SONAR SHIELD

Group Members

Muhammad Arqam Malik (01-131222-028)

Zeeshan Ahmad Abbasi (01-131222-051)

***Supervisor*:** Dr. Joddat Fatima

A Final Year Project submitted to the Department of Software Engineering, Faculty of Engineering Sciences, Bahria University, Islamabad in the partial fulfillment for the award of degree in Bachelor of Software Engineering

May 2026

FYP Completion Certificate

Student Name:	Zeeshan Ahmad Abbasi	**Enrolment No:**	01-131222-051
Student Name:	Muhammad Arqam Malik	**Enrolment No:**	01-131222-028

Programme of Study: Bachelor of Software Engineering

Project Title: SONAR SHIELD

It is to certify that the above students' project has been completed to my satisfaction and to my belief, its standard is appropriate for submission for evaluation. I have also conducted a plagiarism test of this thesis using HEC prescribed software and found a similarity index at_____ that is within the permissible limit set by the HEC. I have also found the thesis in a format recognized by the department.

Supervisor's Signature:_____

Date: _____16th April, 2026_____ Name: Dr. Joddat Fatima

Certificate of Originality

This is to certify that the intellectual contents of the project titled:

SONAR SHIELD – AI-Based System for Underwater Noise Reduction and Satellite Signal Correction

are the product of our own work except, as cited properly and accurately in the acknowledgements and references, the material taken from such sources as research journals, books, internet etc. solely to support, elaborate, compare, extend and/or implement the earlier work. Further, this work has not been submitted by us previously for any degree, nor it shall be submitted by us in the future for obtaining any degree from this University, or any other University or institution. The incorrectness of this information, if proved at any stage, shall authorize the University to cancel our degree.

Name of the Student: Zeeshan Ahmad Abbasi

Signature:_____ Date:_____16th April, 2026_____

Name of the Student: Muhammad Arqam Malik

Signature:_____ Date:_____16th April, 2026_____

SONAR SHIELD

Sustainable Development Goals

SDG No	Description of SDG	SDG No	Description of SDG
SDG 1	No Poverty	SDG 9 ✓	Industry, Innovation, and Infrastructure
SDG 2	Zero Hunger	SDG 10	Reduced Inequalities
SDG 3	Good Health and Well Being	SDG 11	Sustainable Cities and Communities
SDG 4	Quality Education	SDG 12	Responsible Consumption and Production
SDG 5	Gender Equality	SDG 13	Climate Change
SDG 6	Clean Water and Sanitation	SDG 14✓	Life Below Water
SDG 7	Affordable and Clean Energy	SDG 15	Life on Land
SDG 8	Decent Work and Economic Growth	SDG 16	Peace, Justice and Strong Institutions
		SDG 17	Partnerships for the Goals

Range of Complex Solving			
	Attribute	**Complex Problem**	
1	Range of conflicting requirements	Involve wide-ranging or conflicting technical, engineering and other issues.	✓
2	Depth of analysis required	Have no obvious solution and require abstract thinking, originality in analysis to formulate suitable models.	✓
3	Depth of knowledge required	Requires research-based knowledge much of which is at, or informed by, the forefront of the professional discipline and which allows a fundamentals-based, first principles analytical approach.	✓
4	Familiarity of issues	Involve infrequently encountered issues	✓
5	Extent of applicable codes	Are outside problems encompassed by standards and codes of practice for professional engineering	✓
6	Extent of stakeholder involvement and level of conflicting requirements	Involve diverse groups of stakeholders with widely varying needs.	✓
7	Consequences	Have significant consequences in a range of contexts.	✓
8	Interdependence	Are high level problems including many component parts or sub-problems	✓
Range of Complex Problem Activities			
	Attribute	**Complex Activities**	
1	Range of resources	Involve the use of diverse resources (and for this purpose, resources include people, money, equipment, materials, information and technologies).	
2	Level of interaction	Require resolution of significant problems arising from interactions between wide ranging and conflicting technical, engineering or other issues.	
3	Innovation	Involve creative use of engineering principles and research-based knowledge in novel ways.	✓
4	Consequences to society and the environment	Have significant consequences in a range of contexts, characterized by difficulty of prediction and mitigation.	
5	Familiarity	Can extend beyond previous experiences by applying principles-based approaches.	

Abstract

Sonar Shield is an intelligent dual module AI solution developed with the primary objective of solving one of the key problems of contemporary signal processing technologies that relates to the loss of valuable information due to noise in two unique yet equally essential environments: under-water acoustic environment and satellite communication. Two unique modules are incorporated into one universal platform, which can be used by scientists, engineers, and specialists working in the corresponding fields of expertise.

