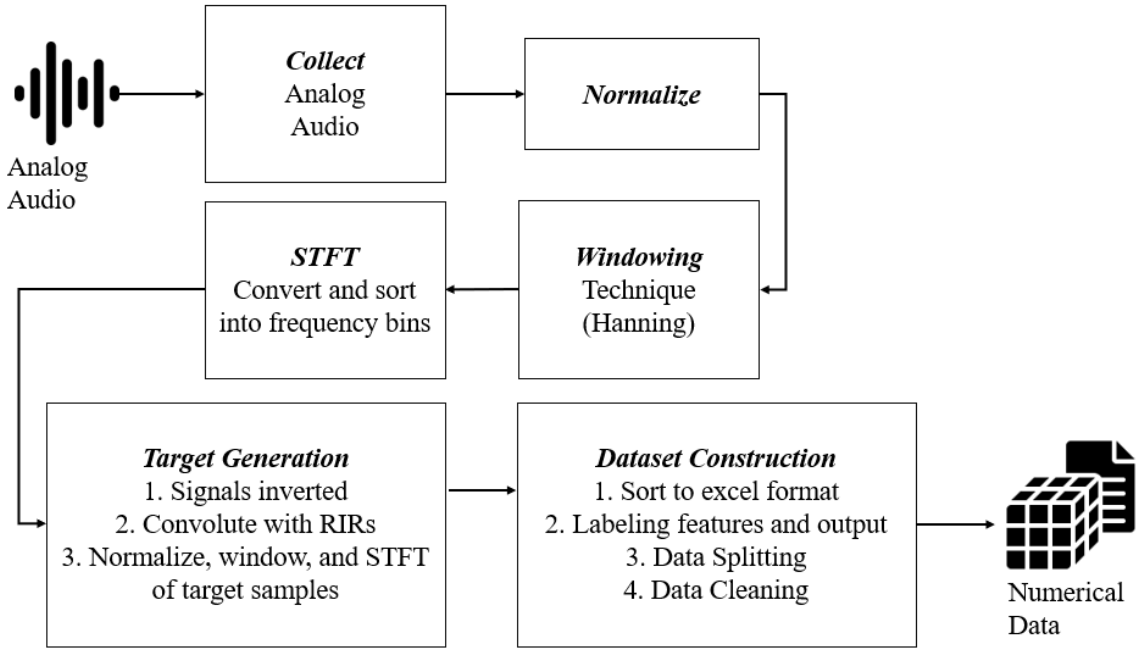


Figure 2.5: Process flow diagram of the data collection Process

Audio samples can be collected from public dataset, field recordings and synthesized audio for special sounds. Each sample is normalized to maintain consistent amplitude level and windowed to prepare the audio data for conversion into frequency domain. For each sample, corresponding anti-noise is generated by convoluting with Room Impulse Responses (RIR) and then inverting the signal. The RIRs are generated by taking into account the enclosure dimension, texture of walls, and the location of major geometric obstacles in the sound path. The RIRs are site-specific and simulate the effects of primary and secondary paths. The domain conversion is then applied simultaneously to input samples and anti-noise samples. Finally, the frequency and phase information are categorized into a numerical dataset, whose dimensions are compatible with the CRN model training algorithm.

Development of the CRN Model:

In the training stage, the model architecture is defined and the model is developed using an algorithm based on the data acquired and processed during the collection stage. The data is first split into training, testing, and validation batches. An experimental setup is implemented with variable configurations of reference microphones, secondary loudspeakers and error microphones.

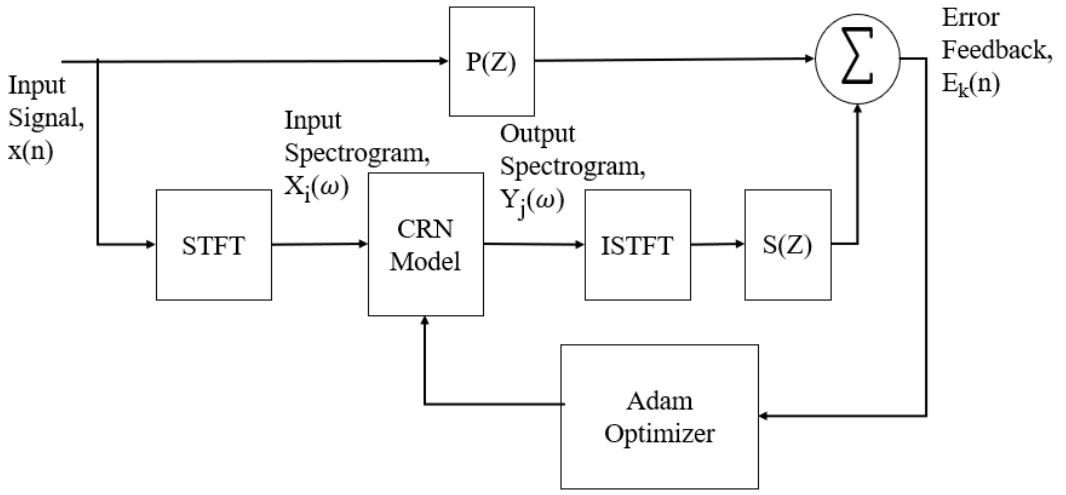


Figure 2.6: Block diagram of the CRN-based ANC training architecture

First, we design the CRN architecture. The CRN model follows the encoder-decoder architecture with additional recurrent layers in-between to detect sequential pattern. The encoder-decoder enables essential spatial features to be extracted. Designing the architecture involves choosing the number of convolutional and recurrent layers, activation function in each layer, the values for hyperparameter, and the learning rate.

Secondly, the training strategy illustrated in the *figure 2.6* is implemented. The input noise spectrogram $X_i(\omega)$ is given as input to the CRN architecture and the output $Y_j(\omega)$ is received. The spectrogram output is converted to time-domain and transformed through known secondary path estimate, which produces the path-compensated anti-noise signal. The difference between input and output noise signal for each channel, the error signal, is squared and the Mean Squared Error (MSE) is used as the loss function. The weights of the CRN model are adjusted using an optimizer algorithm and the loss function, and the final weights are stored for each layer. Finally, the training process is repeated with different configurations of microphones, and loudspeakers to enhance the adaptability of the model.

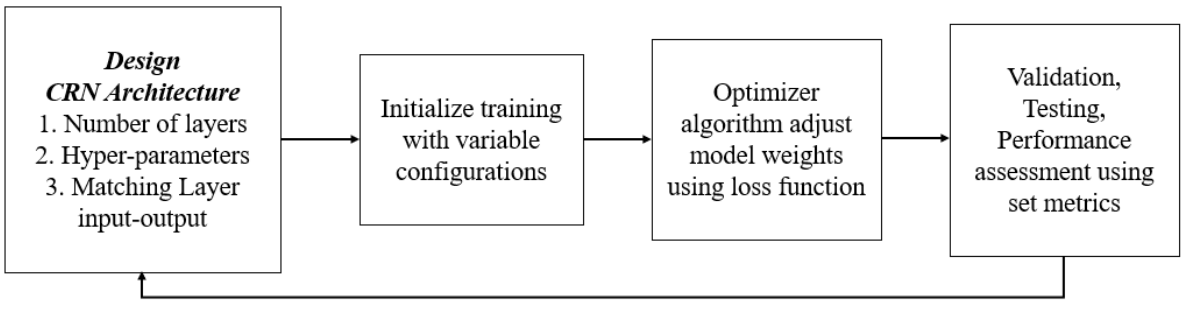


Figure 2.7: Process flow of CRN model Training

2.5 Conclusion

The first design approach, the adaptive ANC module, generates localized quiet zones using independent single-channel modules with microcontrollers based on the FxLMS algorithm. It operates in the time domain with real-time adaptability. On the other hand, the second design approach, the predictive ML-based ANC system, employs a pre-trained CRN to generate cancelling signals for complex, multi-directional incoming noise. CRN operates in the frequency domain, depending on STFT and ISTFT operations for real-time domain conversion. The second approach simplifies circuitry by eliminating error microphones on the basis of being predictive, while the first approaches prioritize real-time performance and adaptability. However, the distinct advantage of the second approach is in the comparatively lower computational and hardware complexity of expanding ANC systems to multi-channel configurations.

Chapter 3: Use of Modern Engineering and IT Tool. [CO9]

3.1 Introduction

In order to evaluate the proposed design and select the optimal approach, quantitative analyses are conducted through engineering simulations. The simulation tools are selected based on the process flow, and operational assumptions, while accounting for potential sources of error. The rigorous validation framework provided a reliable evaluation of the performance of each design, enabling the selection of the optimal design and subsequent modifications for maximum efficiency and functionality.

3.2 Selection of Appropriate Engineering and IT Tools

1. ***Matlab and Simulink***
   Matlab is a numerical computing platform utilized for matrix manipulations, implementation of algorithms, and data visualization. Its companion tool, Simulink, offers useful blocks and an interface for model and process design, enabling the simulation and verification of dynamic systems prior to physical deployment.

2. ***Python & Visual Studio***
   Python is a high-level programming language for developing the deep learning models, offering libraries such as TensorFlow and NumPy which allows efficient linear algebra computation. The Visual Studio Code is an integrated development environment (IDE) in which python programs can be coded, and executed. It has a comprehensive platform featuring advanced debugging and version control.

3. ***Proteus Design Suite***
   Proteus is an Electronic Design Automation (EDA) tool used to sketch electrical and electronic schematic and run embedded system simulations. The software facilitates the verification of circuit logic, as well as analog and digital hardware prior to fabrication.

4. ***COMSOL Multiphysics***
   COMSOL is a finite element analysis tool which facilitates the modeling of physical phenomena, such as acoustic field modeling. It enables rigorous analysis of sound path and propagation characteristics, providing critical boundary condition data necessary for the optimization of physical configurations.

5. ***Arduino IDE***
   Arduino is an open-sourced cross-platform developmental environment designed for compiling and verifying and uploading firmware code to microcontrollers. Its vast collection of C/C++ libraries, as well as strong community support enables real-time debugging of embedded projects.

3.3 Use of Appropriate Engineering and IT Tools

1. ***MATLAB and Simulink***
   In this project, Matlab and Simulink have been used to construct a simulation for the design approaches of adaptive ANC network. The verification of adaptive algorithms such as FxLMS and SFANC have been done through MATLAB scripts while Simulink facilitated the visual verification of the signal flow across the nodes.

2. ***Proteus Design Suite***
   In order to verify the hardware designs of both the adaptive and predictive ANC systems, proteus have been used to sketch the schematics, as well as perform simulations. The simulations have provided a realistic measure of the difficulty of hardware implementation in addition to assisting with debugging and optimization.

3. ***COMSOL Multiphysics***
   COMSOL have enabled us to simulate in three-dimension the effects of ANC system using its acoustic toolbox. The analyses have provided justification for the project, as well as important insights regarding the characteristics of acoustic paths and the effects of speaker placements.

4. ***Arduino IDE***
   Arduino IDE have functioned as the primary firmware development environment for low-level drivers and ESP-32 based subsystems, particularly during the integration of digital audio devices with controller. Additionally, it has facilitated analog to digital conversion during rapid prototyping.

3.4 Conclusion

Overall, a variety of engineering and IT tools have been used for the design, verification, and completion of the project. During the design phase, MATLAB, Simulink, and COMSOL have been utilized at various stages from idea conception, to system level verification. Subsequently, the completion stage requires the Arduino IDE and Proteus for the efficient and reliable implementation and integration of multiple components and subsystems. Visual Studio code, offering a convenient platform for python programming have been heavily relied upon through the whole duration of the project.

Chapter 4: Optimization of Multiple Design and Finding the Optimal Solution. [CO7]

4.1 Introduction

Several parameters have been selected to compare between the adaptive modular ANC system and the predictive ML-based ANC system:

- Latency
- dB reduction
- Ease of Implementation
- Cost
- Power Usage

The first two parameters, latency and dB reduction are related to the quality of cancellation and user experience. The last three parameters - Ease of Implementation, Cost, and Power Usage corresponds to our objective of making the system affordable, and efficient.

The latency defines how quickly the user can feel the effect of the system going live, as well as the level of synchronization between the projected anti-noise and external noise for maximum comfort. The dB reduction level gives us an objective metric to test out the effectiveness of different approaches and the result of fine-tuning the different parameters.

On the other hand, ease of implementation and subsequently the unit cost determines the affordability of our solution. Meanwhile, the power usage is a metric to judge the efficiency of the system, both from the engineering and commercial perspective.

4.2 Optimization of multiple design approach

In the design of an ANC system, the microcontroller and the microphone serve as the sensory processing unit and the sensory interface of the design, respectively. Optimizing these components is critical as they are the primary determinant of system latency and integrity.

4.2.1 Microcontroller

The selection of microprocessors prioritizes a balance between the budget and efficient processing. The significant processing power of the predictive ML based approach requires careful hardware selection to achieve efficiency without exceeding the budget.

Table 4.1 summarizes the key specifications of several competing choices of microcontroller offering a range of functionality at different price points.

Table 4.1: Comparing microcontrollers

Criteria	**ESP32**	**Raspberry Pi 5**	**DSP (TMS320C6678)**
RAM	512 KB SRAM	8GB SDRAM	1GB SDRAM
Clock Speed	240 MHz	2.4 GHz	1.4 GHz/core
GPIO Pins	40	40	16
CPU Architecture	Xtensa - Dual Core	Quad Core	8 Core
Dimensions (mm)	48.3 x 25.4 x 3.0	85.0 x 56.0 x 8.0	24.0 x 24.0 x 2.0
Price	BDT 400	BDT 16,420	BDT 98,800

The ESP32 microcontroller is cost-effective and suitable for simple I/O operations, however it lacks the SRAM and clock speed required to execute the complex CRN model inference within the 100ms threshold. Conversely, the industrial DSP offers superior processing density but is prohibitively expensive, violating the affordability constraint of the project.

The Raspberry Pi 5 serves as the optimal middle ground. At the fraction of the cost, it provides the necessary 2.4 GHz clock speed and 8 GB RAM to handle the ML workload and audio stream processing.

4.2.2 Microphone

The selection of input microphones focuses on the trade-off between the latency and cost. The design selection considers two primary options, digital and analog microphones.

Table 4.2: Microphone Comparison

Criteria	**Digital Mic (INP441)**	**Analog Mic (UMIK-1)**
Signal Output	Digital Output	Analog Voltage
Need for ADC	Built in ADC (analog to digital converter)	Requires an external analog to digital converter
Frequency range	~50 Hz–15 kHz	~50 Hz–15 kHz
Latency	Minimal latency	Depends on external ADC
Power	Low DC supply (1.6–3.6 V)	Connects to Mic Preamp
Price	BDT 400	BDT 20,700

4.3 Identify Optimal Design Approach

This section provides a detailed analysis of all the different criteria established in section 4.1.

4.3.1 Analysis of dB reduction and Latency

In order to evaluate the quality of cancellation through each approach, a variety of simulation tools have been utilized. The evaluation follows several stages to assess the operational efficacy and suitability of the approach for noise cancellation in the 20Hz–2000Hz range. The stages are:

1. System-level modelling
2. Hardware-level simulation
3. Verification of Microcontroller firmware code

The objective is to determine whether the system satisfies the required levels of dB reduction and latency, while sketching out a pathway to real life implementation through schematics diagram using available suitable components.

4.3.1.1 Adaptive ANC system based on FxLMS algorithm

For the adaptive ANC approach based on FxLMS, we use Simulink to perform a system level logical verification. The simulation, justifies the functionality of the design approach, demonstrating a total noise reduction of 43 dB within 3 seconds. However, the Simulink does not consider the effect of the primary and secondary path, as well as the effect of room reverberation. Moreover, the delay introduced during ADC-DAC conversion, and the processing time inherent to the microcontroller is also not compensated. Hence, this is a highly system-level ideal case which does not reflect the constraints of hardware implementation.

Fig 4.1 illustrates the Simulink block diagram utilized to model and verify the FxLMS-based adaptive ANC approach.

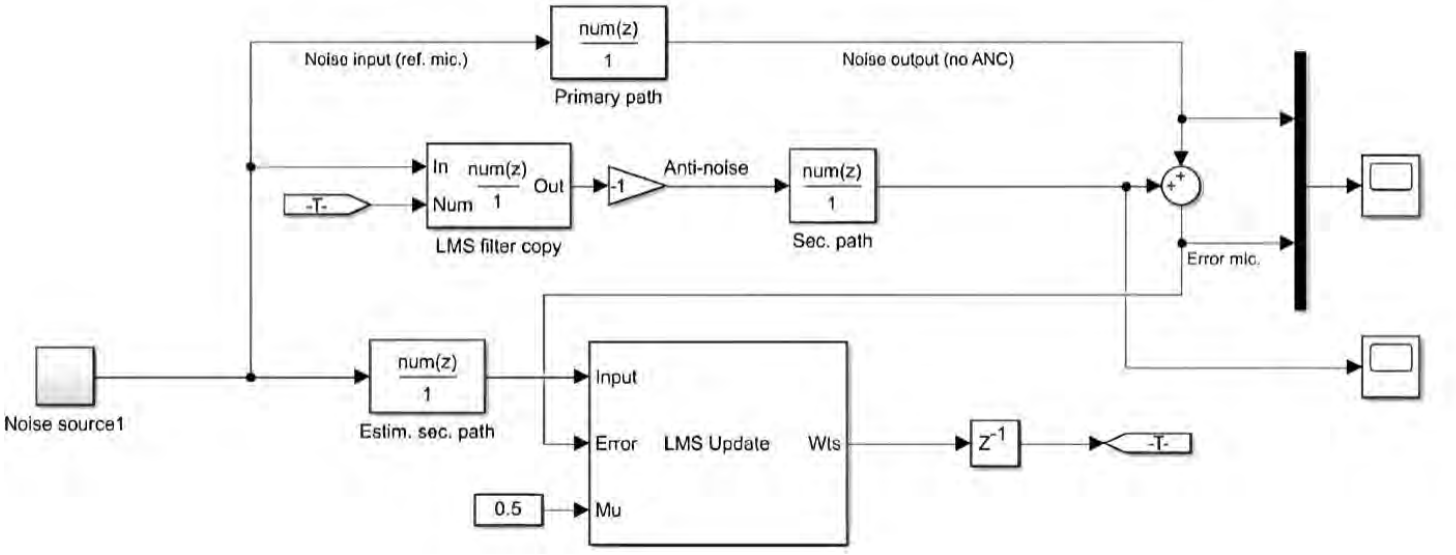


Fig 4.1: Simulink simulation model of FxLMS-based adaptive ANC algorithm

4.3.1.2 Predictive ANC system based on the ML model

The system-level verification of the predictive ML-based ANC system, involves three key stages:

1. Dataset generation
2. CRN model training
3. Testing model output

Dataset generation

The UrbanSound8k dataset, containing over 8,000 audio samples, serves as the primary source for the dataset collection. Room acoustics have been simulated based on the enclosure's shape, size, material, and reverberation time to generate primary and secondary Room Impulse Responses (RIRs). The primary RIR is convolved with an input audio sample to produce noisy signals, denoted as *d(t)*. An anti-noise signal, *y(t)*, representing the ideal noise-canceling signal, is then derived from the noisy signal. Finally, spectrograms are created for both the original input *x(t)* and the anti-noise signal *y(t)* to visualize their frequency content over time.

CRN Model Training

CRN model takes the real and imaginary spectrograms of the incoming noise signal as input and produces the real and imaginary spectrograms of the primary and secondary path compensated anti-noise signal as the output. It essentially cross-matches the input spectrograms to corresponding output spectrograms. Therefore, the model's predictive accuracy and convergence is highly contingent on both the data on which it is trained as well as the model parameters.

The CRN model integrates both convolutional layers and recurrent layers. The designing process involves the selection of the number of hidden layers, specific activation function in each layer, the values for hyperparameter, and the learning rate.

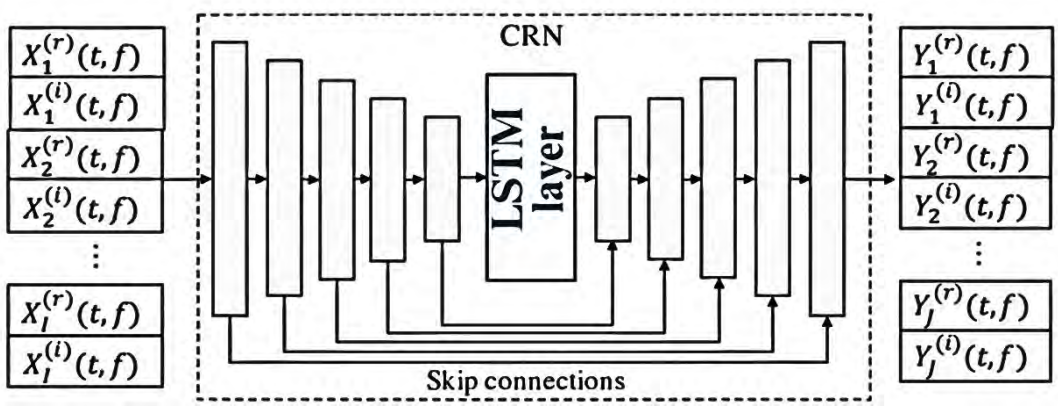


Figure 4.2**:** Model architecture

Table 4.3: Model architecture

Layer Name	**Input Size**	**Hyperparameters**	**Output Size**
conv 1	299 × 161 × 6	1 × 1, (1,1), 16	299 × 161 × 16
conv 2	299 × 161 × 16	3 × 3, (2,2), 32	150 × 81 × 32
conv 3	150 × 81 × 32	3 × 3, (2,2), 64	75 × 41 × 64
reshape_1	75 × 41 × 64	-	3075 × 64
lstm_1	3075 × 64	64	3075 × 64
dense_1	3075 × 64	64	3075 × 64
reshape_2	3075 × 64	-	75 × 41 × 64
deconv 1	75 × 41 × 64	3 × 3, (2,2), 32	150 × 81 × 32
deconv 2	150 × 81 × 64	3 × 3, (2,2), 16	299 × 161 × 16
deconv 3	299 × 161 × 64	1 × 1, (1,1), 6	299 × 161 × 6

Training is conducted over five epochs, since the relatively large dataset allows the model to improve with minimal iterations. After the first iteration, the plot indicates that the model has reached significant accuracy and can be progressed through the validation and testing stages.