The Aqua Noise Reduction Module addresses underwater communication interference from sources like ocean currents, marine life, and vessel vibrations. It utilizes a CNN-based classifier to identify specific noise types within a signal, then automatically selects the most effective deep learning architecture such as Autoencoders, U-Nets, and WaveNet to perform denoising. The resulting high-quality audio is then objectively validated using SNR, PESQ, and STOI performance metrics.

The second module is the Satellite Signal Noise Reduction Module. The problem addressed in this module is that of error propagation within long-distance satellite communication that can be affected by environmental noise, electromagnetic interferences, and degraded communication channels. To solve this problem, BiGru Decoder sequence models along with FEC, namely Reed-Solomon coding algorithm, will be used to detect and correct errors in telemetry data obtained from satellite signals.

The Sonar Shield project is a modular and scalable signal enhancement framework built using Python, TensorFlow, and PyTorch[3]. It integrates a user-friendly interface with advanced data visualization and automated reporting, providing a cross-platform solution for Windows, Linux, and macOS that demonstrates the practical effectiveness of AI in signal processing.

Keywords: Noise Reduction, Signal Enhancement, Underwater Acoustics, Satellite Communications, Deep Learning, CNN, LSTM, Transformer, Autoencoder, U-Net, WaveNet, SNR, BER, PESQ, Forward Error Correction

Dedication

For our parents who have unreservedly loved, sacrificed, and endlessly encouraged us to be the people that we are. You believed in us through each and every struggle that we encountered.

For our advisor, Dr. Joddat Fatima, whose knowledge, guidance, and wisdom made our dreams into reality. You have influenced our thoughts not only in engineering but in many other aspects of life.

To all engineers and researchers working on the cutting edge of signal processing and artificial intelligence, we hope that in our own small way, we can make a contribution.

Acknowledgments

Praise and Thanks are offered to Allah Almighty who bestowed his blessings, guidance, and limitless grace upon us in accomplishing this task. With out his support and guidance nothing would have come out of this effort.

We are very grateful to our supervisor Dr. Joddat Fatima from the Department of Computer Science of Bahria University Islamabad for her unique supervision during this entire project period. Her technical support, constructive comments, valuable guidance, and cooperation have been very helpful in developing this project.

We also like to thank our teachers of the Department of Software Engineering of Bahria University Islamabad who helped us develop the theoretical background and practical knowledge to conduct this project. Their teaching will always be remembered and appreciated.

We are especially grateful to our class fellows who gave us constructive criticism, discussed with us technical matters, and helped us when we were stuck in tough phases of this project.

In addition, we would like to extend our sincere thanks to our families for their unwavering support and love, which has been our biggest source of motivation.

Table of Contents

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Signaling process has become the key to modern technology infrastructures. Be it exploring under the water or communicating through satellites, signals play an important role in ensuring effective operations. However, both under the water acoustic environment as well as the satellite communication channel suffer from the effect of noise, interference, and degradation – all of which are dealt with using existing methodologies in a separate manner.

Sonar Shield is an innovative software-based technology designed to tackle the abovementioned problem through a unified method that uses artificial intelligence to reduce the effect of noise and enhance the signal in two distinct domains. The concept and development of Sonar Shield were done during my final year project at Bahria University Islamabad, supervised by Dr. Joddat Fatima.

1.1. Motivation

The problem statement behind Sonar Shield stems from the increased need for precise and uncorrupted signal information in critical settings where slight noise could result in life-threatening catastrophes. For marine scientists, erroneous information due to sonar could incorrectly identify marine species or pose navigation challenges for naval forces. For space communication systems, noise results in information loss, interruptions in communication services, and reduced efficiency of GPS, weather forecasts, and global Internet networks.

Previous signal processing software options have mainly been single-discipline based and were restricted by limited flexibility and intelligent capabilities. However, the possibility of integrating advanced deep learning models like CNNs, LSTMs, and Transformers together into a single platform represented an appealing and worthwhile challenge for the engineering field.

Pakistan's emerging ambitions to be recognized within the aerospace industry provide a unique window of opportunity for this project. Sonar Shield aims to cater to engineers, researchers, and institutions that seek low-cost, powerful signal processing options free of any proprietary hardware and software constraints.

1.2. Objectives

The key goals for this project include:

1. Demonstrate the ability to build an AI-driven Aqua Noise Reduction Module that is able to classify underwater noise and apply adaptive denoising models to provide high quality sound outputs.
2. Create an AI-driven Satellite Signal Error Correction Module using sequence modeling such as LSTM and Transformer along with Forward Error Correction algorithms to recover corrupted satellite signals.
3. Develop an interface where users can interact with both modules from one application.
4. Demonstrate quantifiable improvements using established measures of SNR, BER, PESQ, and STOI.
5. Create a scalable architecture that would enable future expansion with regards to more types of noise, AI models, and scenarios.
6. Add security measures such as user authentication and encryption.

1.3. Main Contributions

The project contributes in the following ways to signal processing and software engineering domains:

- A dual-domain signal processing platform that innovatively combines the application of underwater acoustic denoising and satellite signal error correction within one software architecture—a technique that has never been attempted in any open-source or scholarly literature.
- A denoising pipeline capable of classifying the type of noise through CNN-based classification algorithms and then automatically selecting between multiple types of deep learning denoising algorithms (Autoencoder, U-Net, WaveNet). This is significantly more effective than a static filtering algorithm.
- Satellite error correction using AI with LSTM/Transformer sequence models and Reed-Solomon Forward Error Correction to achieve a minimum target Bit Error Rate reduction of 70% or higher.

- A complete set of software specification documents, which include SRS, SDS, UML diagrams, database schema design, and user interface designs. These documents can be reused as templates for future projects involving signal processing.
- Implementation with 80% or better test coverage in unit testing, integration testing, and system testing stages.

1.4. Report Organisation

Structure of this report:

Chapter 2: Background Study & Literature Review: Analyzes existing systems for signal processing, noise filtering, uses of AI in similar applications, and highlights gaps in research which led to the creation of Sonar Shield.

Chapter 3: System Requirements: Discusses functional and non-functional requirements of Sonar Shield, use case diagrams, descriptions, interface requirements, and feasibility study for the project.

Chapter 4: System Design: Explains the software architecture along with design models in UML (Class, Activity, Sequence, Deployment, Component and Package diagrams), data model, and user interface design.

Chapter 5: System Implementation: Explains tools, technologies, frameworks, and method used to develop Sonar Shield, along with explanation of algorithms and processing pipeline.

Chapter 6: System Testing & Evaluation: Presents test cases and evaluation results highlighting system performance relative to requirements stated.

Chapter 7: Conclusion: Highlights contributions made by this project, discusses strengths and weaknesses, and presents areas for further work.

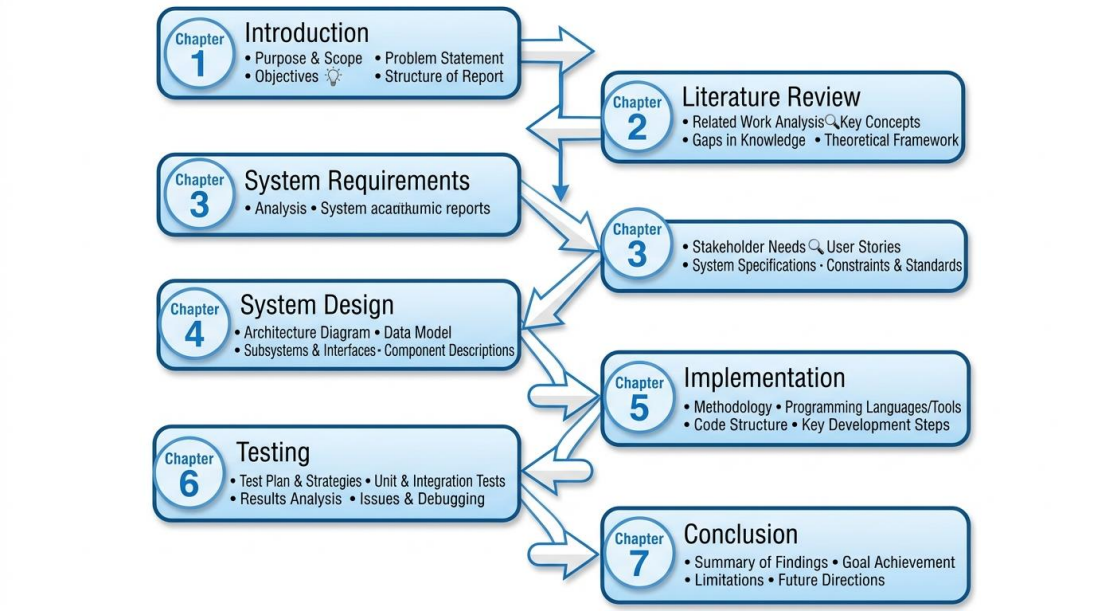


Figure 1 Report Organisation Diagram

Chapter 2

Background Study

This chapter outlines the existing literature regarding the two main problem areas that Sonar Shield aims to address. Specifically, it surveys the state-of-the-art in signal processing algorithms, the shortcomings of current methods, and places Sonar Shield in context with other existing work. This survey is informed by both the academic literature and existing frameworks for signal processing, as well as other software packages.