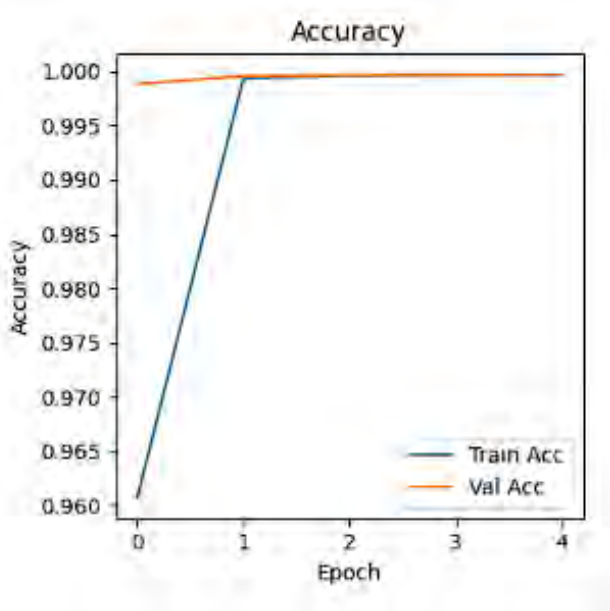


Fig. 4.3: Plot of model accuracy over the iterations

Testing Model Output

Following the training phase, the model has been subjected to rigorous performance testing. A custom test sample file processed through the CRN architecture to evaluate its predictive accuracy. Fig. 4.4 illustrates the resulting output.

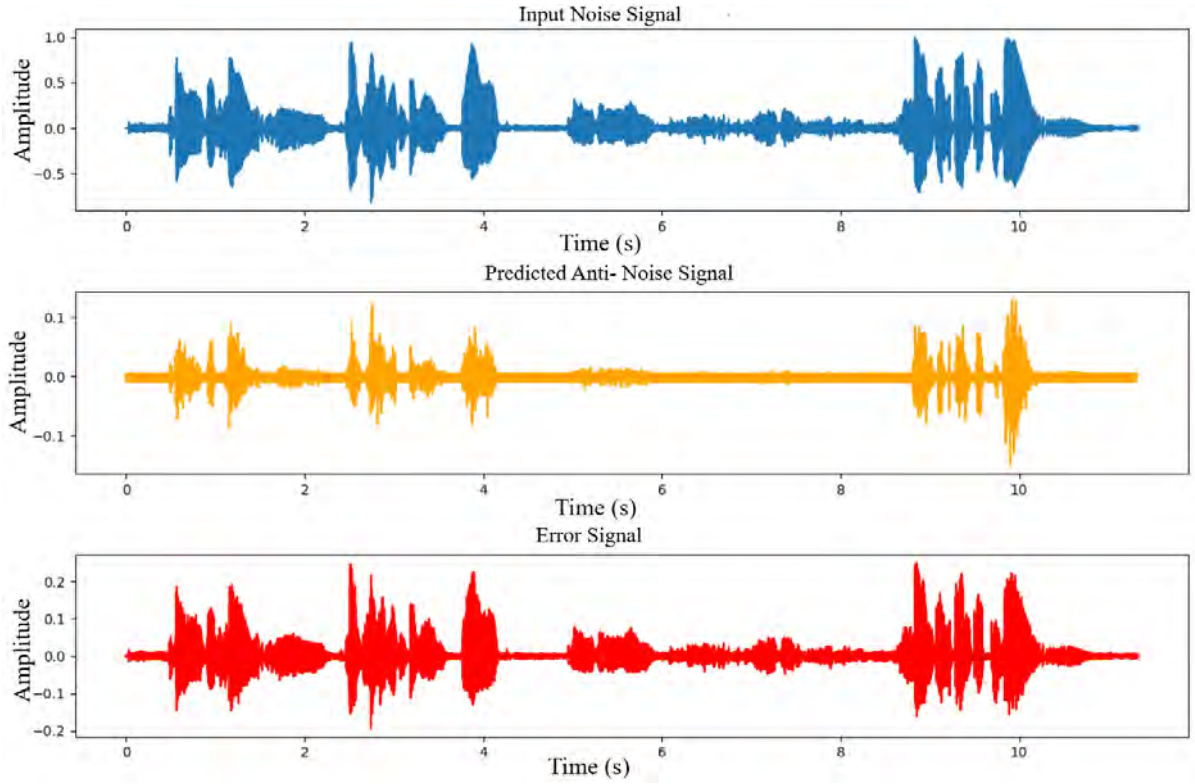


Fig. 4.4: CRN Model Testing using Example Audio for ML based Approach

The model achieves an average noise reduction of 11.62 dB and a latency of approximately 273 ms. Since the model has been already pretrained, a more realistic dB reduction is achieved while maintaining a consistent latency range. Although it exceeds the targeted latency, it has a high potential for improvement with optimizing methods.

```
Latency: 2.730 seconds
dB Reduction per Error Mic: ['11.62', '11.62', '11.62', '11.62']
Average dB Reduction: 11.62 dB
```

Fig. 4.4: Python output displaying latency and dB reduction

4.3.2 Analysis of the Power Budget

Power budget for adaptive ANC system

Table 4.4: Power usage for adaptive ANC system

Unit	Component	Quantity	Power	Total Power (W)
Audio Capture Unit	Primo EM258 Microphone	7	0.5 mA x 3.3V	0.01
Signal Pre-Processing Unit	MAX9814 Pre-Amplifier	7	30 mA x 5V	0.15
Control and Processing Unit	Raspberry Pi 4 (4GB)	3	1.4 A x 5V x 3	21
Audio Playback Unit	PCM5102A (DAC)	3	10 mA x 3.3V	3.3
	TPA3116D2 Amplifier	3	0.5 x 12V x 3	7.2
	Visaton FRS 8 Speaker	3	0.5 W	0
Feedback Unit	LM358N (Audio Mixer)	1	1.8 mA x 5V	9
	PCM5102A (ADC)	1	10 mA x 3.3V	3.3
		28	`	**~ 41 W**

Power budget for predictive ML based ANC system

Table 4.5: Power usage for ML based ANC system

Component	Voltage (V)	Current (A)	Power (W)
Microphones + MAX9814	3.3	0.013	0.043
Audio Mixer (LM358)	5	0.15	0.75

Table continued onto the next page

ADC (MCP3208)	5	0.005	0.025
Raspberry Pi 5	5	2	10
DAC (Topping E30)	5	0.5	2.5
Amplifiers (TPA3116D2)	12	0.5	24
Speakers (x4)	12	0	0
Total (16 components)			**~37 W**

4.3.4 Analysis of Cost

The primary difference in cost for both systems is attributed to the difference in microprocessor and whether there is a requirement of a codec or not. Taking that into account, the total cost of the Adaptive system is BDT 59,870 and the cost of the predictive ML based system is BDT 45,000. From a budgetary perspective, the predictive ML based system demonstrates superior cost efficiency.

4.4 Performance evaluation of developed solution

To determine the optimal design, a weighted scoring matrix is implemented for each design parameter The following tables detail the comparative performance and the specific weight attached to each parameter based on the importance to the project objectives.

Table 4.6: Performance comparison between approaches

Parameters	**Adaptive ANC system**	**ML based ANC system**
dB reduction (dB)	34.90	11.62
Latency (ms)	317	273
Implementation	Moderate (28)	Easier (16)
Cost (BDT)	59,870	45,000
Power usage (W)	41	37

Table 4.7: Comparative weighted analysis of the ANC approaches

Parameters	**Weight**	**Adaptive ANC**	**ML-based ANC**
dB reduction	25	25	21
Latency	25	12	15
Implementation	10	7.5	8
Cost (BDT)	20	12	18
Power usage	20	16	17
	100	**72.5**	**79**

4.5 Conclusion

Following a comprehensive evaluation of both design methodologies for the ANC system, Approach 2 (predictive ML-based ANC) emerges as the more realistic and promising solution. This approach leverages a ML-trained model, which demonstrates superior performance across overall parameters like adaptability, noise reduction efficiency and scalability. Unlike the Adaptive ANC approach, which relies on static algorithms, Approach 2's ML foundation allows it to continuously improve noise reduction as the dataset size increases, enabling better handling of diverse real-world noise patterns. This adaptability makes the Predictive ML-based ANC more practical for real-world ANC applications, where dynamic environments demand robust and evolving solutions.

Chapter 5: Completion of Final Design and Validation. [CO8]

5.1 Introduction

Based on the verification and analysis of the designs in *Chapter 4*, the ML-based predictive ANC system has been selected as the optimal design. The assessment takes into account performance of the system against parameters such as the dB reduction, latency, ease of implementation, cost, and power usage.

The ML model of choice is convolutional recurrent neural network (CRN), as it has demonstrated superior performance when compared with models of similar complexities [7]. This is due to the convolutional layer mapping spatial data from the spectrogram, while the recurrent layers keep track of the sequential information, resulting in accurate construction of ant-noise sequence.

The ML-based ANC design can be broken down into the following sub-systems:
1. The Audio Input Capture Unit
2. The Processing Unit
3. The Playback Unit
4. AC Power Supply

Additionally, an enclosure of MDF-18mm board was constructed to demonstrate the functionality of the proposed design approach. Based on the enclosure dimension and geometric configurations, the Room Impulse Responses (RIR) is calculated. The RIRs map how the sound is transformed due to the distance between sampling point and the center of the quiet zone, room reverberation, and wall absorption [10].

Primarily, the functionality of individual sub-systems has been designed, prototyped, and tested, before being integrated into a single system.

5.2 Completion of Final Design

5.2.1 The Audio Input Capture Unit

The purpose of this subsystem is to faithfully capture the analog signal corresponding to the pressure difference caused by the incoming sound, convert it to digital sequence and continuously feed it to the processing unit.

To achieve this, two choices of microphone are available. Firstly, the UMIK-1 analog measurement microphone, which produces an analog output that needs to be converted to a digital sequence using an ADC unit. Secondly, the INMP441 digital I2S mic module, produces a PCM-coded digital output, which is further received and processed using a microcontroller.

Although the performance, fidelity, and sensitivity of the UMIK-1 is better in comparison to the INMP441 making it much suitable for ANC purposes, it costs significantly more and requires an additional ADC unit. Therefore, the INMP441 mic is selected. However, using INMP441 mic necessitates a microcontroller (ESP32) to receive and transfer the digital sequence it produces.

The INMP441 mic is a digital MEMS microphone that sends out data via the I2S protocol. To operate the microphone with the ESP32 Microcontroller acting as the I2S master, the ESP32 sends Clock and Word Select pulse to the INMP441, which streams Serial Data to the ESP32 in return. ESP32 then collects the binary data into buffer chunks and streams it to the controller. In addition, ESP32's 3.3V pin is used to power the INMP441 mic.

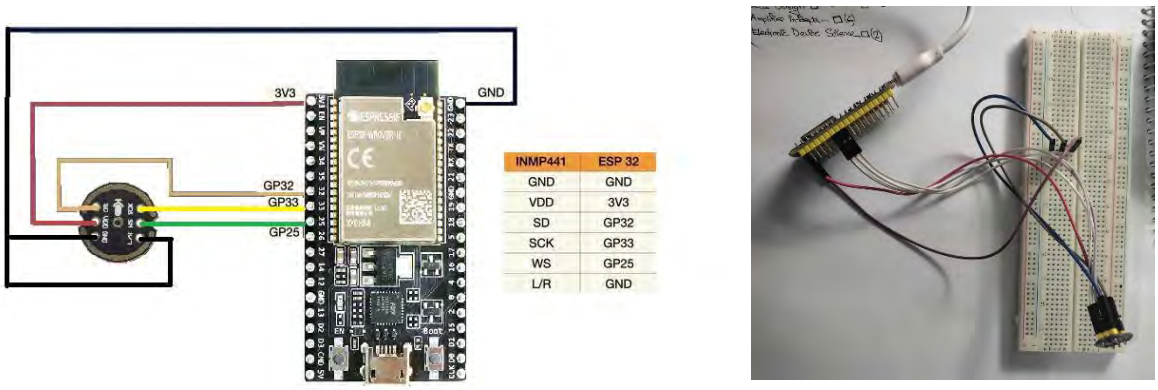


Fig. 5.1**:** INMP441 mic and ESP32 pair
(a) Schematic diagram illustrating the INMP441 and ESP32 connection. (b) Physical hardware implementation on a breadboard

The sampling frequency and mode of operation (mono L/R or stereo) of the microphone is programmed onto the ESP32 controller through the Arduino IDE platform. Additionally, the baud rate and the buffer size—the parameters determining how ESP32 microcontroller transmits the received data—are also programmed into it.

5.2.2 The Enclosure

To demonstrate the effect and efficacy of the Machine Learning Based Approach in active noise cancellation, an enclosure representing a typical closed room is built (volume has been scaled down by a factor of 100). The dimension of the room was determined to be 80 cm×65 cm×50cm. The longest side of the enclosure is 80 cm.

The range of frequencies that can be effectively cancelled within this enclosure was calculated to be 214Hz to 2000Hz. The lowest frequency was determined based on the principle that the longest side should at least be equal to half the wavelength of the wave that needs to be cancelled [11].

The lowest frequency that can physically exist within the room is:

$$f_L = \frac{c}{2L_{\max}} = \frac{343\ m/s}{2(0.8)} = 214\ Hz \tag{5.1}$$

The material that is used to build the enclosure w a Medium-Density-Fiber (MDF) board of 18mm thickness. The idea was to minimize reflection at the walls as much as possible. The inside of the enclosure was padded with acoustic foam to increase wall absorption. The total reverberation time of the room was measured to be around 0.2s.

The absorption coefficient, $\boldsymbol{\alpha}$, was measured using Sabine's formula for Room Reverberation Time, RT60 [12]:

$$T60 = (0.161) * V / S * (\alpha) \tag{5.2}$$

Where,
T60 = Reverberation time,
V = Volume of the enclosure,
S = Surface Area of the Room

The Absorption Coefficient of the Room is calculated to be:

$\boldsymbol{\alpha}$ = 0.182

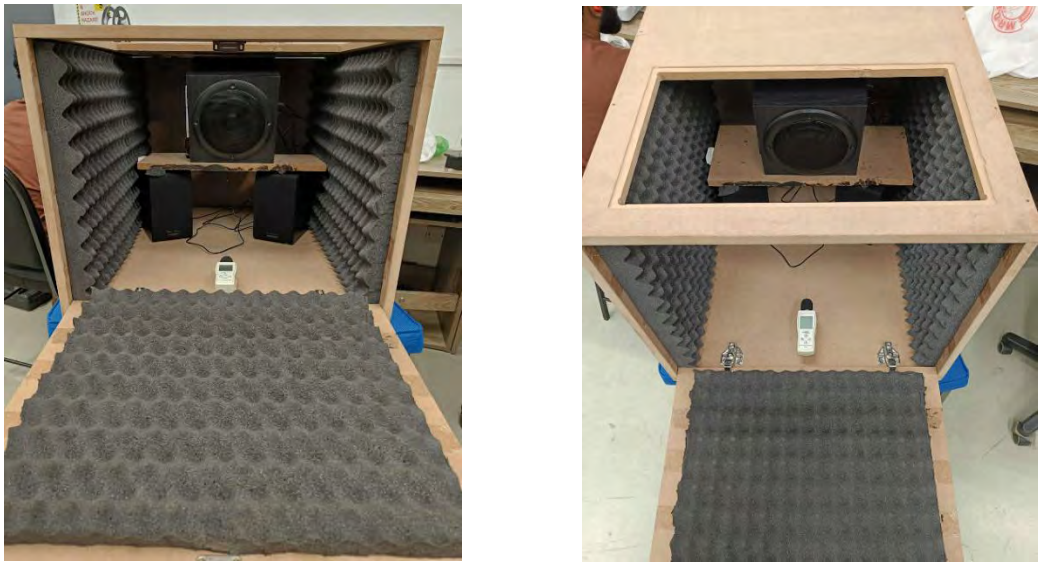


Fig. 5.2: The enclosure.
Front view(left). Top view (right)

5.2.3 Room Impulse Response

Room Impulse Response (RIR) is a way to model the acoustic effect of three-dimensional geometric shapes. RIR makes simulating the effect of the primary path (the distance between reference microphone and the control point) and the secondary path (the distance between the speakers' acoustic center and the control point) possible.

RIR is calculated by taking into account the dimension of the enclosure, the surface texture, wall material, the position of the microphones, the speaker, and the centre of quiet zone. The RIRs are then generated using the Image Method [11], using the code in *Appendix A.1*.

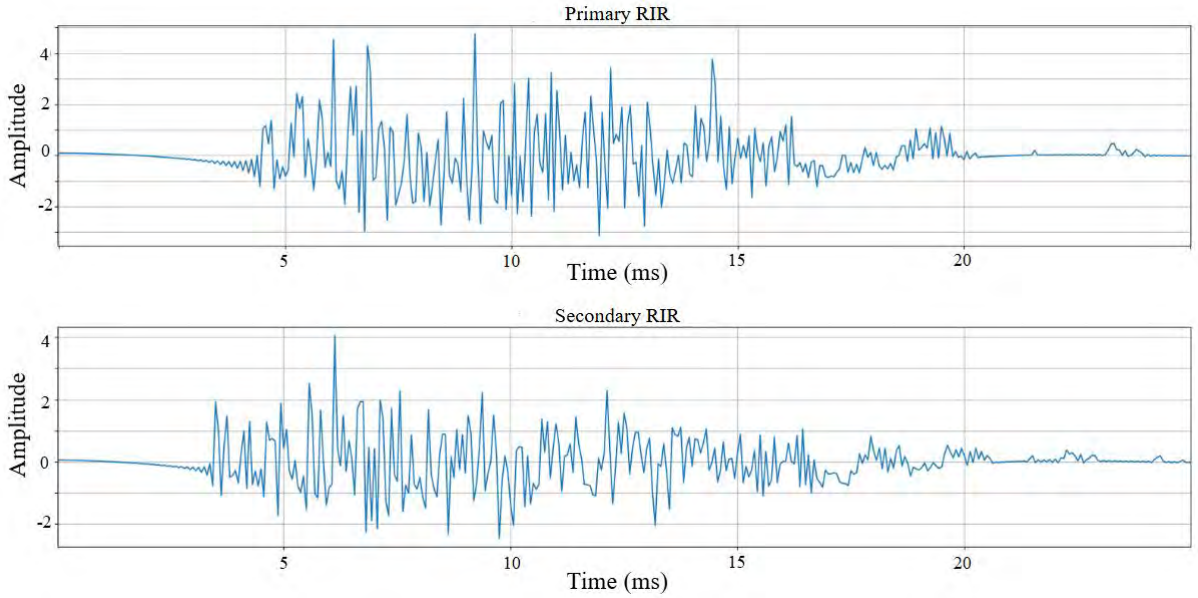


Fig. 5.3: The primary and secondary room impulse response

5.2.4 The Processing Unit

The Processing Unit is responsible for generating the digital anti-noise sequence in real-time such that the incoming noise and the subsequent anti-noise generated by the playback unit are inverts of each other at the center of the quiet zone. The processor is fed with the digital sequence produced by the INMP441 mic and transferred by the ESP32 to the serial port. Its output stream reaches the playback unit, which converts it back to analog signal, and plays it back through the speaker.

For the purpose of generating anti-noise sequence, as mentioned in the chapter 4, the CRN model was trained with a range of environmental and non-speech audio data.

Table 5.1: Sources of audio collections used to build the dataset

Source	No. of 1s Audio Clips	Notes
UrbanSound8k	21350	Non-speech environmental Data from 10 Categories
DEMAND	28650	Both speech and non-speech data. Field Recordings at Different Locations
NOISEX-92	1500	Artificial additive noise, i.e white noise, pink noise, brown noise
Field Recording	3420	Field Recording from Campus- Cafe, Exhibition Room, FYDP Lab

The dataset consists of the input spectrogram and the target spectrogram. The target spectrogram being the spectrogram of the input audio convolved with the primary RIRs, d(n). Using the code in *Appendix A.2*, the dataset was generated.

During training, the error is calculated as the difference between the ideal target spectrogram and the generated anti-noise through the CRN architecture, and the error is used to update model parameters based on the gradient descent technique, which results in the error being minimized through each training loop.

The CRN architecture consists of five convolutional layers that detect the spatial patterns in the spectrogram. After each convolution layer, the content is normalized so that each feature map ended up having unit variance and zero mean, ensuring the network learns the features within the optimal range. The convolutional layers were followed by Long-Short-Term-Memory (LSTM) layers, which captures long-term patterns across frames. The feature maps are then fed through the deconvolutional layer to generate the desired anti-noise spectrogram.

The training of CRN model is achieved using the code in *Appendix A.3.* After the training is completed, the model achieves a total of 39,010,818 parameters —nearly 39 million. *Table 5.2* on the next page shows a detailed breakdown of the model architecture, including the type of layers being used, input and output shape of each layer, as well as the number of parameters.