2.1. Key Concepts in Signal Processing

2.1.1. Noise in Signal Environments

Noise refers to any extraneous signal introduced into a desired signal. Sources of noise in underwater systems may be classified as follows: (1) ambient noise due to waves, wind, and precipitation; (2) biological noise due to animals in the aquatic environment; (3) man-made noise due to ships, drilling equipment, and sonars; and (4) self-noise introduced by the receiving sensor. As such, the range of frequencies involved in noise pollution and the varying dynamic levels of intensity necessitate more sophisticated techniques than static filters [Hildebrand, 2009].

Signal deterioration in satellite communications is brought about by: (1) atmospheric scintillation; (2) Doppler shift; (3) multipath fading; (4) interference from terrestrial electromagnetic signals; and (5) thermal noise. This introduces bit errors, whereby single-bit information changes in value as they traverse the communication channel [Proakis & Salehi, 2008].

2.1.2. Classical Signal Processing Techniques

Both classical algorithms for underwater and satellite noise removal are based on linear filters. The most popular techniques are: spectral subtraction based on Fourier transform, where noise is subtracted from the signal by estimating noise spectrum; Wiener filtering, where a linear filter is chosen optimally in statistical sense, provided that the power spectra of signal and noise are known; bandpass filtering, where only certain frequency bands are passed. Although these algorithms are fast enough, their performance is poor in the presence of non-stationary noise, i.e., when noise properties vary in time.

2.1.3. Adaptive Filtering

In addition, adaptive filters can resolve certain disadvantages associated with fixed linear filters by modifying their filter coefficients depending on the statistical characteristics of the signal being processed. The LMS algorithm and the RLS algorithm are two examples of adaptive filtering techniques that have gained wide acceptance. Adaptive noise cancellation has been successfully implemented in sonar applications for eliminating engine noise and in satellite ground stations for minimizing atmospheric turbulence interference. Nevertheless, adaptive filters are constrained by the same assumptions of linearity and stationarity in small time intervals [Widrow & Stearns, 1985].

2.2. Deep Learning Approaches to Noise Reduction

2.2.1. Autoencoders for Audio Denoising

Autoencoders are neural network architectures that learn compressed representation of input data through an encoder-decoder structure. Applied to audio denoising, an autoencoder is trained to map noisy spectrograms to clean spectrograms, learning to reconstruct the clean signal from a corrupted input. Pascual et al. (2017) demonstrated that convolutional autoencoders outperform conventional Wiener filtering for speech enhancement tasks, achieving higher PESQ and STOI scores. Their work forms a foundational reference for the Aqua module's denoising pipeline.

2.2.2. U-Net Architecture

Originally designed for biomedical images, the U-Net algorithm was successfully used to perform audio source separation and enhancement tasks. The core idea of this algorithm lies in the presence of skip connections between encoder and decoder stages with the same resolution to allow the algorithm to keep fine details while reconstructing audio signals. Jansson et al. (2017) used U-Net to separate musical sources, and further research showed that it can be applied for environmental sound enhancement as well.

2.2.3. WaveNet-Based Approaches

WaveNet was proposed by van den Oord et al. (2016) at DeepMind and is a deep generative model for modeling raw audio waveforms through dilated causal convolutions. This model can handle long-term dependencies well and can be effectively used to characterize complicated acoustic patterns in

the underwater environment. Rethage et al. (2018) showed how WaveNet could be used for speech denoising and produced perceptually superior results compared to other methods.

2.2.4. LSTM and Transformer Models for Sequence Error Correction

Long Short Term Memory Networks, first proposed by Hochreiter and Schmidhuber (1997), are a type of recurrent neural network that can learn dependencies over a long period of time. They have been used successfully in channel estimation and correction for wireless communication channels. More recently, the Transformer architecture, first introduced by Vaswani et al. (2017) in the field of natural language processing, has been implemented in the field of sequence to sequence correction of satellite signal streams due to its ability to learn complex temporal correlations in the stream.

2.3. Related Systems and Tools

2.3.1. Audacity and SoX

Audacity is an open-source software application for sound recording and editing. The application supports noise reduction through spectral subtraction with manual configuration of noise characteristics. Sound eXchange (SoX) is a command line application for audio manipulation. The application supports filtering, normalization, and conversion of audio formats. The applications lack adaptive capability through AI. On the other hand, Sonar Shield offers advanced capabilities in noise classification and denoising by incorporating AI into the process.

2.3.2. MATLAB Signal Processing Toolbox

The MATLAB Signal Processing and Deep Learning Toolboxes offer significant capabilities for both traditional and artificial intelligence approaches to signal processing. Unfortunately, MATLAB is only usable with expensive commercial licensing, which reduces its availability in budgetary-restricted research settings. The Sonar Shield platform is fully based on open source Python toolkits, rendering it free to use and equally effective at specialized tasks.