Table 5.2: Modified final CRN model architecture

Layer Name	Type	Input Shape	Output Shape	Number of Parameters	Notes
stft	STFTLayer	(B, 16000)	(B, 2, 161, 101)	0	STFT
conv1	Conv2d	(B,2,161,101)	(B,16,81,101)	304	Encoder
bn1	BatchNorm2d	(B,16,81,101)	(B,16,81,101)	32	Encoder
conv2	Conv2d	(B,16,81,101)	(B,32,41,101)	4640	Encoder
bn2	BatchNorm2d	(B,32,41,101)	(B,32,41,101)	64	Encoder
conv3	Conv2d	(B,32,41,101)	(B,64,21,101)	18496	Encoder
bn3	BatchNorm2d	(B,64,21,101)	(B,64,21,101)	128	Encoder
conv4	Conv2d	(B,64,21,101)	(B,128,11,101)	73856	Encoder
bn4	BatchNorm2d	(B,128,11,101)	(B,128,11,101)	256	Encoder
conv5	Conv2d	(B,128,11,101)	(B,256,6,101)	295168	Encoder
bn5	BatchNorm2d	(B,256,6,101)	(B,256,6,101)	512	Encoder
lstm1	LSTM	(B,101,1536)	(B,101,1536)	18916352	Bottleneck
lstm2	LSTM	(B,101,1536)	(B,101,1536)	18916352	Bottleneck
deconv5	ConvTranspose2d	(B,512,6,101)	(B,128,11,101)	589952	Decoder
dbn5	BatchNorm2d	(B,128,11,101)	(B,128,11,101)	256	Decoder
deconv4	ConvTranspose2d	(B,256,11,101)	(B,64,21,101)	147520	Decoder
dbn4	BatchNorm2d	(B,64,21,101)	(B,64,21,101)	128	Decoder
deconv3	ConvTranspose2d	(B,128,21,101)	(B,32,41,101)	36896	Decoder
dbn3	BatchNorm2d	(B,32,41,101)	(B,32,41,101)	64	Decoder
deconv2	ConvTranspose2d	(B,64,41,101)	(B,16,81,101)	9232	Decoder
dbn2	BatchNorm2d	(B,16,81,101)	(B,16,81,101)	32	Decoder
deconv1	ConvTranspose2d	(B,32,81,101)	(B,2,161,101)	578	Decoder
istft	ISTFTLayer	(B,2,161,101)	(B,16000)	0	ISTFT

The performance of the trained model is tested offline with a variety of test sounds unseen during training, using the code in *Appendix A.4*. Due to the large dataset that included field recordings as well as audio files covering all types of urban environmental sounds, the model responds satisfactorily, showing clear cancellation of around 12 dB on an average.

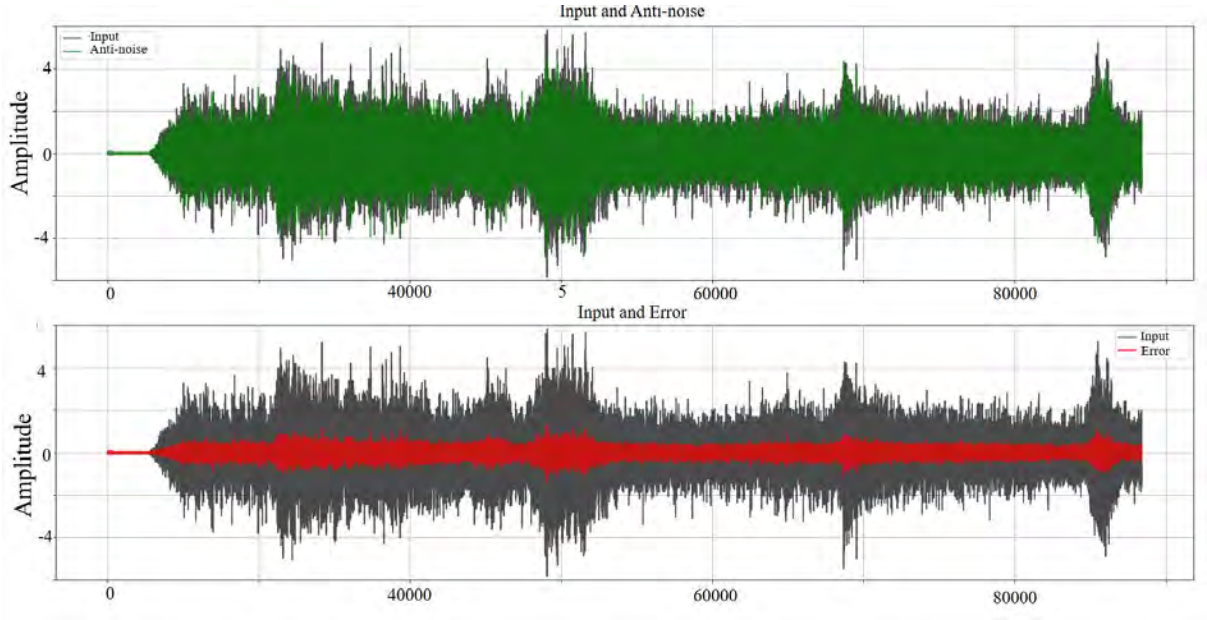


Fig. 5.4: Effect of combining incoming sound with the CRN model-generated anti-noise

The CRN model is then tested offline using the same code against a number of different noise types. *Table 5.3* lists the reduction achieved with each category of noise.

Table 5.3: Evaluation of the CRN model's offline performance

Noise Types	**Reduction**
Traffic Noise	11 dB
Cafe Ambient Noise	14 dB
Unwanted Speech	4 dB
Natural Environmental Sound	16 dB

5.2.5 The Playback Unit

The Playback Unit used is the TMN2.1 speaker set consisting of two satellite speakers and one subwoofer. The subwoofer has a diameter of 5.25 inches, which allows it to play frequencies as low as 35Hz. The maximum power consumption of the speaker set is rated to be 32W, however, for ANC application, the consumption is seen to be much lower.

The unit has a Signal-to-Noise (SNR) ratio of more than 70dB and the total harmonic distortion of less than 0.3%. high fidelity even at lower frequencies, making it a suitable choice for ANC purposes.

5.2.6 Implementation

Once each sub-systems has been proven to be functional, they are now merged to make the final prototype according to the design in *Figure 2.2*.

The Audio Capture unit is connected to the processor, and the audio data sequence from INMP441 is uploaded to the processor's serial port. The data from the serial port is put through the generated model, its output is converted back to the time domain, and is played back through the speaker. The implementation code is given in *Appendix A.5*.

5.2.7 The Challenges and Modification

These are the following technical challenges we faced during the implementation phase of the project. For most of these challenges, workaround was found through a number of debugging,

5.2.7.1 Issue with soldering the INMP441 mic

The INMP441 mic is shipped as a separate PCB board and header pins. Once the pins are soldered on, the mic stopped responding to the environment. Troubleshooting revealed that the internal connections and profuse solder leads mixed together to ground several pins, such that the whole module was shorted.

The modification was to keep an insulating material between the pins while soldering, and keeping it on after chipping away its ends once the solder hardens. This ensured the mics remained functional and non-shorted.

5.2.7.2 The processing delay in the implementation stage

The processing time for the CRN model was exceptionally high with our initial model as it used heavy-framed LSTM layers. In addition, the frequency to time domain conversion was performed outside the model. As a result of high processing time, the noise and anti-noise misaligned, making cancellation ineffective.

In order to reduce processing time, different time-domain based models like the Attention Recurrent Network (ARN) were tried, however, none offered accurate cancellations although they showed promising reduction in processing time.

Afterwards, the CRN model was revisited and several adjustments made. Firstly, the LSTM sequence was shortened. Secondly, the kernel size for encoder and decoder layers was reduced. These two adjustments reduced the total Fused Add-Multiply (FAMs) from the process, allowing the system to converge faster. To optimize the processing even further, the FFT size was brought down and the STFT and ISTFT blocks were made part of the model, as opposed to keeping them as separate DSP processes.

The modifications resulted in a tenfold reduction in processing time, the most significant among the changes was the reduction in LSTM sequence

5.2.7.3 Reduced reduction in real-world implementation

Initially, the dataset did not contain any live recordings from our own environment, which produced unsatisfactory results when left to respond to room noises. It could only respond to given audio files played by another speaker directly into the reference mic.

To solve this, recordings from different locations of the campus and raw traffic noises from the city were taken, and included in the dataset and in training. This resulted in improved cancellation.

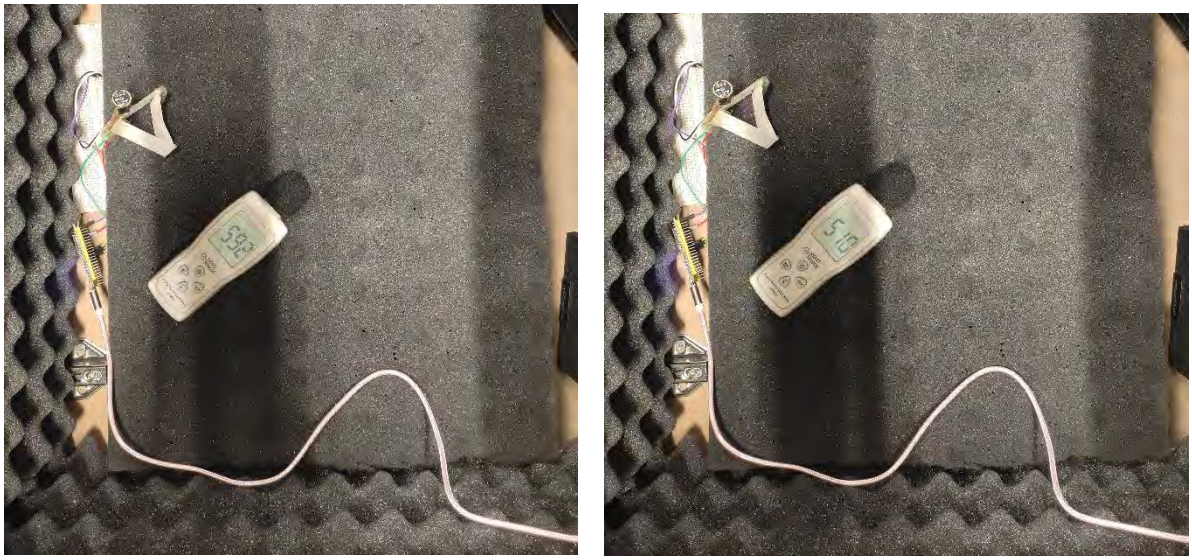
5.3 Performance Evaluation of The ANC System

To evaluate the performance of the ANC system, the prototype is placed at different locations of the campus with distinct acoustic profiles. The variation in sound and the noise level within the enclosure is monitored using a sound meter.

Table 5.4: Performance of the Prototype at Various Campus Locations

Locations	**Reduction**	**Average Latency**
Exhibition Room	14 dB	72 ms
Cafeteria	5 dB	86 ms
Roadside Classroom (8B-C)	8 dB	72ms
FYDP Laboratory (11F38L)	11dB	65ms

The average reduction stands at **9.4 dB**, fulfilling the requirement of minimum noise reduction, particularly in the range of 200 to 2000 Hz.



(a) (b)

Fig 5.5: Demonstration of a live test
(a) Without ANC (59.2 dB). (b)With ANC (51.0 dB)

The functionality of the CRN-based ANC system was visualised with the code in *Appendix A.6*. Fig. 5.6 illustrates that ANC system lowers the noise level of the room enclosure. The demonstration includes slowly turning the ANC system ON and OFF within a time period, and visualizing the waveform of the sound level inside the enclosure.

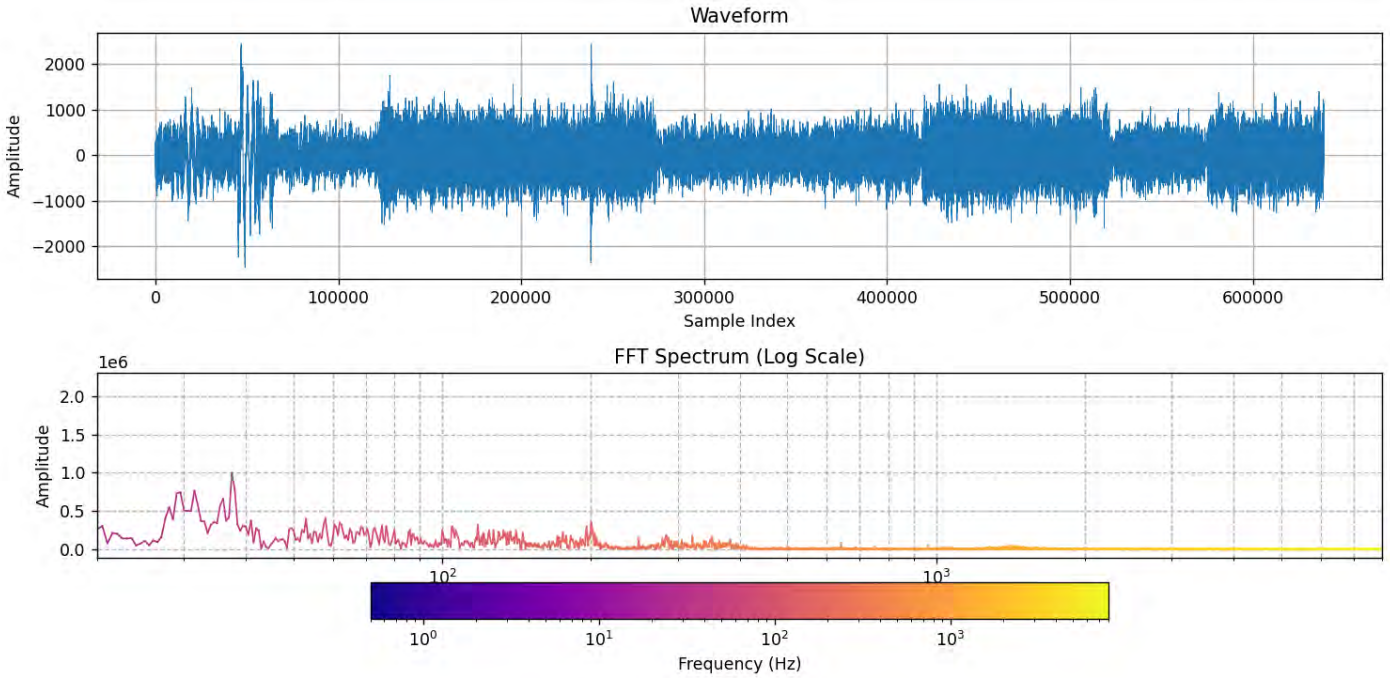


Fig. 5.6: Demonstrating the effect of ANC system on the dB level

The latency between the input sequence being detected and the anti-noise sequence being generated is calculated by timestamping each sample, and calculating the difference between

the first index of corresponding input noise and anti-noise pair. The code in *Appendix A.6* have been used to calculate the latency.

The latency has generally stayed below the required **80ms**

```
Keyboard commands:
  'r' - ANC mode OFF (raw input)
  'a' - ANC mode ON
  'q' - Quit
ANC mode ON, Latency: 1119 samples (69.9ms)
ANC mode OFF, Latency: 1074 samples (67.1ms)
ANC mode ON, Latency: 1093 samples (68.3ms)
```

Fig. 5.7: Python code output displaying latency at each toggle

The predictive CRN-based ANC system, has passed the functionality test by achieving sufficient reduction in noise levels within the required processing time and therefore fulfills the project requirements.

5.4 Conclusion

The design, implementation, and performance validation of the Convolutional Recurrent Network based Active Noise Cancellation system demonstrate the successful integration of the sub-systems into a coherent whole that can effectively reduce the serious ills of noise in urban environments. The systematic methodology that has been employed, ensure the realization of key functionalities such as real-time anti-noise generation, low-latency processing, and effective cancellation of broadband noise (200 Hz to 2000Hz). Performance evaluations showcase the system's ability to respond reliably to unseen noises in the environment. However, several areas of improvement still remain. Firstly, the system performs relatively worse against speech noise compared to non-speech noise. Secondly, to turn it into a commercial product, the processes need to be implemented via a smaller unit consisting of a powerful DSP processor. Future work would focus on making the ANC system more modular, responsive to a greater range of sounds, and suitable for market in terms of cost and portability, without significantly trading off the performance that has been achieved through the current prototype.

Chapter 6: Impact Analysis and Project Sustainability. [CO3, CO4]

6.1 Introduction

This chapter analyses the potential social and economic impacts, as well as the environmental sustainability of the proposed ML-based ANC system. The primary goal of the system is to reduce urban noise pollution to enhance the indoor environmental quality, while ensuring that the carbon footprint of the system remains low through energy-efficient design.

6.2 Assess the Impact of the Solution

Social Impact

Noise reduction provides significant health benefits because it reduces stress levels, offers better sleep quality and improves mental health. Moreover, reducing acoustic disturbances in educational and professional environments increases concentration and productivity, leading to superior learning outcomes and workplace efficiency.

Economic Impact

The system provides economic benefits in the long-term by reducing healthcare expenses and facilitating increased productivity. Moreover, the development and deployment of the proposed ANC system in urban areas generates new economic opportunities in manufacturing, installation and maintenance.

6.3 Evaluate the Sustainability

Energy Efficiency

The ANC System operates with a maximum power consumption of 50W. Compared to existing ANC noise control solutions used in aircraft cabin and in high-end cars, it is relatively energy-efficient. This efficiency is achieved through the integration of hardware components with low-power consumption and quantized real-time adaptive filtering algorithms.

Environmental Sustainability

During system design, careful selection of components eliminates the risks of harmful substances like lead, mercury and cadmium. Moreover, low energy consumption and durable components indicate lowering of carbon footprint and a minimal need for frequent maintenance or replacement. These steps cut down on total electronic waste produced over its lifespan.

SWOT Analysis

SWOT analysis have been performed to evaluate the strengths, weaknesses, opportunities and threats of the ML-based ANC system.

Table 6.1: SWOT Analysis

Strength	**Weakness**
1. Real-Time implementation 2. Seamless integration	1. Installation cost 2. Unfamiliar technology
Opportunities	**Threats**
1. Existing demand in market 2.Affordable	1. Environmental degradation of exposed components

6.4 Conclusion

The predictive ML-based ANC System represents an advanced solution to urban noise pollution that is both sustainable and economically viable. It can reduce persistent low-frequency noise components, consuming minimal power while using eco-friendly materials. These make it an excellent solution for urban noise challenges.

Chapter 7: Engineering Project Management. [CO11, CO14]

7.1 Introduction

This chapter outlines the project management framework employed during the development of the predictive ML-based ANC System. Designing, verification, prototyping, and testing are the developmental stages of the system which has required the adoption of various project management tools and procedures, facilitating the completion of project within deadline and budget while meeting the specified technical and non-technical requirements.

7.2 Define, Plan, and Manage the Engineering Project

Firstly, the initial project scope was defined in consultation with faculty advisors, focusing on the design of an Active Noise Control System that could reduce low-frequency noise in urban environments. Objectives such as noise reduction goals (10 dB), real-time processing requirements and cost-efficiency were made to be the priority. This foundational step guided the project's direction.

A thorough literature review and market survey were conducted to identify the gaps in existing solutions and to ensure that the design would be both innovative and according to the people's need. The research phase was used for selecting components and methodologies for the proposed ANC System.

Project Phases

Table 7.1: Project plan in Gantt chart

TASK	FYDP-P				FYDP-D				FYDP-C			
	Oct	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
Planning												
Research												
Design and data collection												
Simulation												
Procurement of materials												
Build and showcase												
Report												

The FYDP spanned three consecutive semesters, each corresponding to the key phases of proposal, design, and completion respectively. The phases were carefully designed to progress towards the development of the system.

During FYDP-Proposal phase, we have identified the two ANC approaches based on a careful review of literatures, surveys, and expert correspondence. Design reviews by peers and members of FYDP committee ensured that the selected design met the project requirements in terms of performance, energy efficiency and cost. Consequently, in FYDP-Design phase, the detailed simulated analysis of the adaptive ANC system based on FxLMS algorithm, and the predictive ML-based ANC system has enabled the selection of the optimal design approach. For Simulation and real-world testing, Simulink, Proteus and Python have been utilized to validate system performance. Finally, the FYDP-Completion phase has been dedicated to the development of the ANC system prototype based on the optimal design approach. This phase also includes physical testing leading to iterative refinement, preparing the system for large-scale deployment. In our FYDP showcase, we have demonstrated the functionality of the ANC system prototype by meeting the specification, functional and non-functional requirements.

7.3 Evaluation of Project Progress

Progress Tracking

Timeline management has been a key aspect of the project. Regular meetings and progress updates with the faculties has ensured that the project stayed on track. A logbook has served as the primary instrument for tracking milestones and technical iterations. Bi-weekly reviews with the ATC has also been held to evaluate the design's progression and to address any issues related to hardware compatibility or algorithm performance.

Evaluation of FYDP-P

The tasks identified during the proposal stage have been efficiently completed due to coordinated efforts among the peers, suggestions and feedback from the ATC panel, as well as the FYDP supervisory committee.

Table 7.2: FYDP P tasks

Section	Name of Task	Week
	Background Research, Tentative Problem Statement	2
	Tentative Objective, Constraints	3
	Requirements, Specifications	4
	Multiple Design Approach	5

Table continued onto the next page.

	Applicable Codes & Standards, Conclusion	6
Proposal Note	Project Plan, Gantt Chart	7
	Methodology, Expected Outcome	8
	Impact, Sustainability	9
	Budget, Ethical Consideration, Risk management, Safety Considerations	10
	Conclusion, Slide	11
Peer Assessment		12

.

Evaluation of FYDP-D

In the design stage, tools, such as, Simulink, MATLAB, Python, Proteus and Visual studio have been used for evaluating the best design approach. Even though the comparison was a close one between predictive ML based ANC and adaptive ANC system, the conclusion has been that the predictive ML based approach is the optimal design and has the most room for improvement.

Table 7.3: FYDP D tasks

Name of Task	**Week**
Selecting Appropriate Tools	1
Background Research, Survey	2
Software Learning	3,4
Software Simulation	5,6,7,8
Report Writing	9,10
Preparing Slide	11
Peer Assessment	12

Evaluation of FYDP-C

Lastly, during the final phase, the priority has been on designing the prototype for showcase and drafting the project report. In this stage, parallel testing and debugging of subsystems have played a critical role in shaping the development process.

Table 7.4: FYDP C tasks

Name of Task	**Week**
Component Selection	1
Budget Making	2
Purchasing Components	3,4
Implementation of Prototype (ML model training)	5,6,7
Implementation of Prototype (Hardware design)	8,9
System Testing	10
Report Writing	10,11
FYDP Showcase	12
Peer Assessment	13

7.4 Conclusion

The goal was to make an affordable ANC system with capability to reduce low-frequency noise. We achieved that through completion of this project. From proposal to software simulation for getting the best design then the hardware implementation was filled with multiple challenges and unexpected events. However, through collaborative problem-solving and systematic adherence to the project plan, all primary technical objectives were satisfied within the stipulated timeframe.

The current prototype demonstrates consistent noise minimization. Future work involves a couple of months for training and testing the project for real life implementation to ensure maximum accuracy.

Chapter 8: Economical Analysis [CO12]

8.1 Introduction

In this chapter, we will evaluate the ANC system as a commercial product, analysing how it may perform in the market, and whether or not it will be economically sustainable. The ANC system is designed for standard residential units, corporate offices, and general indoor environments. It focuses on affordability and practical use. The analysis has been done in consideration of the Bangladeshi market reality and store prices, providing a comprehensive overview of total purchasing, installation and maintenance cost. Ultimately, the economical analysis will help us determine not only whether the system is within technological reach, but also whether clear economic justification exists to pursue its commercialization. This segment is intended to be useful for potential investors and stakeholders.

8.2 Economic Analysis

For this system, the economic analysis will include the total cost of production, installation, and maintenance, as well as operational cost from electricity use and comparison with other similar solutions. Since this exact product is entirely novel in the market, we will compare it with the closest available solutions that address the same problem.

8.2.1 Production Cost

Investment cost includes all the expenses incurred in order to build one complete system. This includes hardware costs along with miscellaneous costs that are involved in making the product. Here are the estimated costs for one complete system.

Table 8.1: Production cost per unit

Component	**Quantity**	**Cost (BDT)**
Jetson Orin 8GB Module	1	30,355
Microphones (INMP441)	4	1600
PCB board		500
Microcontroller (ESP32)	2	1,000
Speaker set (TMN2.1)	1	6,000
Cables and connectors	N/A	500
Assemble (Locally)	N/A	1,000
Installation cost	N/A	300
Total Cost		**41255**

It is important to note that this cost is considered only for the application of ANC system to a single window. In a larger setup, costs will increase depending on the size of the room and number of windows. To reduce the cost, the product will be locally assembled. Majority of the components can be locally sourced, however those that need to be purchased from international vendors, could be imported in bulk to cut down production cost.

8.2.2 Training Cost

The system does not need user training because it is operated with a simple on and off switch. The setup is plug and play and the user can operate it without technical guidance, therefore the training cost is negligible.

8.2.3. Maintenance Cost

Maintenance is minimal, provided the user occasionally clean the layer of dust on the external components. The software updates to the system can be be provided for free, and the system has an expected lifespan of 15 to 20 years, estimated based on the lifespan of average speaker sets integrated with microcontrollers. Minor replacements, such as degraded external microphones may be required annually.

8.2.4 Estimation of Operational Cost

Operating cost depends on electricity use of each user, depending on the level of noise pollution to which they are being subjected. Power consumption of the system is a maximum of 50 watts and the average electricity rate in Bangladesh is Tk 8.25 per kWh. The device is expected to be used between 4 to 10 hours per day.

Table 8.2: Estimated Operational Cost

Daily use (in hours)	Annual Cost (in BDT)
4	603
7	1054
10	1505

8.2.5 Feasibility Analysis

Feasibility off the project has been measured using Benefit-Cost Ratio (B/C) and Present Net Worth (PNW).

Benefit-Cost ratio

The expected valuation of the ANC product by users is similar to a single-tonne Air Conditioner unit, provided its durability is comparable. Considering the unfamiliarity of the product and the price of an AC unit, the expected selling price is around BDT 55,000

Benefit to cost ratio (B/C) is therefore,

$$\frac{B}{C} = \frac{\text{Selling Price}}{\text{Procduction cost per unit}} = \frac{55{,}000}{41{,}255} = 1.33$$

B/C ratio indicates the viability of a product If B/C is less than 1, then the cost outweighs the benefits. If it is equal to 1, then it is considered to break even. And if it is greater than 1, it is considered to be commercially profitable. Since B/C is greater than 1 for the system, the project is economically viable.

To evaluate the long-term economic feasibility of the product based on ANC system, it is important to determine the PNW and IRR, the two metrics combined indicates the economic return of a project over the span of next ten years.

We assume that calculation100 units will be sold per year for the next 10 years to simplify calculation.

Present net worth

PNW provides a comprehensive valuation by discounting future cash inflows to their present-day equivalent using the Minimum Attractive Rate of Return (MARR). This ensures that the reported profitability reflects the true economic value added by the system. It is calculated using the formula,

$$PNW = -I_0 + A\left[\frac{(1+r)^n - 1}{r(1+r)^n}\right] \quad (1)$$

Where,
I_0 is the initial investment
A is the annual cash flow
n is the project duration
r is the discount rate

Based on conventions, the estimated initial investment, I_0 is calculated as 100 times the production cost per unit.

$$I_0 = 41255 \times 100 = 4\,125\,500\ BDT$$

Annual cash flow, A,

$$A = profit\ per\ unit \times 100 = (55000 - 41255) \times 100 = 1\,374\,000\ BDT$$

Assuming a flat discount rate of 12% and a project duration of 10 years,

$$PNW = -4125500 + 1374000(5.65) = \mathbf{3\,637\,600\ BDT}$$

Current valuation of the market for an ANC system based sound minimizing product is 3.6 million BDT

8.3 Cost Comparison and Market Competitiveness

The Sound Minimization System costs less than similar imported alternatives. DeNoize is a Dutch company which sells window-based noise cancellation systems that vibrate the glass panes in order to minimize noise. For a single window setup, DeNoize charges between €990 and €1210, which equals around Tk 140,000 to Tk 172,000. This calculation significantly increases with additional windows in larger room. Acoustic foam solutions for a standard apartment room in Dhaka cost between Tk 33,000 to Tk 82,500, excluding labour. It is important to note that acoustic foam only reduces echo, while interfering with natural light and

ventilation. In comparison, ANC is similar in functionality to DeNoize enabling sunlight and ventilation while being in a similar pricing range as a high-end acoustic foam.

Table 8.3: Cost comparison between ML-based ANC system and competitors

Product	**Lowest Range (BDT)**	**Highest Range (BDT)**
DeNoize	140,000	172,000
Acoustic Foam	33,000	82,500
ML-based ANC System	50,000	55,000

The ANC system between BDT 88,745 and 118,400 less compared to DeNoize, while providing improved cancellation and improved IEQ compared to acoustic foam.

8.4 Evaluation of Economic and Financial Aspects

The system has low initial investment and can be assembled locally, reducing dependency on import. The Jetson Orin module price may change slightly due to dollar rate fluctuations of around 0.127 percent. Each unit gives an average profit of about Tk 12,400. If hundred units are sold annually, the total profit is about BDT 1.24 million. Electricity cost and maintenance are low and the system is easy to use and long-lasting. The financial result shows the project is feasible for production and marketing. Moreover, cost can be reduced more if components are ordered and purchased in bulk, rather than as a single unit. This chapter explored the prices for a single unit, without bulk purchase discounts, which will also reduce the costs.

8.5 Conclusion

The ML-based ANC system is a cost-effective and financially viable solution to noise pollution. The Benefit-Cost ratio is greater than 1, and the operating and maintenance costs are low as well. The system incorporated into a sound minimizing product can be sold locally at a reasonable profit margin. While being a novel solution, it is also affordable compared to similar imported solutions and more effective than passive alternatives. The product can be deemed as a sustainable option for improving indoor environments.

Chapter 9: Ethics and Professional Responsibilities [CO13, CO2]

9.1 Introduction

When developing projects that incorporate technologies into consumer devices, it is crucial to systematically identify and address a wide range of ethical issues. This proactive approach is essential to guarantee that the resulting technology not only delivers tangible benefit but also respects core values like individual privacy rights, user safety, and long-term environmental sustainability. For instance, without careful ethical oversight, innovations in audio-based systems could inadvertently lead to misuse, such as unauthorized data collection, or contribute to environmental degradation through inefficient resource use.

The predictive ML based ANC System incorporating the CRN model, is a project designed to actively minimize low-frequency noise (in the 200–1000 Hz range) entering indoor spaces, thereby alleviating health issues like stress, sleep disturbances, and reduced productivity. However, its reliance on microphones for noise detection and machine learning for adaptive cancellation introduces potential ethical dilemmas that must be mitigated from the outset.

This chapter provides a comprehensive exploration of the ethical facets inherent to the ML-based ANC System's development and deployment. It emphasizes the alignment of the project with established professional responsibilities, including those outlined in engineering ethics codes such as those from the IEEE or NSPE, which stress the importance of public welfare, honesty, and sustainable practices.

By examining aspects like social impact, inclusivity for diverse user groups, and environmental sustainability, the chapter illustrates how ethical considerations are not peripheral but integral to the system's design. This ensures that the technology promotes equitable access to quieter living environments while minimizing any negative repercussions, ultimately fostering trust among users and stakeholders and contributing to broader societal goals like enhanced public health and eco-conscious urban development.

9.2 Identify Ethical Issues and Professional Responsibility

Privacy Concerns

One of the foremost ethical issues in the ML based ANC System stems from its use of microphones to continuously capture ambient noise data, which is processed in real-time to generate anti-noise signals via the CRN model. While this functionality is critical for achieving up to 12 dB noise reduction with low latency (around 80 milliseconds as demonstrated in simulations), it raises significant concerns about potential privacy invasions if the system were to inadvertently record personal conversations, sensitive discussions, or other private audio elements. To address this, the system is engineered to focus exclusively on measuring acoustic parameters, such as sound intensity levels, frequency spectra, and phase information, without any capability to store or analyse intelligible speech. This design choice prevents the

transformation of the system into an unintended surveillance device, which could otherwise violate user autonomy and lead to ethical breaches.

Professional responsibility in engineering demands that developers prioritize transparency and user trust, and in this project, this is achieved through multiple layers of safeguarding and transparent disclaimers. Users will be explicitly informed—via clear labelling on the device, user manuals, and initial setup interfaces—that the microphones are dedicated solely to noise cancellation purposes and do not engage in audio recording or external data transmission. All processing occurs on-device using local microcontrollers like the Raspberry Pi, eliminating risks associated with cloud-based vulnerabilities such as hacking or data leaks. Furthermore, the system's privacy policy will outline these mechanisms in detail, allowing users to verify compliance and opt out if needed, thereby aligning with global standards like GDPR or similar data protection regulations. This approach not only mitigates ethical risks but also enhances user adoption by building confidence in the technology's benign intent.

Inclusivity and Accessibility

Inclusivity and accessibility represent key ethical imperatives in the development of ANC System project, ensuring that the benefits of noise reduction are equitably distributed across all segments of society, including vulnerable groups who may be disproportionately affected by noise pollution. For example, individuals with sensory disabilities, such as those with autism spectrum disorders or hyperacusis (heightened sensitivity to sound), often experience exacerbated stress from low-frequency urban noises like traffic or construction, which the system targets for mitigation. By creating calmer indoor environments, the project directly supports mental health, improved concentration, and overall well-being, addressing broader social issues like productivity losses and health disparities in noise-exposed populations. Professional engineers have a duty to design inclusively, avoiding solutions catering to only affluent socio-economic demographic. This system fulfills that by being affordable, as well as scalable and adaptable to various infrastructures without requiring major modifications.

The ethical focus on inclusivity leads to cost-effective component choices, which allows us to keeping the total budget for the prototype around 45,000 BDT, making it viable for lower-to-middle economic demographic in the context of Bangladesh. By democratizing access to advanced ANC technology, the project upholds professional responsibilities of fostering social equity.

Environmental Impact

The environmental impact of the ML based ANC System constitutes another critical ethical issue, which centers around the reduction of ecological footprint to avoid contributing to resource wastage or climate change. Traditional passive noise control methods, such as thick acoustic panels, often rely on non-recyclable, energy-intensive materials that generate significant waste during production and disposal, whereas this active system leverages electronic components for destructive interference, minimizing material use.

The design incorporates eco-friendly principles from design stage, selecting energy-efficient hardware like low-powered Raspberry Pi processors and speakers with a frequency range of 20Hz–15kHz, which operate at under 37W total consumption, far below the 80W threshold. This efficiency reduces operational energy demands, particularly important in regions with unstable power grids, and aligns with global sustainability goals by lowering greenhouse gas emissions associated with prolonged device usage.

Professional responsibility extends to lifecycle sustainability. To that end, the system uses non-toxic, recyclable materials for enclosures and circuitry, ensuring safe end-of-life disposal without releasing harmful substances like lead or mercury into ecosystems. For example, components are chosen to comply with Restriction of Hazardous Substances (RoHS) standards, promoting a circular economy where parts can be reused or recycled. Additionally, by avoiding bulky installations, the system reduces the environmental burden of transportation and manufacturing, making it a more sustainable alternative for urban noise management. This ethical commitment not only mitigates direct impacts but also sets a precedent for future engineering endeavours, encouraging the integration of green practices that balance technological advancement with planetary health.

9.3 Apply Ethical Issues and Professional Responsibility

Environmental Sustainability

In applying ethical principles to environmental sustainability, the ML-based ANC System positions itself as a progressive solution that actively combats urban noise pollution while minimizing ecological harm. The system's energy efficiency—achieved through optimized ML models that predict anti-noise with minimal computational overhead—ensures low power usage during continuous operation, thereby reducing reliance on fossil fuel-based electricity and curbing carbon emissions. For instance, simulations show the system stabilizing in about 80 ms with around 10 dB reduction, while consuming only 37W. This is significantly lower than traditional ANC setups requiring high-end DSPs. By eliminating the need for resource-heavy passive materials like dense foams, the design further decreases manufacturing waste and raw material extraction, contributing to conservation efforts in material-scarce regions.

Future iterations will build on this foundation by emphasizing sustainable sourcing, such as partnering with suppliers of recycled electronics and incorporating biodegradable elements where feasible, to enhance the system's environmental profile. This application of ethics involves ongoing assessments, like lifecycle analyses, to quantify impacts and refine designs, ensuring compliance with standards like ISO 14001 for environmental management. Ultimately, this approach not only fulfills professional duties to protect the environment but also amplifies the project's positive ripple effects, such as promoting energy-efficient urban infrastructures that support global climate initiatives.

Transparency

Transparency is integral to the ML-based ANC system project, as evident through robust mechanisms that empower users with complete knowledge of data handling practices, therefore preventing misconceptions and building accountability. Users are provided with detailed, accessible information—via app interfaces, physical labels, and digital policies—explaining that ambient noise data is processed instantaneously for cancellation purposes only, with no storage of personal audio or transmission to third parties. For example, the privacy policy will include specifics on how the CRN model uses frequency-domain data without retaining raw inputs, allowing users to monitor processes upon request. This addresses potential fears of hidden surveillance, especially in privacy-sensitive contexts like homes or offices.

Moreover, users retain control through features like opt-in diagnostics or manual overrides, ensuring informed participation. Professional responsibility is upheld by integrating these elements into the design phase, with regular ethical reviews to adapt to emerging concerns, such as new data laws. This transparent framework not only mitigates risks but also enhances the system's societal value, encouraging ethical adoption of AI-driven technologies.

9.4 Conclusion

In conclusion, the ML-based ANC system demonstrates an unwavering commitment to ethical design by embedding principles of privacy protection, universal accessibility, and environmental sustainability deeply into its architecture and operations. Through careful cybersecurity measures like local processing, the project ensures that noise reduction benefits are delivered responsibly, without compromising user rights or ecological balance. This holistic approach aligns with professional engineering ethics, prioritizing public welfare over mere functionality.

Affordability further extends these ethical imperatives, achieved through a cost-effective hardware setup, making the system accessible to low-middle income groups, promoting social inclusivity in noisy urban areas. By adhering to high performance and ethical standards, the system contributes to improved public health, reduced environmental strain, and greater societal harmony, exemplifying how ethical engineering can drive meaningful, positive change.

Chapter 10: Conclusion and Future Work

10.1 Project Summary

The aim of this project is to address the growing issue of noise pollution in metropolitan cities without the invasive disruption to the infrastructure. We have explored multiple design approaches to establish suitable method of reducing external noise in indoor spaces, before narrowing down to an optimal ANC system design. In addition to the basic functional requirement of reducing noise in real-time, the system needs to be cost-efficient and modular, with a minimal carbon footprint and adverse environmental impact. The system targeted frequency components within the range of 20 to 2000 Hz, since most background noise falls within this range. The working principle of the system is to use microphones positioned outside the enclosure to capture up unwanted noise, process it in real-time through the ML algorithm that we trained with both open-source and local audio samples, generating an anti-noise signal which cancels the noise when played through the speakers. Here, the unwanted sound, which we have referred to as noise, is classified as the sound entering the indoor space from the external environment.

For the sake of demonstration in the final year showcase, we have built a prototype with a wooden enclosure to represent a room. We used MDF boards of 18 mm thickness to create the walls of the box. Inside the enclosure, acoustic foam was attached to the walls in order to accurately simulate how sound would react in a regular room. To clarify, the MDF board and acoustic foam will not be part of the final design or product. As we have demonstrated in the FYDP showcase, the prototype reduces low-frequency noise within a set duration, illustrating that both the design and algorithm is performing as intended. The design, implementation, and performance validation of the CRN-based ANC system described in previous chapters demonstrate the successful integration of the sub-systems as well.

The system reduces incoming noise by 10 to 12 dB, as we have verified with a noise meter. Although the protype currently runs on an 8GB Ryzen processor, future iterations will switch to Jetson Orin module, which is superior for embedded machine learning and will miniaturize the system, making it more efficient and affordable in the long run.

Beside balancing cost and performance in hardware selection, ensuring low processing time have been the most pressing engineering challenge for the project. In addition, determining the optimal position of speakers in the ANC system is the primary acoustic challenge we have overcome over the course of the project.

However, several areas of improvement still remain. Firstly, the system performs relatively worse against speech noise compared to non-speech noise. Secondly, to turn it into a commercial product, the processes need to be implemented via a smaller unit consisting of a powerful DSP processor. Ethical considerations and applicable codes are also considered in the process of finalising the system as a finished product, which have also been addressed in previous chapters. Economic analysis was also done for the system to prove that it has potential for being a viable product. As a finished product, the Sound Minimization System has a potential credibility to become a cost effective, practical solution built to tackle noise pollution.

10.2 Future Work

In order to advance the proposed ANC system from a functional prototype to a commercially viable product, several key areas of improvements remain. Firstly, transitioning the processing unit into a dedicated Jetson Orin module, will reduce the physical footprint as well as offer faster inference times for the CRN model. Secondly, system latency can be further reduced through software optimization by streamlining the signal processing pipeline and experimenting with different variations of the CRN model. Thirdly, modular design can be adopted to increase product longevity and user-friendliness, however that will require both acoustic analysis of the typical state of the enclosures in which the system is implemented. Finally, extensive field testing across diverse acoustic environments found in residential homes, corporate offices, and industrial sites is required in order to validate the acoustic performance and durability of the system under different physical conditions.

To summarize, future work will prioritize enhancing the modularity and performance of the current prototype. Although necessary improvements are needed to ensure the product is ready to be released to the market, it has successfully met the design requirements of the FYDP, demonstrating the physical realization of a conceptual solution addressing a relevant problem in the context of Bangladesh.

Chapter 11: Identification of Complex Engineering Problems and Activities

11.1 Applicable Attributes of Complex Engineering Problem

Table 11.1: Attributes of Complex Engineering Problems

	Attributes	**Put tick (√) as appropriate**
P1	Depth of knowledge required	√
P2	Range of conflicting requirements	√
P3	Depth of analysis required	√
P4	Familiarity of issues	
P5	Extent of applicable codes	
P6	Extent of stakeholder involvement and needs	√
P7	Interdependence	

11.2 Justification of the Selected Attributes of Engineering Problem

P1: Depth of Knowledge

The project necessitates a high degree of technical proficiency, synthesizing the FxLMS algorithm with CRN models for predictive noise attenuation. COMSOL and Python have been used to prepare the system for real-world acoustic environments. This integration has been validated through power budgets and dB reduction metrics which showcases comprehensive knowledge in signal processing, ML and hardware enables effective low-frequency noise minimization.

P2: Range of Conflicting Requirements

The primary conflict in this project has been the trade-off between latency and affordability. Low latency requires expensive components; however, affordability is one of our key non-functional requirements. The design process of the ANC system involves evaluation of several such conflicts, resolved through balancing trade-offs based on ethical, practical and engineering considerations.