2.3.3. GNU Radio and SDR Tools

GNU Radio is an open source software-defined radio platform utilized in the process of satellite signal reception and processing. While GNU Radio offers strong capabilities in the design of signal chains, it demands a high level of knowledge in RF engineering and signal flow graph coding. Sonar Shield supports GNU Radio in its back-end processing using AI technology for correcting errors in received satellite signals.

2.4. Research Gaps and Motivation for Sonar Shield

Key research gaps identified from the review of existing literature and tools include:

- Multidomain processing limitation: Existing solutions consider either underwater acoustic processing or satellite communication processing but cannot perform both. Sonar Shield solves this gap.
- Failure to adapt dynamically: Traditional and existing AI solutions rely on static pipeline designs. Sonar Shield is able to dynamically select a machine learning model based on noise types.
- Barrier to entry: High-performance tools are either proprietary (MATLAB) or very technical (GNU Radio). Sonar Shield aims to make the process accessible through a Python GUI interface.
- Evaluation capability inadequacy: Integration of evaluation metrics (SNR, BER, PESQ, and STOI) is rare in current toolboxes. Sonar Shield includes this feature in its design.
- Scalability issues: Current academic toolboxes are usually for proof-of-concept only. Sonar Shield incorporates role-based access control, database management, and modularity to cater to large institutions.

Table 1 Comparison of Related Signal Processing Systems

Feature	Audacity	MATLAB	GNU Radio	Sonar Shield
Underwater Audio Denoising	Basic	Advanced	None	Advanced AI
Satellite Signal Correction	None	Advanced	Advanced	Advanced AI
AI/ML Integration	None	Partial	None	Full
Noise-Type Classification	None	Manual	None	Automated
Open Source	Yes	No	Yes	Yes
GUI Interface	Yes	Yes	Yes	Yes
Automated Reporting	None	Partial	None	Full
Multi-User Support	None	None	None	Yes (RBAC)

2.5. Conclusion

The literature review presented above shows that although many techniques related to underwater acoustic noise suppression and satellite signal error correction have been well investigated individually, no such existing solution exists today that incorporates all these techniques effectively through an artificial intelligence algorithm and is available freely for use by a general audience. The inclusion of CNN for noise detection, adaptation-based noise removal, and LSTM/Transformer for error correction through a handy application becomes an important milestone for this research project. This will be described further in the following chapters.

Chapter 3

System Requirements

In this chapter, all of the system requirements for Sonar Shield have been provided based on the analysis of the problem domain, stakeholder interviews, and study of other existing systems. The system requirements have been described according to IEEE Std 830-1998 and include: use cases, interface requirements, database requirements, non-functional requirements, and project feasibility.

3.1. Use Case Diagram

Below is an exemplary top-level use case diagram depicting the communication between the three main actors involved, namely User, External Hardware/Sensor, and System Administrator, with regard to the functionality of Sonar Shield.

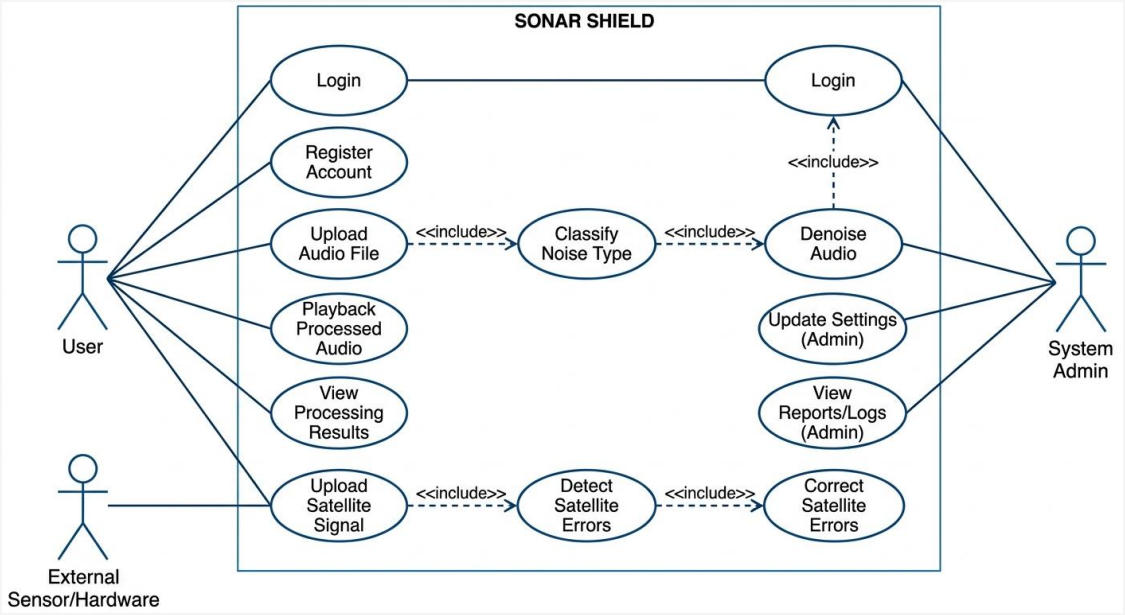


Figure 2 Top-Level Use Case Diagram of Sonar Shield

3.2. Functional Requirements

We describe our functional requirements using structured use cases, all of which are related to the use case diagram. For each use case, we have specified the actors involved, the preconditions required for initiating the use case, the main flow for accomplishing it, and alternate flows that cover any deviations or exceptions.