P3: Depth of Analysis

Based on extensive literature review and expert correspondence, the different design configurations of both the adaptive ANC system based on FxLMS algorithm and the ML-based ANC system employing the CRN model has been analyzed. The analysis also includes detailed component and system-level simulations.

P6: Extent of Stakeholder Involvement and Needs

Stakeholder requirements significantly influenced the design process. In particular, the extent of the problem being addressed, as well as the economic feasibility, relevant engineering and societal challenges of the ANC system have set the constraints within which the system was conceptualized, designed and assessed.

11.3 Applicable Activities of Complex Engineering Problem

Table 11.2: Attributes of complex engineering activities

	Attributes	**Put tick (√) as appropriate**
A1	Range of resource	√
A2	Level of interaction	
A3	Innovation	
A4	Consequences for society and the environment	√
A5	Familiarity	√

11.4 Justification for the Selected Activities of Complex Engineering Problem

A1: Range of Resources

A comprehensive suite of hardware and software tools have been utilized to facilitate the successive stages of project development. This ranges from physical testing in a number of diverse acoustic environments to design validation through simulation software. The diversity of tools employed seeks to address potential sources of error during the conceptual stage.

A4: Consequences for Society and Environment

It improves quality of life by reducing noise-related ailments while using environmentally efficient components with lower energy use. Specific considerations have been made to ensure that the system is not engaged in potential unethical practices, by making data processing strictly local and offline.

A5: Familiarity

The design of the ANC system based on ML algorithms and signal processing techniques has consistently challenged us on unfamiliar grounds, necessitating a broadening of our areas of expertise in theoretical as well as practical engineering domains.

References

[1] D. S. Jahan, “Noise pollution and its probable impacts on public health in Dhaka city,” *J. Eden Mohila Coll.*, vol. 2, pp. 69–81, Dec. 2022.

[2] J. Ma, C. Li, M. P. Kwan, and Y. Chai, “A multilevel analysis of perceived noise pollution, geographic contexts and mental health in Beijing,” *Int. J. Environ. Res. Public Health*, vol. 15, no. 7, p. 1479, Jul. 2018. doi: 10.3390/ijerph15071479.

[3] W. Peng, J. Zhang, M. Shi, J. Li, and S. Guo, “Low-frequency sound insulation optimization design of membrane-type acoustic metamaterials based on Kriging surrogate model,” *Mater. Des*., vol. 225, p. 111491, Jan. 2023, doi: 10.1016/j.matdes.2022.111491. 24/25

[4] B. Lam, W.S. Gan, D. Shi, M. Nishimura, and S. Elliott, “Ten questions concerning active noise control in the built environment,” *Build. Environ.*, vol. 200, p. 107928, Aug. 2021, doi: 10.1016/j.buildenv.2021.107928.

[5] S. J. Elliott and P. A. Nelson, “Active noise control,” *IEEE Signal Process. Mag*., vol. 10, no. 4, pp. 12–35, Oct. 1993, doi: 10.1109/79.248551.

[6] Y. Li and W. Zheng, “A noise control method using adaptive adjustable parametric array loudspeaker to eliminate environmental noise in real time,” *Int. J. Environ. Res. Public. Health*, vol. 19, no. 1, p. 269, Dec. 2021, doi: 10.3390/ijerph19010269.

[7] H. Zhang and D. Wang, “Deep MCANC: A deep learning approach to multi-channel active noise control,” *Neural Netw.*, vol. 158, pp. 318–327, Jan. 2023. doi: 10.1016/j.neunet.2022.11.029.

[8] S. M. Kuo and D. R. Morgan, “Active noise control: A tutorial review,” *Proc. IEEE*, vol. 87, no. 6, pp. 943–973, Jun. 1999. doi: 10.1109/5.763310.

[9] “Active Noise Reduction using LMS and FxLMS Algorithms,” ResearchGate, Jan. 2025. [Online]. Available: www.researchgate.net

[10] J. Lin, G. Götz, and S. J. Schlecht, “Deep room impulse response completion,” *J. Audio Speech Music Proc.*, vol. 2025, no. 20, 2025. doi: 10.1186/s13636-024-00383-1.

[11] F. Ma, W. Zhang, and T. D. Abhayapala, “Active control of outgoing noise fields in rooms,” *J. Acoust. Soc. Am.*, vol. 144, no. 5, pp. 2822–2831, Nov. 2018. doi: 10.1121/1.5055217.

[12] L. L. Beranek, “Analysis of Sabine and Eyring equations and their application to concert hall audience and chair absorption,” *J. Acoust. Soc. Am.*, vol. 120, no. 3, pp. 1399–1410, Sep. 2006. doi: 10.1121/1.2221392.

APPENDIX A

Source Codes

A.1: RIR Generation and Acoustic Modeling

```
import numpy as np
import os
import pyroomacoustics as pra
import h5py
import matplotlib.pyplot as plt

# Hyperparameters
FS = 16000  # Sampling rate
T60_VALUE = 0.2  # Single T60 value
ROOM_DIM = [0.8, 0.65, 0.5]  # x, y, z in meters
REF_MIC_POS = np.array([0.00, 0.325, 0.12])  # Reference mic
SPEAKER_POS = np.array([0.35, 0.325, 0.12])  # Speaker (mono midpoint of
satellites)
CONTROL_POINT_POS = np.array([0.70, 0.325, 0.12])  # Control Point Center
(error mic)
MAX_ORDER = 10  # As specified
RIR_FILE = r"E:\Final\RIRs\t60rirs.h5"  # Single .h5 file

# Function to generate and store RIRs in .h5
# Function to generate and store RIRs in .h5
def generate_rirs():
    os.makedirs(os.path.dirname(RIR_FILE), exist_ok=True)
    with h5py.File(RIR_FILE, 'w') as f:
        primary_group = f.create_group('primary')
        secondary_group = f.create_group('secondary')

        # Compute absorption for T60 = 0.2s
        absorption, _ = pra.inverse_sabine(T60_VALUE, ROOM_DIM)

        # Room for primary RIR (ref mic to control point)
        room_primary = pra.ShoeBox(ROOM_DIM, fs=FS,
materials=pra.Material(absorption), max_order=MAX_ORDER,
air_absorption=True)
        room_primary.add_source(REF_MIC_POS)
        room_primary.add_microphone(CONTROL_POINT_POS)
        room_primary.compute_rir()
        rir_primary = room_primary.rir[0][0]
        # Apply L2 normalization to preserve energy decay
        rir_primary /= np.sqrt(np.sum(rir_primary**2))
        primary_group.create_dataset('rir_0', data=rir_primary,
compression='gzip')

        # Room for secondary RIR (speaker to control point)
        room_secondary = pra.ShoeBox(ROOM_DIM, fs=FS,
materials=pra.Material(absorption), max_order=MAX_ORDER,
air_absorption=True)
        room_secondary.add_source(SPEAKER_POS)
        room_secondary.add_microphone(CONTROL_POINT_POS)
        room_secondary.compute_rir()
        rir_secondary = room_secondary.rir[0][0]
        # Apply L2 normalization
        rir_secondary /= np.sqrt(np.sum(rir_secondary**2))
        secondary_group.create_dataset('rir_0', data=rir_secondary,
compression='gzip')
```

```
    print("RIRs generated with L2 normalization (preserves realistic decay)
and saved to", RIR_FILE)
# Function to plot waveforms for T60 = 0.2s
def plot_rirs():
    with h5py.File(RIR_FILE, 'r') as f:
        rir_primary = f['primary/rir_0'][:]
        rir_secondary = f['secondary/rir_0'][:]

    t_primary = np.arange(len(rir_primary)) / FS * 1000  # ms
    t_secondary = np.arange(len(rir_secondary)) / FS * 1000

    plt.figure(figsize=(12, 6))

    plt.subplot(2, 1, 1)
    plt.plot(t_primary, rir_primary)
    plt.title(f'Primary RIR (Ref Mic to CP, T60={T60_VALUE}s, Realistic
Attenuation)')
    plt.xlabel('Time (ms)')
    plt.ylabel('Amplitude')
    # Auto ylim to show small amplitudes
    plt.grid(True)

    plt.subplot(2, 1, 2)
    plt.plot(t_secondary, rir_secondary)
    plt.title(f'Secondary RIR (Speaker to CP, T60={T60_VALUE}s, Realistic
Attenuation)')
    plt.xlabel('Time (ms)')
    plt.ylabel('Amplitude')
    plt.grid(True)

    plt.tight_layout()
    plt.show()

    # Print max amplitudes for verification (should be ~1/(4*pi*distance))
    dist_primary = np.linalg.norm(REF_MIC_POS - CONTROL_POINT_POS)
    dist_secondary = np.linalg.norm(SPEAKER_POS - CONTROL_POINT_POS)
    expected_primary = 1 / (4 * np.pi * dist_primary)
    expected_secondary = 1 / (4 * np.pi * dist_secondary)
    print(f"Primary RIR max amplitude: {np.max(np.abs(rir_primary)):.4f}
(expected ~{expected_primary:.4f})")
    print(f"Secondary RIR max amplitude:
{np.max(np.abs(rir_secondary)):.4f} (expected ~{expected_secondary:.4f})")

    print("Primary RIR values:", rir_primary)
    print("Secondary RIR values:", rir_secondary)
if __name__ == "__main__":
    generate_rirs()
    plot_rirs()
```