Table 2 Use Case UM-01: User Registration and Authentication

Field	Details
Use Case ID	UM-01
Use Case Name	User Registration and Authentication
Actor(s)	User, System Administrator
Priority	High
Pre-Conditions	User has valid credentials or ennrollment data; system is running
Post-Conditions	User is authenticated and directed to dashboard; session is initiated.

System Flow:

1. User accesses Sonar Shield application and clicks 'Register' or 'Login'.
2. Application shows corresponding registration/login screen.
3. User enters his/her username, password, and email address (optional).
4. Application validates input fields by checking for any duplicate entries and applying password strength policy.
5. In case of registration, the application generates an account entry in the database with encrypted password.
6. In case of login, the application verifies user credentials and starts session.
7. Application directs user to main dashboard page.

Main Flow:

1. User launches Sonar Shield and clicks either 'Register' or 'Login'.
2. Registration or Login page is shown based on user choice.
3. User inputs their Username, Password, and Email (optional).

4. System verifies entered data: verifies user existence, follows password strength criteria.
5. In case of Registration: new user account gets created with hashed password and assigned role.
6. In case of Login: system verifies credentials and starts a session.
7. Authenticated user is redirected to the Dashboard page.

Alternative Flows:

- 2a – Incorrect login information: Error message is displayed, and login form is presented again.
- 2b – Three unsuccessful attempts to log in: User account is locked for 60 minutes.
- 3a – Username already exists while registering: System asks for another username.

Table 3 Use Case AD-01: Noise-Type-Aware Audio Denoising

Field	Details
Use Case ID	AD-01
Use Case Name	Noise-Type-Aware Audio Denoising
Actor(s)	Registered User
Priority	High
Pre-Conditions	User is authenticated; audio file is available in WAV or MP3 format.
Post-Conditions	Denoised audio file is generated; evaluation metrics are computed and displayed.

Main Workflow:

1. User navigates to the Audio Denoising module.
2. User uploads an audio file in WAV or MP3 formats, with size up to 100 MB.
3. System checks if the file format is correct and analyzes audio characteristics (spectrogram, MFCC).
4. System feeds the extracted audio features to the CNN-based noise classifier to determine the type of noise (marine traffic, machine, etc.).

5. System sends input signal to one of the noise reduction models (Autoencoder, U-Net, or WaveNet).
6. Model applies the processing algorithm on audio and creates clean audio output.
7. System calculates SNR, PESQ, and STOI quality assessment metrics.
8. User can listen to both original and denoised audios, compare waveforms/spectrograms, and download the result.

Alternate Workflow:

- 3a. File format is not supported: System shows message with the list of supported file formats.
- 5a. An error occurs during processing: System logs error and asks the user to upload another file.

Table 4 Use Case SC-01: AI-Based Satellite Signal Error Correction

Field	Details
Use Case ID	SC-01
Use Case Name	AI-Based Satellite Signal Error Correction
Actor(s)	Registered User
Priority	High
Pre-Conditions	User is authenticated; satellite signal dataset is available in CSV or BIN format.
Post-Conditions	Corrected dataset is generated; BER, throughput, and latency metrics are displayed.

Primary Flow:

1. User visits Satellite Error Correction Module.
2. User uploads satellite signal data in CSV/BIN file format.
3. System checks validity of uploaded data file format and structure.
4. Data pre-processing occurs through normalization, encoding, and sequencing.

5. Prediction of errors and error type in signal sequence performed using AI model (LSTM/Transformer).
6. Error correction is done using Error Corrector and correctness of the output is ensured by FEC algorithms.
7. Metrics of BER improvement, throughput, and latency computed.
8. Corrected data downloaded by user along with metrics for comparison.

Alternative Flows:

- 3a. File format unsupported: Error message displayed by system, specifying file formats that are supported.
- 4a. Data corrupt or incomplete structurally: Error message logged by system, requesting user upload proper data.

Table 5 Use Case RP-01: Report Generation

Field	Details
Use Case ID	RP-01
Use Case Name	Report Generation and Download
Actor(s)	Registered User, Administrator
Priority	Medium
Pre-Conditions	At least one processing task has been completed.
Post-Conditions	Report file is generated and available for download.

Main Flow:

1. User accesses the reports tab.
2. User specifies start and end dates, signal type, and report file format (PDF or CSV).
3. System consolidates processing statistics, metrics, and logs.
4. System creates a report document that can be downloaded.

3.3. Interface Requirements

3.3.1. User Interface Requirements

- The system will include a graphical user interface with a consistent menu structure (Home, Denoising, Error Correction, Reports, Settings, Help).
- The system will include a dashboard that displays activity, active signals, noise detection, and error count on satellites.
- File uploads will use a drag-and-drop option and file selection options with progress notifications.
- All results will include real-time waveform and spectrogram visualization.
- Hovering help will be included in all interactive objects.
- The interface will allow access through high contrast mode and window resizing options.
- Shortcuts keys: Ctrl+O (Open File), Ctrl+S (Save Result), Ctrl+R (Create Report).

3.3.2. Hardware Interface Requirements

- Inputs: Hydrophones via USB, microphones (with sampling rate between 16 kHz and 96 kHz), satellite data receivers (RS-232/RS-485, USB, and Ethernet).
- Outputs: Audio through 3.5 mm/USB/Bluetooth for noise-free listening, hard disk drive/flash memory for storing the generated files locally.
- System Requirement: Minimum requirement is an Intel Core i5 or AMD Ryzen 5 processor

3.3.3. Software Interface Requirements

- Operating Systems: Windows 10/11 (64-bit), Ubuntu 20.04+, macOS 12+.
- Python 3.9+ with TensorFlow/PyTorch[3], NumPy[4], SciPy, Librosa[5], pandas, scikit-learn, Matplotlib/Seaborn.
- File formats: WAV, MP3, FLAC (audio); CSV, BIN (satellite data); PDF, CSV (reports).

3.4. Non-Functional Requirements

Table 6 Non-Functional Requirements Summary

ID	Category	Requirement	Target
NFR-01	Performance	Audio denoising for 5-min file	≤ 60 seconds (GPU)

ID	Category	Requirement	Target
NFR-02	Performance	Noise classification accuracy	≥ 90%
NFR-03	Performance	SNR improvement	≥ 10 dB over input
NFR-04	Performance	Satellite dataset (1M points)	≤ 2 minutes (GPU)
NFR-05	Performance	BER reduction	≥ 70%
NFR-06	Performance	System startup time	≤ 10 seconds
NFR-07	Reliability	System uptime	≥ 99%
NFR-08	Security	Password storage	bcrypt hashed
NFR-09	Security	Data in transit	TLS/SSL encryption
NFR-10	Usability	New user task completion	≤ 5 minutes
NFR-11	Maintainability	Test coverage	≥ 80%
NFR-12	Portability	OS support	Windows, Linux, macOS

3.5. Project Feasibility

3.5.1. Technical Feasibility

It is possible to develop Sonar Shield due to the existence of multiple free-to-use machine learning frameworks and libraries (such as TensorFlow and PyTorch[3] for deep learning models, and Librosa[5] and SciPy for audio manipulation). The required hardware resources are accessible, as well as existing proof that it is possible to classify noise and correct errors using CNNs and LSTM/Transformer models.

3.5.2. Operational Feasibility

This program incorporates a user-friendly GUI which does not require any command line knowledge on the part of the users. A role based access controls policy enables each category of user, whether researcher, engineer, student or administrator to work within their own domain.

3.5.3. Legal and Ethical Feasibility

All software dependencies have either an open source license or licenses that can be used for academic and research purposes. There is no dependency on any hardware that is not open source. Data protection standards are met by this project. In other words, users' data is protected, secured, and kept for just as long as is necessary.

3.6. Analysis Models

Activity diagram above depicts the process flow of the Aqua Noise Reduction Module starting from the input of the signal to output.