A.2 Dataset Preparation Code

```
import os
import time
import struct
import queue
import threading
import wave
import signal
import sys
import numpy as np
```

```
import serial
import pyaudio
import torch
from collections import deque
from scipy.signal import butter, lfilter

# ----------------------------
# CONFIG
# ----------------------------
COM_PORT = "COM9"  # ESP32 serial port
BAUD_RATE = 500000  # Must match ESP32
MODEL_JIT_PATH = r"C:\Users\barrl\Downloads\crn_e1.pt"  # TorchScript file
OUTPUT_INPUT_WAV = r"D:\TestFIles\CRN\esp32_raw_input.wav"
OUTPUT_MODEL_WAV = r"D:\TestFIles\CRN\mcanc_model_output.wav"
OUTPUT_ERROR_WAV = r"D:\TestFIles\CRN\error_signal.wav"
SAMPLE_RATE = 16000
CHANNELS = 1
PY_AUDIO_FORMAT = pyaudio.paInt16

# ESP32 framing
ESP32_BUFFER_SIZE = 256  # Samples per frame from ESP32
ESP32_PAYLOAD_BYTES = ESP32_BUFFER_SIZE * 2  # 16-bit samples

# Model input/output
MODEL_WINDOW_SAMPLES = 16000  # 1 second at 16 kHz
MODEL_HOP_SAMPLES = 1600  # 0.1 seconds for streaming output

# Queue sizes - increased to prevent drops
PLAYBACK_Q_MAX = 32
MODEL_Q_MAX = 64
OUTPUT_Q_MAX = 32

# Playback modes
MODE_OFF = 'off'
MODE_ON = 'on'

# ----------------------------
# Global state
# ----------------------------
current_mode = MODE_OFF
latency_samples = 0
mode_lock = threading.Lock()

# ----------------------------
# Utility functions
# ----------------------------
def save_wav_int16(path, int16_numpy_array, sample_rate=SAMPLE_RATE):
    os.makedirs(os.path.dirname(path), exist_ok=True)
    with wave.open(path, "wb") as wf:
        wf.setnchannels(CHANNELS)
        wf.setsampwidth(2)
        wf.setframerate(sample_rate)
        wf.writeframes(int16_numpy_array.tobytes())

def apply_crossfade(buffer1, buffer2, crossfade_samples=32):
    """Apply crossfade between two buffers to reduce clicks"""
    if len(buffer1) < crossfade_samples or len(buffer2) <
crossfade_samples:
        return buffer2

    result = buffer2.copy()
```

```
    fade_out = np.linspace(1.0, 0.0, crossfade_samples)
    fade_in = np.linspace(0.0, 1.0, crossfade_samples)

    result[:crossfade_samples] = (buffer1[-crossfade_samples:] * fade_out
+
                                  buffer2[:crossfade_samples] * fade_in)
    return result

# ----------------------------
# SerialReader - FIXED
# ----------------------------
class SerialReader(threading.Thread):
    def __init__(self, ser, playback_q, model_q):
        super().__init__(daemon=True)
        self.ser = ser
        self.playback_q = playback_q
        self.model_q = model_q
        self.running = True
        self.packet_count = 0  # FIXED: Add packet counter for debugging

    def wait_for_header(self):
        state = 0
        while self.running:
            b = self.ser.read(1)
            if not b:
                return False
            if state == 0:
                if b == b'\xAA':
                    state = 1
            else:
                if b == b'\x55':
                    return True
                state = 1 if b == b'\xAA' else 0
        return False

    def run(self):
        while self.running:
            ok = self.wait_for_header()
            if not ok:
                continue
            size_bytes = self.ser.read(2)
            if len(size_bytes) < 2:
                continue
            payload_size = struct.unpack("<H", size_bytes)[0]
            audio_bytes = self.ser.read(payload_size)
            if len(audio_bytes) < payload_size:
                continue

            # Add timestamp for latency calculation
            timestamp = time.time()
            packet = (audio_bytes, timestamp)
            self.packet_count += 1

            # FIXED: Better queue management - don't drop packets
unnecessarily
            # For playback queue
            if not self.playback_q.full():
                self.playback_q.put(packet, block=False)
            else:
                if self.packet_count % 100 == 0:  # Log occasionally
```

```
                    print(f"Warning: Playback queue full, dropping packet
{self.packet_count}")

            # For model queue
            if not self.model_q.full():
                self.model_q.put(packet, block=False)
            else:
                if self.packet_count % 100 == 0:  # Log occasionally
                    print(f"Warning: Model queue full, dropping packet
{self.packet_count}")

    def stop(self):
        self.running = False

# ----------------------------
# ModelWorker - UNCHANGED (working correctly)
# ----------------------------
class ModelWorker(threading.Thread):
    def __init__(self, jit_model, model_q, output_q, input_save_list,
output_save_list, device='cpu'):
        super().__init__(daemon=True)
        self.model = jit_model
        self.device = device
        self.q = model_q
        self.output_q = output_q
        self.input_save_list = input_save_list
        self.output_save_list = output_save_list
        self.running = True
        self.feed_buf = np.zeros(0, dtype=np.float32)
        self.out_fifo = np.zeros(0, dtype=np.float32)
        self.window_samples = MODEL_WINDOW_SAMPLES
        self.hop_samples = MODEL_HOP_SAMPLES
        self.latency_measurements = deque(maxlen=10)  # Keep last 10
measurements

        # For smoother processing - reduce crossfade to minimize artifacts
        self.last_output = np.zeros(MODEL_HOP_SAMPLES, dtype=np.float32)

    def run(self):
        global latency_samples

        while self.running:
            try:
                audio_bytes, input_timestamp = self.q.get(timeout=0.5)
            except queue.Empty:
                continue

            # Convert bytes to int16 and save for WAV
            in_int16 = np.frombuffer(audio_bytes,
dtype='<i2').astype(np.int16)
            self.input_save_list.append(in_int16.copy())

            # Normalize to float32 [-1, 1)
            samples = in_int16.astype(np.float32) / 32768.0
            self.feed_buf = np.concatenate([self.feed_buf, samples])

            # Process 1-second windows when available
            while len(self.feed_buf) >= self.window_samples:
                window = self.feed_buf[:self.window_samples].copy()

                # Convert to torch tensor
```

```
                input_tensor =
torch.from_numpy(window).float().unsqueeze(0).to(self.device)

                # Measure inference time
                start_time = time.time()
                with torch.no_grad():
                    output_tensor = self.model(input_tensor)
                end_time = time.time()

                # Calculate latency in samples
                inference_time_sec = end_time - start_time
                inference_samples = int(inference_time_sec * SAMPLE_RATE)
                self.latency_measurements.append(inference_samples)

                # Update global latency with running average
                if len(self.latency_measurements) > 0:
                    latency_samples =
int(np.mean(self.latency_measurements))

                y_window =
output_tensor.squeeze(0).cpu().numpy().astype(np.float32)
                self.out_fifo = np.concatenate([self.out_fifo, y_window])
                self.feed_buf = self.feed_buf[self.hop_samples:]

            # Flush hop-sized chunks to output queue
            while len(self.out_fifo) >= self.hop_samples:
                to_play = self.out_fifo[:self.hop_samples].copy()
                self.out_fifo = self.out_fifo[self.hop_samples:]

                # Apply light crossfade for smoother transitions
                to_play = apply_crossfade(self.last_output, to_play,
crossfade_samples=32)
                self.last_output = to_play.copy()

                # Convert to int16
                int16_chunk = np.clip(to_play * 32767.0, -32768,
32767).astype(np.int16)
                self.output_save_list.append(int16_chunk.copy())

                try:
                    self.output_q.put((int16_chunk, input_timestamp),
block=False)
                except queue.Full:
                    # Remove oldest if full
                    try:
                        self.output_q.get_nowait()
                        self.output_q.put((int16_chunk, input_timestamp),
block=False)
                    except queue.Empty:
                        pass

    def stop(self):
        self.running = False

# ----------------------------
# ErrorPlaybackThread - MAJOR FIXES
# ----------------------------
class ErrorPlaybackThread(threading.Thread):
    def __init__(self, playback_q, output_q, error_save_list,
frames_per_buffer=MODEL_HOP_SAMPLES):
        super().__init__(daemon=True)
```

```
        self.playback_q = playback_q
        self.output_q = output_q
        self.error_save_list = error_save_list
        self.running = True
        self.p = pyaudio.PyAudio()
        self.stream = self.p.open(format=PY_AUDIO_FORMAT,
channels=CHANNELS,
                                  rate=SAMPLE_RATE, output=True,
                                  frames_per_buffer=frames_per_buffer)

        # FIXED: Use accumulator for proper chunk alignment
        self.input_accumulator = np.array([], dtype=np.int16)
        self.input_timestamps = []
        self.last_played = np.zeros(frames_per_buffer, dtype=np.int16)
        self.packet_count = 0  # For debugging

    def accumulate_input_to_model_size(self):
        """FIXED: Accumulate 256-sample ESP32 chunks into 1600-sample model
chunks"""
        try:
            while len(self.input_accumulator) < MODEL_HOP_SAMPLES:
                input_data, input_timestamp = self.playback_q.get_nowait()
                input_int16 = np.frombuffer(input_data,
dtype='<i2').astype(np.int16)

                self.input_accumulator =
np.concatenate([self.input_accumulator, input_int16])
                self.input_timestamps.append(input_timestamp)

        except queue.Empty:
            pass

        # Return a chunk if we have enough samples
        if len(self.input_accumulator) >= MODEL_HOP_SAMPLES:
            chunk = self.input_accumulator[:MODEL_HOP_SAMPLES].copy()
            # Use the timestamp from the first sample in the chunk
            chunk_timestamp = self.input_timestamps[0] if
self.input_timestamps else time.time()

            # Remove processed samples
            self.input_accumulator =
self.input_accumulator[MODEL_HOP_SAMPLES:]
            # Keep only relevant timestamps (approximate)
            samples_per_timestamp = ESP32_BUFFER_SIZE
            timestamps_to_remove = MODEL_HOP_SAMPLES //
samples_per_timestamp
            self.input_timestamps =
self.input_timestamps[timestamps_to_remove:]

            return chunk, chunk_timestamp

        return None, None

    def run(self):
        while self.running:
            # FIXED: Accumulate input to match model output chunk size
            input_chunk, input_timestamp =
self.accumulate_input_to_model_size()

            # Get model output
            try:
```

```
                output_chunk, output_timestamp =
self.output_q.get(timeout=0.1)
            except queue.Empty:
                continue

            if input_chunk is None:
                continue  # Not enough input accumulated yet

            self.packet_count += 1

            # FIXED: Correct error signal calculation for model that
outputs -x(n)
            # Since model outputs -x(n), we want: e(n) = x(n) + y(n) = x(n)
+ (-x(n)) = 0
            error_signal = input_chunk.astype(np.float32) +
output_chunk.astype(np.float32)
            error_int16 = np.clip(error_signal, -32768,
32767).astype(np.int16)


            # Save error signal
            self.error_save_list.append(error_int16.copy())

            # FIXED: Minimal processing to reduce artifacts
            # Reduce gain to prevent clipping/distortion
            error = error_int16.astype(np.float32)

            to_play = np.clip(0.4*error, -32768, 32767).astype(np.int16)

            self.last_played = to_play.copy()

            try:
                self.stream.write(to_play.tobytes())
            except Exception as e:
                print("Error playback error:", e)

    def stop(self):
        self.running = False
        time.sleep(0.05)
        try:
            if self.stream.is_active():
                self.stream.stop_stream()
            self.stream.close()
        except:
            pass
        try:
            self.p.terminate()
        except:
            pass

# ----------------------------
# Keyboard input handler - UNCHANGED
# ----------------------------
def keyboard_handler():
    global current_mode
    print("\nKeyboard commands:")
    print("  'r' - ANC OFF (silence)")
    print("  'a' - ANC ON")
    print("  'q' - Quit")

    while True:
```

```
        try:
            if sys.platform == "win32":
                import msvcrt
                if msvcrt.kbhit():
                    key = msvcrt.getch().decode('utf-8').lower()
                else:
                    time.sleep(0.1)
                    continue
            else:
                # Unix-like systems
                key = input().strip().lower()

            if key == 'r':
                with mode_lock:
                    current_mode = MODE_OFF
                print("ANC OFF - Speaker silenced")
            elif key == 'a':
                with mode_lock:
                    current_mode = MODE_ON
                print(f"ANC ON, Latency: {latency_samples} samples
({latency_samples/SAMPLE_RATE*1000:.1f}ms)")
            elif key == 'q':
                break

        except (KeyboardInterrupt, EOFError):
            break
        except:
            time.sleep(0.1)

# ----------------------------
# Main - UNCHANGED
# ----------------------------
def main():
    global current_mode, latency_samples

    # Sanity checks
    if not os.path.exists(MODEL_JIT_PATH):
        print("ERROR: MODEL_JIT_PATH not found:", MODEL_JIT_PATH)
        return

    device = torch.device('cpu')
    print("Loading TorchScript model (CPU)...")
    jit_model = torch.jit.load(MODEL_JIT_PATH, map_location='cpu')
    jit_model.eval()

    # Serial port
    try:
        ser = serial.Serial(COM_PORT, BAUD_RATE, timeout=1)
    except Exception as e:
        print(f"Error opening serial port {COM_PORT}: {e}")
        return

    # Queues
    playback_q = queue.Queue(maxsize=PLAYBACK_Q_MAX)
    model_q = queue.Queue(maxsize=MODEL_Q_MAX)
    output_q = queue.Queue(maxsize=OUTPUT_Q_MAX)

    # Lists for saving WAVs
    input_save_list = []
    output_save_list = []
    error_save_list = []
```

```
    # Threads
    reader = SerialReader(ser, playback_q, model_q)
    model_worker = ModelWorker(jit_model, model_q, output_q,
input_save_list, output_save_list, device='cpu')
    error_player = None

    # Start threads
    reader.start()
    model_worker.start()

    print(f"System started - Mode: {current_mode}")
    print("Keep volume low on first run!")

    # Start keyboard handler in separate thread
    kb_thread = threading.Thread(target=keyboard_handler, daemon=True)
    kb_thread.start()

    def cleanup_and_save(signum=None, frame=None):
        nonlocal ser, reader, model_worker, error_player
        print("\nStopping threads...")
        try:
            reader.stop()
        except:
            pass
        try:
            model_worker.stop()
        except:
            pass
        try:
            if error_player:
                error_player.stop()
        except:
            pass

        # give threads a moment to stop
        time.sleep(0.3)

        try:
            ser.close()
        except:
            pass

        print("Saving WAV files...")

        try:
            if len(input_save_list) > 0:
                in_all = np.concatenate(input_save_list).astype(np.int16)
                save_wav_int16(OUTPUT_INPUT_WAV, in_all, SAMPLE_RATE)
                print(f"Saved raw input -> {OUTPUT_INPUT_WAV}")
            else:
                print("No input audio recorded.")
        except Exception as e:
            print("Failed saving input WAV:", e)

        try:
            if len(output_save_list) > 0:
                out_all = np.concatenate(output_save_list).astype(np.int16)
                save_wav_int16(OUTPUT_MODEL_WAV, out_all, SAMPLE_RATE)
                print(f"Saved model output -> {OUTPUT_MODEL_WAV}")
            else:
```

```
                print("No model output recorded.")
        except Exception as e:
            print("Failed saving model output WAV:", e)

        try:
            if len(error_save_list) > 0:
                err_all = np.concatenate(error_save_list).astype(np.int16)
                save_wav_int16(OUTPUT_ERROR_WAV, err_all, SAMPLE_RATE)
                print(f"Saved error signal -> {OUTPUT_ERROR_WAV}")
            else:
                print("No error signal recorded.")
        except Exception as e:
            print("Failed saving error WAV:", e)

        print(f"Final latency measurement: {latency_samples} samples
({latency_samples/SAMPLE_RATE*1000:.1f}ms)")
        print("Done.")
        # force exit
        os._exit(0)

    signal.signal(signal.SIGINT, cleanup_and_save)

    # Mode switching logic
    active_player = None
    try:
        while True:
            # If user closed keyboard thread (pressed 'q'), then exit
            if not kb_thread.is_alive():
                print("Keyboard thread ended — exiting.")
                cleanup_and_save()
                break

            with mode_lock:
                mode = current_mode

            if mode == MODE_OFF:
                if active_player == 'error':
                    try:
                        if error_player:
                            error_player.stop()
                    except:
                        pass
                    active_player = None

            elif mode == MODE_ON:
                if active_player != 'error':
                    if error_player is None or not error_player.is_alive():
                        error_player = ErrorPlaybackThread(playback_q,
output_q, error_save_list, frames_per_buffer=MODEL_HOP_SAMPLES)
                        error_player.start()
                    active_player = 'error'

            time.sleep(0.2)
    except KeyboardInterrupt:
        cleanup_and_save()
    except Exception as e:
        print("Fatal error in main loop:", e)
        cleanup_and_save()

if __name__ == "__main__":
    main()import os
```

```
import numpy as np
import scipy.signal as signal
import librosa
import h5py
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from pathlib import Path
try:
    import soundfile as sf
except ImportError as e:
    raise ImportError("Failed to import soundfile. Please install it using
'pip install soundfile'.")

# Configuration
SAMPLE_RATE = 16000  # 16 kHz, as per the paper
WINDOW_SIZE = 0.02   # 20 ms
FRAME_SHIFT = 0.01   # 10 ms
FFT_SIZE = 320       # 320-point STFT
TEST_SPLIT = 6000     # Number of test samples
OUTPUT_PATH =
r"C:\Users\21321024\Desktop\Labib_Transfer\Dataset\final_final_dataset.h5"
AUDIO_FOLDERS =
[r"C:\Users\21321024\Desktop\Mehdi_Transfer\Audio\audioCollection"]
RIR_FILE = r"C:\Users\21321024\Desktop\Labib_Transfer\rir\t60rirs.h5"
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")

def load_rir(rir_file):
    """Load primary and secondary path RIRs from .h5 file."""
    try:
        with h5py.File(rir_file, 'r') as f:
            if 'primary/rir_0' not in f or 'secondary/rir_0' not in f:
                raise KeyError("Expected 'primary/rir_0' and
'secondary/rir_0' datasets in RIR file")
            primary_rir = np.array(f['primary/rir_0'])  # Shape:
(variable,)
            secondary_rir = np.array(f['secondary/rir_0'])  # Shape:
(variable,)
            # Warn about short RIR lengths
            primary_duration = len(primary_rir) / SAMPLE_RATE
            secondary_duration = len(secondary_rir) / SAMPLE_RATE
            if primary_duration < 0.05 or secondary_duration < 0.05:
                print(f"Warning: RIR durations are short (primary:
{primary_duration:.5f}s, "
                      f"secondary: {secondary_duration:.5f}s). Typical T60
for small enclosures "
                      f"is 0.05–0.15s. Verify RIRs capture full impulse
response.")
        return primary_rir, secondary_rir
    except Exception as e:
        raise ValueError(f"Error loading RIR file {rir_file}: {str(e)}")

def compute_spectrogram(audio, sr, window_size, frame_shift, fft_size):
    """Compute real and imaginary spectrograms for an audio signal."""
    try:
        window_samples = int(window_size * sr)  # 320 samples
        hop_samples = int(frame_shift * sr)     # 160 samples
        stft = librosa.stft(
            audio,
            n_fft=fft_size,
```

```
            win_length=window_samples,
            hop_length=hop_samples,
            window='hann'
        )  # Shape: (161, T)
        real_spec = np.real(stft)  # Real part
        imag_spec = np.imag(stft)  # Imaginary part
        return real_spec, imag_spec  # Each of shape (161, T)
    except Exception as e:
        raise ValueError(f"Error computing spectrogram: {str(e)}")

def process_audio_file(audio_path, sr=16000):
    """Process a single audio file to generate input and target
spectrograms."""
    try:
        # Load audio
        audio, file_sr = sf.read(audio_path)
        # Convert to mono if stereo
        if audio.ndim > 1:
            audio = np.mean(audio, axis=1)  # Average across channels
        if file_sr != sr:
            audio = librosa.resample(audio, orig_sr=file_sr, target_sr=sr)
        if len(audio) != 1 * sr:  # Ensure 1-second length
            audio = audio[:1*sr] if len(audio) > 1*sr else np.pad(audio,
(0, 1*sr - len(audio)))

        # Input: Spectrograms of reference signal x(n)
        ref_real, ref_imag = compute_spectrogram(audio, sr, WINDOW_SIZE,
FRAME_SHIFT, FFT_SIZE)

        # Target: Spectrograms of negated signal -x(n)
        negated_audio = -1*audio  # Simply negate the input
        target_real, target_imag = compute_spectrogram(negated_audio, sr,
WINDOW_SIZE, FRAME_SHIFT, FFT_SIZE)

        # Stack real and imaginary parts
        input_spec = np.stack([ref_real, ref_imag], axis=0)  # Shape: (2,
161, 101)
        target_spec = np.stack([target_real, target_imag], axis=0)  #
Shape: (2, 161, 101)

        return input_spec, target_spec
    except Exception as e:
        raise ValueError(f"Error processing audio file {audio_path}:
{str(e)}")

def create_dataset(audio_folders, output_path, test_split=6000,
max_files=101000):
    """Create HDF5 dataset from audio folders - simplified version without
RIR."""
    # Validate paths
    for folder in audio_folders:
        if not Path(folder).exists():
            raise ValueError(f"Audio folder does not exist: {folder}")

    # Ensure output directory exists
    Path(output_path).parent.mkdir(parents=True, exist_ok=True)

    # Collect audio files
    audio_files = []
    for folder in audio_folders:
        folder_path = Path(folder)
```

```
        files = [f for f in folder_path.glob("*.wav")]
        print(f"Scanning folder: {folder_path}, found {len(files)} .wav
files")
        audio_files.extend(files)

    if not audio_files:
        raise ValueError(f"No .wav files found in folders:
{audio_folders}")
    if len(audio_files) < max_files:
        print(f"Warning: Found {len(audio_files)} files, expected at least
{max_files}. Proceeding with available files.")
    audio_files = audio_files[:max_files]

    # Shuffle and split into train and test
    np.random.shuffle(audio_files)
    test_files = audio_files[:test_split]
    train_files = audio_files[test_split:]

    # Create HDF5 file
    try:
        with h5py.File(output_path, 'w') as h5f:
            # Store metadata
            h5f.attrs['sample_rate'] = SAMPLE_RATE
            h5f.attrs['description'] = 'Dataset with x(n) as input and -
x(n) as target'

            # Define fixed shape for spectrograms (T=101 for 1s audio)
            input_shape = (2, 161, 101)
            chunk_shape = (1, 2, 161, 101)

            # Initialize datasets
            train_inputs = h5f.create_dataset('train/inputs', shape=(0,) +
input_shape,
                                              maxshape=(None,) +
input_shape, dtype=np.float32,
                                              chunks=chunk_shape)
            train_targets = h5f.create_dataset('train/targets', shape=(0,)
+ input_shape,
                                               maxshape=(None,) +
input_shape, dtype=np.float32,
                                               chunks=chunk_shape)
            test_inputs = h5f.create_dataset('test/inputs', shape=(0,) +
input_shape,
                                             maxshape=(None,) + input_shape,
dtype=np.float32,
                                             chunks=chunk_shape)
            test_targets = h5f.create_dataset('test/targets', shape=(0,) +
input_shape,
                                              maxshape=(None,) +
input_shape, dtype=np.float32,
                                              chunks=chunk_shape)

            # Process training files
            skipped_files = []
            for i, audio_path in enumerate(train_files):
                try:
                    input_spec, target_spec =
process_audio_file(audio_path)  # Removed RIR parameter
                    if input_spec.shape != input_shape or target_spec.shape
!= input_shape:
```

```
                        print(f"Warning: Skipping {audio_path} due to shape
mismatch. "
                              f"Expected {input_shape}, got
{input_spec.shape}")
                        skipped_files.append(audio_path)
                        continue
                    train_inputs.resize(train_inputs.shape[0] + 1, axis=0)
                    train_targets.resize(train_targets.shape[0] + 1,
axis=0)
                    train_inputs[-1] = input_spec
                    train_targets[-1] = target_spec
                    if (i + 1) % 100 == 0:
                        print(f"Processed {i + 1}/{len(train_files)}
training files")
                except ValueError as e:
                    print(f"Warning: Skipping {audio_path} due to error:
{str(e)}")
                    skipped_files.append(audio_path)
                    continue

            # Process test files
            for i, audio_path in enumerate(test_files):
                try:
                    input_spec, target_spec =
process_audio_file(audio_path)  # Removed RIR parameter
                    if input_spec.shape != input_shape or target_spec.shape
!= input_shape:
                        print(f"Warning: Skipping {audio_path} due to shape
mismatch. "
                              f"Expected {input_shape}, got
{input_spec.shape}")
                        skipped_files.append(audio_path)
                        continue
                    test_inputs.resize(test_inputs.shape[0] + 1, axis=0)
                    test_targets.resize(test_targets.shape[0] + 1, axis=0)
                    test_inputs[-1] = input_spec
                    test_targets[-1] = target_spec
                    if (i + 1) % 10 == 0:
                        print(f"Processed {i + 1}/{len(test_files)} test
files")
                except ValueError as e:
                    print(f"Warning: Skipping {audio_path} due to error:
{str(e)}")
                    skipped_files.append(audio_path)
                    continue

            # Log final statistics
            print(f"Final dataset shapes:")
            print(f"Train inputs: {train_inputs.shape}")
            print(f"Train targets: {train_targets.shape}")
            print(f"Test inputs: {test_inputs.shape}")
            print(f"Test targets: {test_targets.shape}")

            # Log skipped files to a text file
            if skipped_files:
                with open('skipped_files.txt', 'w') as f:
                    for file in skipped_files:
                        f.write(str(file) + '\n')
                print(f"Completed processing. Skipped {len(skipped_files)}
files, logged to skipped_files.txt")
    except Exception as e:
```

```
        raise ValueError(f"Error creating HDF5 dataset at {output_path}:
{str(e)}")

def main():
    print("Creating simplified dataset (x(n) -> -x(n))...")
    try:
        create_dataset(AUDIO_FOLDERS, OUTPUT_PATH, TEST_SPLIT)
        print("Dataset created successfully.")
    except Exception as e:
        print(f"Failed to create dataset: {str(e)}")
if __name__ == "__main__":
    main()
```

A.3 Training Code

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
import h5py
import numpy as np
import os
from tqdm import tqdm
import time
import torch.nn.functional as F

# Configuration matching the paper and dataset
CONFIG = {
    'DATASET_PATH':
r"C:\Users\21321024\Desktop\Labib_Transfer\Dataset\final_dataset.h5",
    'MODEL_SAVE_PATH': r"C:\Users\21321024\Desktop\Labib_Transfer\model",
    'BATCH_SIZE': 32,
    'EPOCHS': 30,
    'LEARNING_RATE': 1e-3,  # Paper uses 0.001
    'SAMPLE_RATE': 16000,
    'N_FFT': 320,  # Paper uses 320-point STFT
    'HOP_LENGTH': 160,  # 10ms frame shift at 16kHz
    'WIN_LENGTH': 320,  # 20ms window at 16kHz
    'WINDOW_DURATION': 1,  # 1 second
    'INPUT_CHANNELS': 2,  # I=1 reference mic (real + imaginary)
    'OUTPUT_CHANNELS': 2,  # J=1 loudspeaker (real + imaginary)
    'FREQ_BINS': 161,  # (320//2 + 1) frequency bins
    'TIME_FRAMES': 101,  # Number of time frames for 1 second
    'DEVICE': torch.device("cuda" if torch.cuda.is_available() else "cpu")
}

class TimeDomainDataset(Dataset):
    """
    Dataset that loads spectrograms but converts them to time domain for
end-to-end training.
    """
    def __init__(self, hdf5_file_path, dataset_type='train'):
        self.hdf5_file_path = hdf5_file_path
        self.dataset_type = dataset_type
        self.file = h5py.File(hdf5_file_path, 'r')

        # Load dataset dimensions
        if dataset_type == 'train':
            self.inputs = self.file['train/inputs']
```

```
            self.targets = self.file['train/targets']
        else:
            self.inputs = self.file['test/inputs']
            self.targets = self.file['test/targets']
        self.length = self.inputs.shape[0]

        # Precompute Hann window for ISTFT
        self.window = torch.hann_window(CONFIG['WIN_LENGTH'])

    def spectrogram_to_time(self, spectrogram):
        """
        Convert complex spectrogram back to time domain using ISTFT.
        Input: (2, 161, 101) - [real, imag, freq_bins, time_frames]
        Output: (16000,) - time domain signal
        """
        real_part = spectrogram[0]  # (161, 101)
        imag_part = spectrogram[1]  # (161, 101)

        # Create complex spectrogram
        complex_spec = torch.complex(real_part, imag_part)  # (161, 101)

        # ISTFT with overlap-add reconstruction
        signal_length = CONFIG['SAMPLE_RATE']  # 16000 samples for 1 second
        time_signal = torch.zeros(signal_length)

        window = self.window

        for t in range(complex_spec.shape[1]):  # 101 time frames
            # IFFT to get time domain frame
            frame = torch.fft.irfft(complex_spec[:, t],
n=CONFIG['N_FFT'])  # (320,)

            # Apply window
            frame = frame * window

            # Overlap-add
            start = t * CONFIG['HOP_LENGTH']
            end = start + CONFIG['WIN_LENGTH']
            if end <= signal_length:
                time_signal[start:end] += frame

        return time_signal

    def __len__(self):
        return self.length

    def __getitem__(self, idx):
        # Load spectrograms
        input_spec = torch.tensor(self.inputs[idx], dtype=torch.float32)  #
(2, 161, 101)
        target_spec = torch.tensor(self.targets[idx],
dtype=torch.float32)  # (2, 161, 101)

        # Convert to time domain
        input_time = self.spectrogram_to_time(input_spec)  # (16000,)
        target_time = self.spectrogram_to_time(target_spec)  # (16000,)

        return input_time, target_time

    def __del__(self):
        if hasattr(self, 'file'):
```

```
            self.file.close()

class STFTLayer(nn.Module):
    """
    Differentiable STFT layer for converting time domain to spectrogram.
    """
    def __init__(self, n_fft=320, hop_length=160, win_length=320):
        super(STFTLayer, self).__init__()
        self.n_fft = n_fft
        self.hop_length = hop_length
        self.win_length = win_length

        # Register Hann window as buffer (non-trainable)
        window = torch.hann_window(win_length)
        self.register_buffer('window', window)

    def forward(self, x):
        """
        Convert time domain signal to complex spectrogram.
        Input: (batch_size, signal_length) - time domain
        Output: (batch_size, 2, freq_bins, time_frames) - [real, imag]
        """
        # STFT
        stft_result = torch.stft(
            x,
            n_fft=self.n_fft,
            hop_length=self.hop_length,
            win_length=self.win_length,
            window=self.window,
            return_complex=True,
            pad_mode='constant'
        )  # (batch_size, freq_bins, time_frames)

        # Split into real and imaginary parts
        real_part = stft_result.real  # (batch_size, freq_bins,
time_frames)
        imag_part = stft_result.imag  # (batch_size, freq_bins,
time_frames)

        # Stack to get (batch_size, 2, freq_bins, time_frames)
        output = torch.stack([real_part, imag_part], dim=1)

        return output

class ISTFTLayer(nn.Module):
    """
    Differentiable ISTFT layer for converting spectrogram back to time
domain.
    """
    def __init__(self, n_fft=320, hop_length=160, win_length=320,
signal_length=16000):
        super(ISTFTLayer, self).__init__()
        self.n_fft = n_fft
        self.hop_length = hop_length
        self.win_length = win_length
        self.signal_length = signal_length

        # Register Hann window as buffer
        window = torch.hann_window(win_length)
        self.register_buffer('window', window)
```

```
    def forward(self, x):
        """
        Convert complex spectrogram back to time domain.
        Input: (batch_size, 2, freq_bins, time_frames) - [real, imag]
        Output: (batch_size, signal_length) - time domain
        """
        # Reconstruct complex spectrogram
        real_part = x[:, 0]  # (batch_size, freq_bins, time_frames)
        imag_part = x[:, 1]  # (batch_size, freq_bins, time_frames)
        complex_spec = torch.complex(real_part, imag_part)

        # ISTFT
        time_signal = torch.istft(
            complex_spec,
            n_fft=self.n_fft,
            hop_length=self.hop_length,
            win_length=self.win_length,
            window=self.window,
            length=self.signal_length
        )  # (batch_size, signal_length)

        return time_signal

class DeepMCANC_CRN(nn.Module):
    """
    End-to-end Convolutional Recurrent Network for Deep MCANC.
    Takes time domain input, processes in spectrogram domain, outputs time
domain.
    """
    def __init__(self, input_channels=2, output_channels=2):
        super(DeepMCANC_CRN, self).__init__()
        self.input_channels = input_channels
        self.output_channels = output_channels

        # STFT/ISTFT layers
        self.stft = STFTLayer(
            n_fft=CONFIG['N_FFT'],
            hop_length=CONFIG['HOP_LENGTH'],
            win_length=CONFIG['WIN_LENGTH']
        )
        self.istft = ISTFTLayer(
            n_fft=CONFIG['N_FFT'],
            hop_length=CONFIG['HOP_LENGTH'],
            win_length=CONFIG['WIN_LENGTH'],
            signal_length=CONFIG['SAMPLE_RATE']
        )

        # Encoder layers - adjusted for correct dimensions
        self.conv1 = nn.Conv2d(self.input_channels, 16, kernel_size=(3, 3),
stride=(2, 1), padding=(1, 1))
        self.bn1 = nn.BatchNorm2d(16)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=(3, 3), stride=(2, 1),
padding=(1, 1))
        self.bn2 = nn.BatchNorm2d(32)
        self.conv3 = nn.Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 1),
padding=(1, 1))
        self.bn3 = nn.BatchNorm2d(64)
        self.conv4 = nn.Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 1),
padding=(1, 1))
        self.bn4 = nn.BatchNorm2d(128)
```

```
        self.conv5 = nn.Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 1),
padding=(1, 1))
        self.bn5 = nn.BatchNorm2d(256)

        # Calculate LSTM input size based on encoder output
        # After 5 conv layers with stride=(2,1): 161 -> 81 -> 41 -> 21 ->
11 -> 6
        self.lstm_input_size = 256 * 6  # 1536

        # LSTM layers (2 layers)
        self.lstm1 = nn.LSTM(self.lstm_input_size, self.lstm_input_size,
num_layers=1, batch_first=True)
        self.lstm2 = nn.LSTM(self.lstm_input_size, self.lstm_input_size,
num_layers=1, batch_first=True)

        # Decoder layers with skip connections
        self.deconv5 = nn.ConvTranspose2d(512, 128, kernel_size=(3, 3),
stride=(2, 1), padding=(1, 1), output_padding=(0, 0))
        self.dbn5 = nn.BatchNorm2d(128)
        self.deconv4 = nn.ConvTranspose2d(256, 64, kernel_size=(3, 3),
stride=(2, 1), padding=(1, 1), output_padding=(0, 0))
        self.dbn4 = nn.BatchNorm2d(64)
        self.deconv3 = nn.ConvTranspose2d(128, 32, kernel_size=(3, 3),
stride=(2, 1), padding=(1, 1), output_padding=(0, 0))
        self.dbn3 = nn.BatchNorm2d(32)
        self.deconv2 = nn.ConvTranspose2d(64, 16, kernel_size=(3, 3),
stride=(2, 1), padding=(1, 1), output_padding=(0, 0))
        self.dbn2 = nn.BatchNorm2d(16)
        self.deconv1 = nn.ConvTranspose2d(32, self.output_channels,
kernel_size=(3, 3), stride=(2, 1), padding=(1, 1), output_padding=(0, 0))

        self.elu = nn.ELU()
        self.dropout = nn.Dropout(0.1)

    def forward(self, x):
        """
        Forward pass: time domain -> spectrogram -> processing -> time
domain
        Input: (batch_size, signal_length) - time domain signal
        Output: (batch_size, signal_length) - time domain signal (negated)
        """
        # Convert time domain to spectrogram
        spec_input = self.stft(x)  # (batch_size, 2, 161, time_frames)

        # Encoder path with skip connections
        e1 = self.elu(self.bn1(self.conv1(spec_input)))  # (batch, 16, 81,
time_frames)
        e2 = self.elu(self.bn2(self.conv2(e1)))  # (batch, 32, 41,
time_frames)
        e3 = self.elu(self.bn3(self.conv3(e2)))  # (batch, 64, 21,
time_frames)
        e4 = self.elu(self.bn4(self.conv4(e3)))  # (batch, 128, 11,
time_frames)
        e5 = self.elu(self.bn5(self.conv5(e4)))  # (batch, 256, 6,
time_frames)

        # Reshape for LSTM
        batch_size, channels, freq_bins, time_steps = e5.shape
        lstm_input = e5.permute(0, 3, 1, 2).reshape(batch_size, time_steps,
channels * freq_bins)
```

```
        # LSTM layers
        lstm_out1, _ = self.lstm1(lstm_input)
        lstm_out1 = self.dropout(lstm_out1)
        lstm_out2, _ = self.lstm2(lstm_out1)

        # Reshape back for decoder
        lstm_out = lstm_out2.reshape(batch_size, time_steps, channels,
freq_bins).permute(0, 2, 3, 1)

        # Decoder path with skip connections
        d5 = self.elu(self.dbn5(self.deconv5(torch.cat([e5, lstm_out],
dim=1))))
        d4 = self.elu(self.dbn4(self.deconv4(torch.cat([e4, d5], dim=1))))
        d3 = self.elu(self.dbn3(self.deconv3(torch.cat([e3, d4], dim=1))))
        d2 = self.elu(self.dbn2(self.deconv2(torch.cat([e2, d3], dim=1))))
        spec_output = self.deconv1(torch.cat([e1, d2], dim=1))  # (batch,
2, 161, time_frames)

        # Convert spectrogram back to time domain
        time_output = self.istft(spec_output)  # (batch_size,
signal_length)

        return time_output

class TimeDomainMSELoss(nn.Module):
    """
    Time domain MSE loss for end-to-end training.
    """
    def __init__(self):
        super(TimeDomainMSELoss, self).__init__()
        self.mse = nn.MSELoss()

    def forward(self, predicted, target):
        return self.mse(predicted, target)

def train_deep_mcanc():
    """
    Training function for Deep MCANC model
    """
    print(f"🚀 Starting Deep MCANC training on {CONFIG['DEVICE']}")

    # Create model save directory
    os.makedirs(CONFIG['MODEL_SAVE_PATH'], exist_ok=True)

    # Load datasets (now returns time domain signals)
    train_dataset = TimeDomainDataset(CONFIG['DATASET_PATH'], 'train')
    test_dataset = TimeDomainDataset(CONFIG['DATASET_PATH'], 'test')

    train_loader = DataLoader(
        train_dataset,
        batch_size=CONFIG['BATCH_SIZE'],
        shuffle=True,
        num_workers=0  # Set to 0 for Windows/HDF5 compatibility
    )

    test_loader = DataLoader(
        test_dataset,
        batch_size=CONFIG['BATCH_SIZE'],
        shuffle=False,
        num_workers=0
    )
```

```
    print(f"📊 Training samples: {len(train_dataset)}")
    print(f"📊 Test samples: {len(test_dataset)}")
    print(f"📊 Time domain signal length: {CONFIG['SAMPLE_RATE']}
samples")

    # Initialize model
    model = DeepMCANC_CRN(
        input_channels=CONFIG['INPUT_CHANNELS'],
        output_channels=CONFIG['OUTPUT_CHANNELS']
    ).to(CONFIG['DEVICE'])

    # Initialize loss function (time domain MSE)
    loss_fn = TimeDomainMSELoss()

    # Initialize optimizer
    optimizer = optim.Adam(
        model.parameters(),
        lr=CONFIG['LEARNING_RATE'],
        amsgrad=True
    )

    # Add learning rate scheduler
    scheduler = optim.lr_scheduler.ReduceLROnPlateau(
        optimizer, mode='min', factor=0.7, patience=3, verbose=True
    )

    print(f"🏗️ Model parameters: {sum(p.numel() for p in
model.parameters()):,}")

    # Training loop
    best_loss = float('inf')
    timestamp = time.strftime("%Y%m%d_%H%M%S")

    # Test model with a single batch to check dimensions
    print("🔍 Testing model architecture with sample batch...")
    sample_batch = next(iter(train_loader))
    sample_input, sample_target = sample_batch[0][:1].to(CONFIG['DEVICE']),
sample_batch[1][:1].to(CONFIG['DEVICE'])
    print(f"Sample input shape: {sample_input.shape}")
    print(f"Sample target shape: {sample_target.shape}")

    with torch.no_grad():
        try:
            sample_output = model(sample_input)
            print(f"Sample output shape: {sample_output.shape}")
            print("☑ Model architecture test passed!")
        except Exception as e:
            print(f"❌ Model architecture test failed: {e}")
            return None

    for epoch in range(CONFIG['EPOCHS']):
        # Training phase
        model.train()
        train_loss = 0.0
        train_pbar = tqdm(train_loader, desc=f"Epoch
{epoch+1}/{CONFIG['EPOCHS']} [Train]")

        for batch_idx, (inputs, targets) in enumerate(train_pbar):
            inputs = inputs.to(CONFIG['DEVICE'])
```

```
            targets = targets.to(CONFIG['DEVICE'])

            optimizer.zero_grad()
            outputs = model(inputs)
            loss = loss_fn(outputs, targets)
            loss.backward()

            # Gradient clipping for stability
            torch.nn.utils.clip_grad_norm_(model.parameters(),
max_norm=1.0)

            optimizer.step()

            train_loss += loss.item()
            train_pbar.set_postfix({
                'Loss': f"{loss.item():.6f}",
                'Avg Loss': f"{train_loss/(batch_idx+1):.6f}"
            })

        avg_train_loss = train_loss / len(train_loader)

        # Validation phase
        model.eval()
        val_loss = 0.0
        with torch.no_grad():
            val_pbar = tqdm(test_loader, desc=f"Epoch
{epoch+1}/{CONFIG['EPOCHS']} [Val]")
            for inputs, targets in val_pbar:
                inputs = inputs.to(CONFIG['DEVICE'])
                targets = targets.to(CONFIG['DEVICE'])
                outputs = model(inputs)
                loss = loss_fn(outputs, targets)
                val_loss += loss.item()
                val_pbar.set_postfix({'Val Loss': f"{loss.item():.6f}"})

        avg_val_loss = val_loss / len(test_loader)

        # Update learning rate
        scheduler.step(avg_val_loss)

        print(f"Epoch {epoch+1}/{CONFIG['EPOCHS']}: "
              f"Train Loss: {avg_train_loss:.6f}, "
              f"Val Loss: {avg_val_loss:.6f}")

        # Save best model
        if avg_val_loss < best_loss:
            best_loss = avg_val_loss
            model_path = os.path.join(CONFIG['MODEL_SAVE_PATH'],
f'deep_mcanc_best_{timestamp}.pth')
            torch.save({
                'epoch': epoch + 1,
                'model_state_dict': model.state_dict(),
                'optimizer_state_dict': optimizer.state_dict(),
                'scheduler_state_dict': scheduler.state_dict(),
                'train_loss': avg_train_loss,
                'val_loss': avg_val_loss,
                'config': CONFIG,
                'model_class': 'DeepMCANC_CRN'
            }, model_path)
            print(f"☑ Best model saved: {model_path}")
```

```
    # Save final model
    final_model_path = os.path.join(CONFIG['MODEL_SAVE_PATH'],
f'deep_mcanc_final_{timestamp}.pth')
    torch.save({
        'epoch': CONFIG['EPOCHS'],
        'model_state_dict': model.state_dict(),
        'optimizer_state_dict': optimizer.state_dict(),
        'scheduler_state_dict': scheduler.state_dict(),
        'train_loss': avg_train_loss,
        'val_loss': avg_val_loss,
        'config': CONFIG,
        'model_class': 'DeepMCANC_CRN'
    }, final_model_path)

    print(f"🎯 Training completed! Final model saved: {final_model_path}")
    print(f"📈 Best validation loss: {best_loss:.6f}")

    return model

def load_trained_model(model_path):
    """
    Load a trained Deep MCANC model for deployment
    """
    checkpoint = torch.load(model_path, map_location=CONFIG['DEVICE'])
    model = DeepMCANC_CRN(
        input_channels=checkpoint['config']['INPUT_CHANNELS'],
        output_channels=checkpoint['config']['OUTPUT_CHANNELS']
    )
    model.load_state_dict(checkpoint['model_state_dict'])
    model.to(CONFIG['DEVICE'])
    model.eval()
    print(f" Model loaded from: {model_path}")
    print(f" Trained for {checkpoint['epoch']} epochs")
    print(f"Final validation loss: {checkpoint['val_loss']:.6f}")
    return model

def test_negation_property(model, test_loader, num_samples=5):
    """
    Test if the model has learned the negation property: y(n) = -x(n) in
time domain
    """
    print("Testing time-domain negation property...")
    model.eval()

    with torch.no_grad():
        for i, (inputs, targets) in enumerate(test_loader):
            if i >= num_samples:
                break

            inputs = inputs.to(CONFIG['DEVICE'])
            targets = targets.to(CONFIG['DEVICE'])
            outputs = model(inputs)

            # Check if outputs are approximately negatives of inputs
            negated_inputs = -inputs
            mse_with_negation = torch.mean((outputs - negated_inputs) **
2).item()
            mse_with_targets = torch.mean((outputs - targets) ** 2).item()

            # Calculate correlation between output and negated input
```

```
            correlation = torch.corrcoef(torch.stack([outputs.flatten(),
negated_inputs.flatten()]))[0, 1].item()

            print(f"Sample {i+1}:")
            print(f" MSE(output, -input): {mse_with_negation:.6f}")
            print(f" MSE(output, target): {mse_with_targets:.6f}")
            print(f" Correlation(output, -input): {correlation:.4f}")
            print(f" Phase shift accuracy: {(correlation + 1) *
50:.2f}%")  # Perfect negation = correlation of -1

if __name__ == "__main__":
    print("=" * 80)
    print(" DEEP MCANC: Multi-Channel Active Noise Control")
    print(" Goal: End-to-End x(n) -> -x(n) mapping with STFT/ISTFT")
    print(" Architecture: Time Domain -> STFT -> CRN -> ISTFT -> Time
Domain")
    print("=" * 80)
    trained_model = train_deep_mcanc()

    if trained_model is not None:
        print("\n Training completed successfully!")
        print(f" Models saved in: {CONFIG['MODEL_SAVE_PATH']}")

        # Test the negation property
        test_dataset = TimeDomainDataset(CONFIG['DATASET_PATH'], 'test')
        test_loader = DataLoader(test_dataset,
batch_size=CONFIG['BATCH_SIZE'], shuffle=False, num_workers=0)
        test_negation_property(trained_model, test_loader)

        print("\n To load the model for inference:")
        print("model =
load_trained_model('path/to/deep_mcanc_best_*.pth')")
        print("\n Usage: output_signal = model(input_signal)")
        print(" Input: (batch_size, 16000) - 1 second audio at 16kHz")
        print(" Output: (batch_size, 16000) - negated audio signal")
    else:
        print("Training failed due to architecture issues.")
```

A.4 Offline Test Code

```
import os
os.environ["KMP_DUPLICATE_LIB_OK"] = "TRUE"

import torch
import numpy as np
import librosa
import soundfile as sf
import matplotlib.pyplot as plt
from tqdm import tqdm
import math

# ==============================
# 🔧 Config
# ==============================
CONFIG = {
    'SAMPLE_RATE': 16000,
    'WINDOW_DURATION': 1,  # in seconds
    'DEVICE': torch.device("cuda" if torch.cuda.is_available() else "cpu")
}
```

```
# 🔧 Paths
MODEL_PATH =
r"C:\Users\21321024\Desktop\Labib_Transfer\TrainedModel\crn_e6.pt"  #
Update with your .pt file
SAVE_DIR = r"C:\Users\21321024\Desktop\Labib_Transfer\CRN_Test"
os.makedirs(SAVE_DIR, exist_ok=True)

# ==============================
# Offline Test Function
# ==============================
def offline_test_wav(wav_path):
    device = CONFIG['DEVICE']

    # Load scripted model from .pt file
    model = torch.jit.load(MODEL_PATH, map_location=device)
    model.eval()

    # Load and resample WAV (assumed to be x(n))
    x_np, sr = librosa.load(wav_path, sr=None)
    if sr != CONFIG['SAMPLE_RATE']:
        x_np = librosa.resample(x_np, orig_sr=sr,
target_sr=CONFIG['SAMPLE_RATE'])
    x = torch.FloatTensor(x_np).to(device)

    # Segment into 1s windows (50% overlap)
    window_samples = int(CONFIG['WINDOW_DURATION'] *
CONFIG['SAMPLE_RATE'])  # 16000
    total_samples = len(x)
    segments = []
    step = window_samples // 2  # 8000
    for start in range(0, total_samples, step):
        end = min(start + window_samples, total_samples)
        seg = x[start:end].cpu().numpy()
        if len(seg) < window_samples:
            seg = np.pad(seg, (0, window_samples - len(seg)),
mode='constant')
        segments.append(seg)

    # Process segments with overlap-add
    y = torch.zeros(total_samples, device=device)  # Anti-noise y(n)
    overlap_count = torch.zeros(total_samples, device=device)
    window = torch.hann_window(window_samples, device=device)
    with torch.no_grad():
        for i, seg in enumerate(tqdm(segments, desc="Processing
segments")):
            seg_tensor = torch.FloatTensor(seg).to(device)
            y_seg = model(seg_tensor.unsqueeze(0))  # (1, 16000)
            start = i * step
            end = min(start + window_samples, total_samples)
            y[start:end] += y_seg[0, :end-start] * window[:end-start]
            overlap_count[start:end] += window[:end-start]

    # Normalize overlap regions
    y = torch.where(overlap_count > 0, y / (overlap_count + 1e-10), y)
    y=1*y
    # Compute error signal: e(n) = x(n) + y(n)
    e = 5*(x[:len(y)] + y)

    # Noise reduction metrics
    rms_x = torch.sqrt((x ** 2).mean()).item()
    rms_e = torch.sqrt((e ** 2).mean()).item()
```

```
    nr_db = 20 * math.log10(rms_x / rms_e) if rms_e > 0 else float('inf')
    print(f"Noise reduction: {nr_db:.2f} dB")

    # Plot
    min_len = min(len(x), len(y), len(e))
    x_plot = x.cpu().numpy()[:min_len]
    y_plot = y.cpu().numpy()[:min_len]
    e_plot = e.cpu().numpy()[:min_len]

    plt.figure(figsize=(14, 8))
    plt.subplot(2, 1, 1)
    plt.plot(x_plot, label='Input x(n)', color='black', alpha=0.7)
    plt.plot(y_plot, label='Anti-Noise y(n)', color='green', alpha=0.7)
    plt.title('Input vs Anti-Noise (Should be Opposite)')
    plt.ylim([-0.6, 0.6])
    plt.grid(True)
    plt.legend()

    plt.subplot(2, 1, 2)
    plt.plot(x_plot, label='Input x(n)', color='black', alpha=0.7)
    plt.plot(e_plot, label='Error e(n)', color='red', alpha=0.7)
    plt.title('Error Signal (Should be Near Zero)')
    plt.ylim([-0.6, 0.6])
    plt.grid(True)
    plt.legend()

    plt.tight_layout()
    plt.savefig(os.path.join(SAVE_DIR, 'offline_test_results.png'))
    plt.show()

    # Save results
    sf.write(os.path.join(SAVE_DIR, 'x_input.wav'), x.cpu().numpy(),
CONFIG['SAMPLE_RATE'])
    sf.write(os.path.join(SAVE_DIR, 'y_anti_noise.wav'), y.cpu().numpy(),
CONFIG['SAMPLE_RATE'])
    sf.write(os.path.join(SAVE_DIR, 'e_error_signal.wav'), e.cpu().numpy(),
CONFIG['SAMPLE_RATE'])

    print("Offline test completed. Results saved to:", SAVE_DIR)

# ==============================
# Main
# ==============================
if __name__ == "__main__":
    wav_file =
r"C:\Users\21321024\Desktop\Presentation\Basic\test1_dhaka_sound.wav"
    offline_test_wav(wav_file)
```

A.5 Real-Time Deployment Code

```
import os
import time
import struct
import queue
import threading
import wave
import signal
import sys
import numpy as np
import serial
import pyaudio
```

```
import torch
from collections import deque
from scipy.signal import butter, lfilter

# ----------------------------
# CONFIG
# ----------------------------
COM_PORT = "COM9"  # ESP32 serial port
BAUD_RATE = 500000  # Must match ESP32
MODEL_JIT_PATH = r"C:\Users\barrl\Downloads\crn_e1.pt"  # TorchScript file
OUTPUT_INPUT_WAV = r"D:\TestFIles\CRN\esp32_raw_input.wav"
OUTPUT_MODEL_WAV = r"D:\TestFIles\CRN\mcanc_model_output.wav"
OUTPUT_ERROR_WAV = r"D:\TestFIles\CRN\error_signal.wav"
SAMPLE_RATE = 16000
CHANNELS = 1
PY_AUDIO_FORMAT = pyaudio.paInt16

# ESP32 framing
ESP32_BUFFER_SIZE = 256  # Samples per frame from ESP32
ESP32_PAYLOAD_BYTES = ESP32_BUFFER_SIZE * 2  # 16-bit samples

# Model input/output
MODEL_WINDOW_SAMPLES = 16000  # 1 second at 16 kHz
MODEL_HOP_SAMPLES = 1600  # 0.1 seconds for streaming output

# Queue sizes - increased to prevent drops
PLAYBACK_Q_MAX = 32
MODEL_Q_MAX = 64
OUTPUT_Q_MAX = 32

# Playback modes
MODE_OFF = 'off'
MODE_ON = 'on'

# ----------------------------
# Global state
# ----------------------------
current_mode = MODE_OFF
latency_samples = 0
mode_lock = threading.Lock()

# ----------------------------
# Utility functions
# ----------------------------
def save_wav_int16(path, int16_numpy_array, sample_rate=SAMPLE_RATE):
    os.makedirs(os.path.dirname(path), exist_ok=True)
    with wave.open(path, "wb") as wf:
        wf.setnchannels(CHANNELS)
        wf.setsampwidth(2)
        wf.setframerate(sample_rate)
        wf.writeframes(int16_numpy_array.tobytes())

def apply_crossfade(buffer1, buffer2, crossfade_samples=32):
    """Apply crossfade between two buffers to reduce clicks"""
    if len(buffer1) < crossfade_samples or len(buffer2) <
crossfade_samples:
        return buffer2

    result = buffer2.copy()
    fade_out = np.linspace(1.0, 0.0, crossfade_samples)
    fade_in = np.linspace(0.0, 1.0, crossfade_samples)
```

```
    result[:crossfade_samples] = (buffer1[-crossfade_samples:] * fade_out
+
                                  buffer2[:crossfade_samples] * fade_in)
    return result

# ----------------------------
# SerialReader
# ----------------------------
class SerialReader(threading.Thread):
    def __init__(self, ser, playback_q, model_q):
        super().__init__(daemon=True)
        self.ser = ser
        self.playback_q = playback_q
        self.model_q = model_q
        self.running = True
        self.packet_count = 0  # FIXED: Add packet counter for debugging

    def wait_for_header(self):
        state = 0
        while self.running:
            b = self.ser.read(1)
            if not b:
                return False
            if state == 0:
                if b == b'\xAA':
                    state = 1
            else:
                if b == b'\x55':
                    return True
                state = 1 if b == b'\xAA' else 0
        return False

    def run(self):
        while self.running:
            ok = self.wait_for_header()
            if not ok:
                continue
            size_bytes = self.ser.read(2)
            if len(size_bytes) < 2:
                continue
            payload_size = struct.unpack("<H", size_bytes)[0]
            audio_bytes = self.ser.read(payload_size)
            if len(audio_bytes) < payload_size:
                continue

            # Add timestamp for latency calculation
            timestamp = time.time()
            packet = (audio_bytes, timestamp)
            self.packet_count += 1

            # FIXED: Better queue management - don't drop packets
unnecessarily
            # For playback queue
            if not self.playback_q.full():
                self.playback_q.put(packet, block=False)
            else:
                if self.packet_count % 100 == 0:  # Log occasionally
                    print(f"Warning: Playback queue full, dropping packet
{self.packet_count}")
```

```
            # For model queue
            if not self.model_q.full():
                self.model_q.put(packet, block=False)
            else:
                if self.packet_count % 100 == 0:  # Log occasionally
                    print(f"Warning: Model queue full, dropping packet
{self.packet_count}")

    def stop(self):
        self.running = False

# ----------------------------
# ModelWorker
# ----------------------------
class ModelWorker(threading.Thread):
    def __init__(self, jit_model, model_q, output_q, input_save_list,
output_save_list, device='cpu'):
        super().__init__(daemon=True)
        self.model = jit_model
        self.device = device
        self.q = model_q
        self.output_q = output_q
        self.input_save_list = input_save_list
        self.output_save_list = output_save_list
        self.running = True
        self.feed_buf = np.zeros(0, dtype=np.float32)
        self.out_fifo = np.zeros(0, dtype=np.float32)
        self.window_samples = MODEL_WINDOW_SAMPLES
        self.hop_samples = MODEL_HOP_SAMPLES
        self.latency_measurements = deque(maxlen=10)  # Keep last 10
measurements

        # For smoother processing - reduce crossfade to minimize artifacts
        self.last_output = np.zeros(MODEL_HOP_SAMPLES, dtype=np.float32)

    def run(self):
        global latency_samples

        while self.running:
            try:
                audio_bytes, input_timestamp = self.q.get(timeout=0.5)
            except queue.Empty:
                continue

            # Convert bytes to int16 and save for WAV
            in_int16 = np.frombuffer(audio_bytes,
dtype='<i2').astype(np.int16)
            self.input_save_list.append(in_int16.copy())

            # Normalize to float32 [-1, 1)
            samples = in_int16.astype(np.float32) / 32768.0
            self.feed_buf = np.concatenate([self.feed_buf, samples])

            # Process 1-second windows when available
            while len(self.feed_buf) >= self.window_samples:
                window = self.feed_buf[:self.window_samples].copy()

                # Convert to torch tensor
                input_tensor =
torch.from_numpy(window).float().unsqueeze(0).to(self.device)
```

```
                # Measure inference time
                start_time = time.time()
                with torch.no_grad():
                    output_tensor = self.model(input_tensor)
                end_time = time.time()

                # Calculate latency in samples
                inference_time_sec = end_time - start_time
                inference_samples = int(inference_time_sec * SAMPLE_RATE)
                self.latency_measurements.append(inference_samples)

                # Update global latency with running average
                if len(self.latency_measurements) > 0:
                    latency_samples =
int(np.mean(self.latency_measurements))

                y_window =
output_tensor.squeeze(0).cpu().numpy().astype(np.float32)
                self.out_fifo = np.concatenate([self.out_fifo, y_window])
                self.feed_buf = self.feed_buf[self.hop_samples:]

            # Flush hop-sized chunks to output queue
            while len(self.out_fifo) >= self.hop_samples:
                to_play = self.out_fifo[:self.hop_samples].copy()
                self.out_fifo = self.out_fifo[self.hop_samples:]

                # Apply light crossfade for smoother transitions
                to_play = apply_crossfade(self.last_output, to_play,
crossfade_samples=32)
                self.last_output = to_play.copy()

                # Convert to int16
                int16_chunk = np.clip(to_play * 32767.0, -32768,
32767).astype(np.int16)
                self.output_save_list.append(int16_chunk.copy())

                try:
                    self.output_q.put((int16_chunk, input_timestamp),
block=False)
                except queue.Full:
                    # Remove oldest if full
                    try:
                        self.output_q.get_nowait()
                        self.output_q.put((int16_chunk, input_timestamp),
block=False)
                    except queue.Empty:
                        pass

    def stop(self):
        self.running = False

# ----------------------------
# ErrorPlaybackThread -
# ----------------------------
class ErrorPlaybackThread(threading.Thread):
    def __init__(self, playback_q, output_q, error_save_list,
frames_per_buffer=MODEL_HOP_SAMPLES):
        super().__init__(daemon=True)
        self.playback_q = playback_q
        self.output_q = output_q
        self.error_save_list = error_save_list
```

```
        self.running = True
        self.p = pyaudio.PyAudio()
        self.stream = self.p.open(format=PY_AUDIO_FORMAT,
channels=CHANNELS,
                                  rate=SAMPLE_RATE, output=True,
                                  frames_per_buffer=frames_per_buffer)

        # FIXED: Use accumulator for proper chunk alignment
        self.input_accumulator = np.array([], dtype=np.int16)
        self.input_timestamps = []
        self.last_played = np.zeros(frames_per_buffer, dtype=np.int16)
        self.packet_count = 0  # For debugging

    def accumulate_input_to_model_size(self):
        """FIXED: Accumulate 256-sample ESP32 chunks into 1600-sample model
chunks"""
        try:
            while len(self.input_accumulator) < MODEL_HOP_SAMPLES:
                input_data, input_timestamp = self.playback_q.get_nowait()
                input_int16 = np.frombuffer(input_data,
dtype='<i2').astype(np.int16)

                self.input_accumulator =
np.concatenate([self.input_accumulator, input_int16])
                self.input_timestamps.append(input_timestamp)

        except queue.Empty:
            pass

        # Return a chunk if we have enough samples
        if len(self.input_accumulator) >= MODEL_HOP_SAMPLES:
            chunk = self.input_accumulator[:MODEL_HOP_SAMPLES].copy()
            # Use the timestamp from the first sample in the chunk
            chunk_timestamp = self.input_timestamps[0] if
self.input_timestamps else time.time()

            # Remove processed samples
            self.input_accumulator =
self.input_accumulator[MODEL_HOP_SAMPLES:]
            # Keep only relevant timestamps (approximate)
            samples_per_timestamp = ESP32_BUFFER_SIZE
            timestamps_to_remove = MODEL_HOP_SAMPLES //
samples_per_timestamp
            self.input_timestamps =
self.input_timestamps[timestamps_to_remove:]

            return chunk, chunk_timestamp

        return None, None

    def run(self):
        while self.running:
            # FIXED: Accumulate input to match model output chunk size
            input_chunk, input_timestamp =
self.accumulate_input_to_model_size()

            # Get model output
            try:
                output_chunk, output_timestamp =
self.output_q.get(timeout=0.1)
            except queue.Empty:
```

```
                continue

            if input_chunk is None:
                continue  # Not enough input accumulated yet

            self.packet_count += 1

            # FIXED: Correct error signal calculation for model that
outputs -x(n)
            # Since model outputs -x(n), we want: e(n) = x(n) + y(n) = x(n)
+ (-x(n)) = 0
            error_signal = input_chunk.astype(np.float32) +
output_chunk.astype(np.float32)
            error_int16 = np.clip(error_signal, -32768,
32767).astype(np.int16)


            # Save error signal
            self.error_save_list.append(error_int16.copy())

            # FIXED: Minimal processing to reduce artifacts
            # Reduce gain to prevent clipping/distortion
            error = error_int16.astype(np.float32)

            to_play = np.clip(0.4*error, -32768, 32767).astype(np.int16)

            self.last_played = to_play.copy()

            try:
                self.stream.write(to_play.tobytes())
            except Exception as e:
                print("Error playback error:", e)

    def stop(self):
        self.running = False
        time.sleep(0.05)
        try:
            if self.stream.is_active():
                self.stream.stop_stream()
            self.stream.close()
        except:
            pass
        try:
            self.p.terminate()
        except:
            pass

# ----------------------------
# Keyboard input handler - UNCHANGED
# ----------------------------
def keyboard_handler():
    global current_mode
    print("\nKeyboard commands:")
    print("  'r' - ANC OFF (silence)")
    print("  'a' - ANC ON")
    print("  'q' - Quit")

    while True:
        try:
            if sys.platform == "win32":
                import msvcrt
```

```
                if msvcrt.kbhit():
                    key = msvcrt.getch().decode('utf-8').lower()
                else:
                    time.sleep(0.1)
                    continue
            else:
                # Unix-like systems
                key = input().strip().lower()

            if key == 'r':
                with mode_lock:
                    current_mode = MODE_OFF
                print("ANC OFF - Speaker silenced")
            elif key == 'a':
                with mode_lock:
                    current_mode = MODE_ON
                print(f"ANC ON, Latency: {latency_samples} samples
({latency_samples/SAMPLE_RATE*1000:.1f}ms)")
            elif key == 'q':
                break

        except (KeyboardInterrupt, EOFError):
            break
        except:
            time.sleep(0.1)

# ----------------------------
# Main
# ----------------------------
def main():
    global current_mode, latency_samples

    # Sanity checks
    if not os.path.exists(MODEL_JIT_PATH):
        print("ERROR: MODEL_JIT_PATH not found:", MODEL_JIT_PATH)
        return

    device = torch.device('cpu')
    print("Loading TorchScript model (CPU)...")
    jit_model = torch.jit.load(MODEL_JIT_PATH, map_location='cpu')
    jit_model.eval()

    # Serial port
    try:
        ser = serial.Serial(COM_PORT, BAUD_RATE, timeout=1)
    except Exception as e:
        print(f"Error opening serial port {COM_PORT}: {e}")
        return

    # Queues
    playback_q = queue.Queue(maxsize=PLAYBACK_Q_MAX)
    model_q = queue.Queue(maxsize=MODEL_Q_MAX)
    output_q = queue.Queue(maxsize=OUTPUT_Q_MAX)

    # Lists for saving WAVs
    input_save_list = []
    output_save_list = []
    error_save_list = []

    # Threads
    reader = SerialReader(ser, playback_q, model_q)
```

```
    model_worker = ModelWorker(jit_model, model_q, output_q,
input_save_list, output_save_list, device='cpu')
    error_player = None

    # Start threads
    reader.start()
    model_worker.start()

    print(f"System started - Mode: {current_mode}")
    print("Keep volume low on first run!")

    # Start keyboard handler in separate thread
    kb_thread = threading.Thread(target=keyboard_handler, daemon=True)
    kb_thread.start()

    def cleanup_and_save(signum=None, frame=None):
        nonlocal ser, reader, model_worker, error_player
        print("\nStopping threads...")
        try:
            reader.stop()
        except:
            pass
        try:
            model_worker.stop()
        except:
            pass
        try:
            if error_player:
                error_player.stop()
        except:
            pass

        # give threads a moment to stop
        time.sleep(0.3)

        try:
            ser.close()
        except:
            pass

        print("Saving WAV files...")

        try:
            if len(input_save_list) > 0:
                in_all = np.concatenate(input_save_list).astype(np.int16)
                save_wav_int16(OUTPUT_INPUT_WAV, in_all, SAMPLE_RATE)
                print(f"Saved raw input -> {OUTPUT_INPUT_WAV}")
            else:
                print("No input audio recorded.")
        except Exception as e:
            print("Failed saving input WAV:", e)

        try:
            if len(output_save_list) > 0:
                out_all = np.concatenate(output_save_list).astype(np.int16)
                save_wav_int16(OUTPUT_MODEL_WAV, out_all, SAMPLE_RATE)
                print(f"Saved model output -> {OUTPUT_MODEL_WAV}")
            else:
                print("No model output recorded.")
        except Exception as e:
            print("Failed saving model output WAV:", e)
```

```
        try:
            if len(error_save_list) > 0:
                err_all = np.concatenate(error_save_list).astype(np.int16)
                save_wav_int16(OUTPUT_ERROR_WAV, err_all, SAMPLE_RATE)
                print(f"Saved error signal -> {OUTPUT_ERROR_WAV}")
            else:
                print("No error signal recorded.")
        except Exception as e:
            print("Failed saving error WAV:", e)

        print(f"Final latency measurement: {latency_samples} samples
({latency_samples/SAMPLE_RATE*1000:.1f}ms)")
        print("Done.")
        # force exit
        os._exit(0)

    signal.signal(signal.SIGINT, cleanup_and_save)

    # Mode switching logic
    active_player = None
    try:
        while True:
            # If user closed keyboard thread (pressed 'q'), then exit
            if not kb_thread.is_alive():
                print("Keyboard thread ended — exiting.")
                cleanup_and_save()
                break

            with mode_lock:
                mode = current_mode

            if mode == MODE_OFF:
                if active_player == 'error':
                    try:
                        if error_player:
                            error_player.stop()
                    except:
                        pass
                    active_player = None

            elif mode == MODE_ON:
                if active_player != 'error':
                    if error_player is None or not error_player.is_alive():
                        error_player = ErrorPlaybackThread(playback_q,
output_q, error_save_list, frames_per_buffer=MODEL_HOP_SAMPLES)
                        error_player.start()
                    active_player = 'error'

            time.sleep(0.2)
    except KeyboardInterrupt:
        cleanup_and_save()
    except Exception as e:
        print("Fatal error in main loop:", e)
        cleanup_and_save()

if __name__ == "__main__":
    main()
```

A.6 Serial Data Recording and Visualization Code

```
import serial
import wave
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm
import matplotlib.cm as cm

# === Configuration ===
SERIAL_PORT = 'COM9'
BAUD_RATE = 500000
SAMPLE_RATE = 16000
SYNC_WORD = b'\xAA\x55'
BYTES_PER_SAMPLE = 2
OUTPUT_WAV = "output.wav"

# === Initialize Serial ===
ser = serial.Serial(SERIAL_PORT, BAUD_RATE, timeout=1)
ser.reset_input_buffer()
print(f"🎙️ Recording from ESP32... Press Ctrl+C to stop")

all_samples = bytearray()

try:
    while True:
        # --- Wait for sync header (0xAA55) ---
        while True:
            sync = ser.read(2)
            if sync == SYNC_WORD:
                break

        # --- Read payload size (2 bytes, little-endian) ---
        payload_size_bytes = ser.read(2)
        if len(payload_size_bytes) != 2:
            continue
        payload_size = int.from_bytes(payload_size_bytes, 'little')

        # --- Read audio payload ---
        chunk = ser.read(payload_size)
        if len(chunk) == payload_size:
            all_samples.extend(chunk)
        else:
            print(f"⚠️ Incomplete chunk ({len(chunk)} bytes, expected
{payload_size}), skipping")

except KeyboardInterrupt:
    ser.close()
    print("🛑 Stopped by user")

    # === Save audio to WAV ===
    if len(all_samples) % 2 != 0:
        all_samples = all_samples[:-1]

    with wave.open(OUTPUT_WAV, 'wb') as wf:
        wf.setnchannels(1)
        wf.setsampwidth(2)
        wf.setframerate(SAMPLE_RATE)
        wf.writeframes(all_samples)
```

```
    total_samples = len(all_samples) // 2
    duration = total_samples / SAMPLE_RATE
    print(f"💾 Saved {total_samples} samples ({duration:.2f} sec) to
{OUTPUT_WAV}")

    # === Plotting Section ===
    print("📈 Generating waveform and FFT plots...")

    samples = np.frombuffer(all_samples, dtype=np.int16)

    # --- Prepare FFT (up to 2 seconds of data) ---
    fft_len = min(2 * SAMPLE_RATE, len(samples))
    windowed = samples[:fft_len] * np.hanning(fft_len)
    fft_data = np.fft.rfft(windowed)
    fft_freqs = np.fft.rfftfreq(fft_len, 1 / SAMPLE_RATE)
    amplitude = np.abs(fft_data)

    # === Plot Waveform + FFT Spectrum ===
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 6), height_ratios=[1,
1])

    # --- Waveform ---
    ax1.plot(samples, linewidth=0.5)
    ax1.set_title("Waveform")
    ax1.set_xlabel("Sample Index")
    ax1.set_ylabel("Amplitude")
    ax1.grid(True)

    # --- FFT (Log-scaled color gradient) ---
    log_norm = LogNorm(vmin=fft_freqs[1], vmax=fft_freqs[-1])
    cmap = plt.get_cmap("plasma")
    colors = cmap(log_norm(fft_freqs))

    for i in range(len(fft_freqs) - 1):
        ax2.plot(fft_freqs[i:i+2], amplitude[i:i+2], color=colors[i],
linewidth=1.0)

    sm = cm.ScalarMappable(cmap=cmap, norm=log_norm)
    sm.set_array([])
    cbar = plt.colorbar(sm, ax=ax2, orientation='horizontal', pad=0.1)
    cbar.set_label('Frequency (Hz)')

    ax2.set_title("FFT Spectrum (Log Scale)")
    ax2.set_xlabel("Frequency (Hz)")
    ax2.set_ylabel("Amplitude")
    ax2.set_xscale("log")
    ax2.set_xlim(20, SAMPLE_RATE / 2)
    ax2.grid(True, which='both', linestyle='--')

    plt.tight_layout()
    plt.show()
```

A.7 Energy Spectrum Comparison Code

```
import librosa
import librosa.display
import soundfile as sf
```

```
import numpy as np
import matplotlib.pyplot as plt

SAMPLE_RATE = 16000

def load_and_resample(path, sr=SAMPLE_RATE):
    d, file_sr = sf.read(path)
    if d.ndim > 1:
        d = np.mean(d, axis=1)  # convert to mono
    if file_sr != sr:
        d = librosa.resample(d.astype(np.float32), orig_sr=file_sr,
target_sr=sr)
    return d, sr

def plot_energy_spectrum(file1, file2):
    # Load both files
    d1, sr1 = load_and_resample(file1)
    d2, sr2 = load_and_resample(file2)

    # Compute FFT magnitude
    freqs1 = np.fft.rfftfreq(len(d1), 1/sr1)
    freqs2 = np.fft.rfftfreq(len(d2), 1/sr2)
    mag1 = np.abs(np.fft.rfft(d1))**2  # Energy spectrum
    mag2 = np.abs(np.fft.rfft(d2))**2

        # Plot
    plt.figure(figsize=(12,6))
    plt.semilogx(freqs1, mag1, label=f"{file1}")  # log frequency axis
    plt.semilogx(freqs2, mag2, label=f"{file2}", alpha=0.7)
    plt.title("Energy Spectrum of Two WAV Files")
    plt.xlabel("Frequency (Hz, log scale)")
    plt.ylabel("Energy (linear scale)")
    plt.legend()
    plt.grid(True, which="both", ls="--")
    plt.show()



# Example usage
file1 = r"D:\TestFIles\CRN\d_primary.wav"
file2 = r"D:\TestFIles\CRN\error_signal.wav"
plot_energy_spectrum(file1, file2)
```

A.8 ESP32 Arduino Code

```
#include <driver/i2s.h>

#define I2S_WS   25
#define I2S_SD   33
#define I2S_SCK  32
#define I2S_PORT I2S_NUM_0
#define SAMPLE_RATE 16000
#define BUFFER_SIZE 256   // changed from 64

int32_t i2s_buffer[BUFFER_SIZE];

void setupI2S() {
  i2s_config_t i2s_config = {
```

```
    .mode = (i2s_mode_t)(I2S_MODE_MASTER | I2S_MODE_RX),
    .sample_rate = SAMPLE_RATE,
    .bits_per_sample = I2S_BITS_PER_SAMPLE_32BIT,
    .channel_format = I2S_CHANNEL_FMT_ONLY_LEFT,
    .communication_format = I2S_COMM_FORMAT_I2S,
    .intr_alloc_flags = 0,
    .dma_buf_count = 8,
    .dma_buf_len = BUFFER_SIZE,
    .use_apll = false
  };

  i2s_pin_config_t pin_config = {
    .bck_io_num = I2S_SCK,
    .ws_io_num = I2S_WS,
    .data_out_num = -1,
    .data_in_num = I2S_SD
  };

  i2s_driver_install(I2S_PORT, &i2s_config, 0, NULL);
  i2s_set_pin(I2S_PORT, &pin_config);
  i2s_zero_dma_buffer(I2S_PORT);
}

void setup() {
  Serial.begin(500000);
  setupI2S();
}

void loop() {
  size_t bytesRead;
  i2s_read(I2S_PORT, i2s_buffer, sizeof(i2s_buffer), &bytesRead,
portMAX_DELAY);

  // --- Frame header ---
  Serial.write(0xAA);
  Serial.write(0x55);
  uint16_t payload_size = BUFFER_SIZE * 2; // 16-bit samples
  Serial.write((uint8_t*)&payload_size, 2);

  // --- Send audio samples ---
  for (int i = 0; i < BUFFER_SIZE; i++) {
    int16_t sample = i2s_buffer[i] >> 13;  // fixed scaling
    Serial.write((uint8_t*)&sample, 2);
  }
}
```