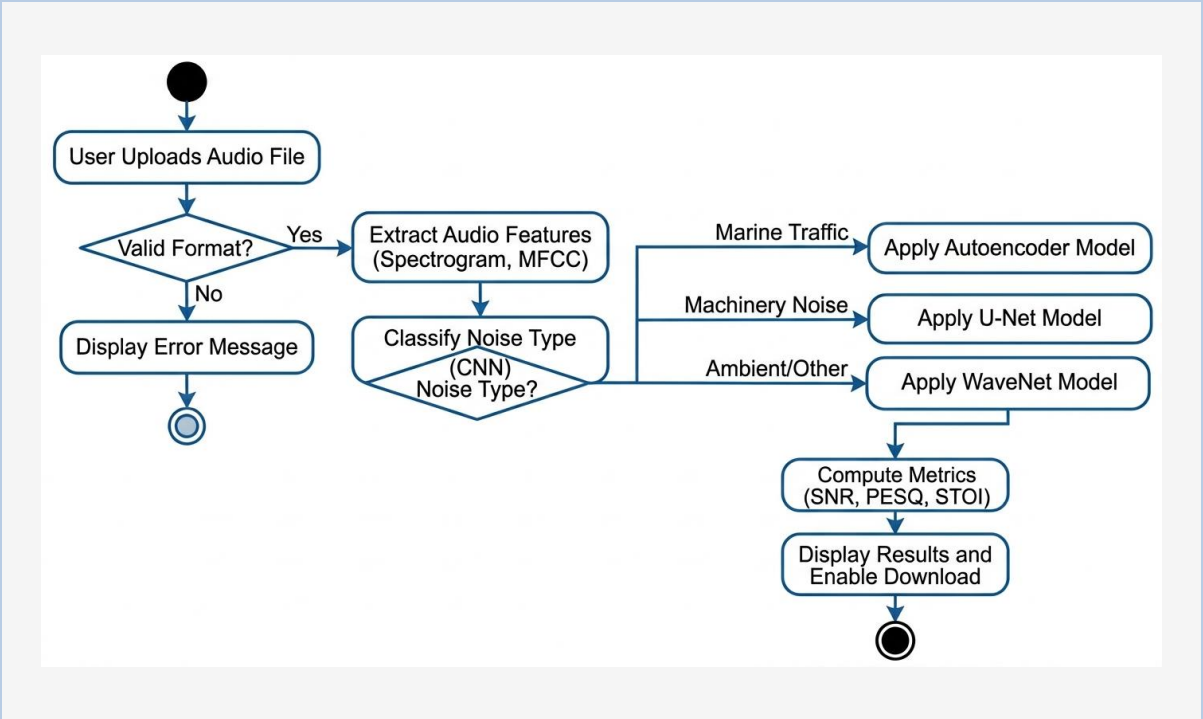


Figure 3 Activity Diagram – Aqua Noise Reduction Module

3.7. Conclusion

This chapter has detailed all system requirements needed for the Sonar Shield system, including functional requirements described in terms of use cases, interface requirements, database requirements, and non-functional requirements with quantifiable objectives. From a feasibility analysis conducted on this project, it has been found to be feasible from technical, operational, and legal perspectives. The requirements defined in this chapter form the final requirement specification document of the project.

Chapter 4

System Design

The design of Sonar Shield System is presented in this chapter based on the requirements specified in Chapter 3. The design process has been carried out based on the object-oriented approach and the UML notation standard has been followed to illustrate the designs. In this chapter, design process, architecture of the system, logical design (class diagrams), dynamic view (activity and sequence diagrams), deployment view, development view (package and component diagrams), data modeling, and UI design have been explained.

4.1. Design Approach

The Sonar Shield System follows a hierarchical and modular approach, following the below guidelines:

- **Layering:** Each layer (Presentation Layer, Processing Layer, Data Layer) has well-defined functions, resulting in minimal coupling between layers and making the system easier to test and maintain.
- **Modularity:** Each core module (Audio Denoising, Satellite Error Correction, User Management, Reporting) operates independently with clearly defined interfaces, making it easier to extend the system in the future without altering its current functioning.
- **Adaptability:** The system design allows adding new types of noises and error correction methods in the processing layer without any changes in architecture; just a new configuration of deep learning models will do.
- **AI-First:** In this architecture, deep learning models are first-class citizens of the architecture, where specific preprocessing, inference, and post-processing modules can be swapped out easily.

4.2. Design Constraints

Table 7 Design Constraints Summary

Constraint	Description	Impact
Python 3.9+	Primary development language is Python	All components must use Python-compatible libraries
Open-Source Only	No proprietary software dependencies	Limits GUI framework to Tkinter, PyQt, or web-based alternatives
PEP 8 Standards	Code must follow Python style guidelines	Requires consistent code review and linting
SQLite / PostgreSQL	Database restricted to open-source RDBMS	Schema must be compatible with both databases
CUDA Optional	GPU acceleration is optional	System must gracefully degrade to CPU-only mode
Max File Size 100MB	Upload limit per file	Large datasets must be segmented before processing

4.3. System Architecture

The architecture of Sonar Shield follows the design of three layers:

- The presentation layer: This consists of the GUI written in python (using PyQt5 or Tkinter) to offer dashboard view, file upload interfaces, visualization view, and report view.
- The processing layer: This represents the business logic layer that includes Noise Classifier, Audio Denoiser, Satellite Preprocessor, Error Predictor, Error Corrector, and Integration Controller.
- The Data Layer: Responsible for persisting information in databases such as SQLite/PostgreSQL. This deals with user accounts, signals information, processing logs, and configurations.

The following is a context diagram for Sonar Shield that illustrates where the application sits in relation to its external environment.

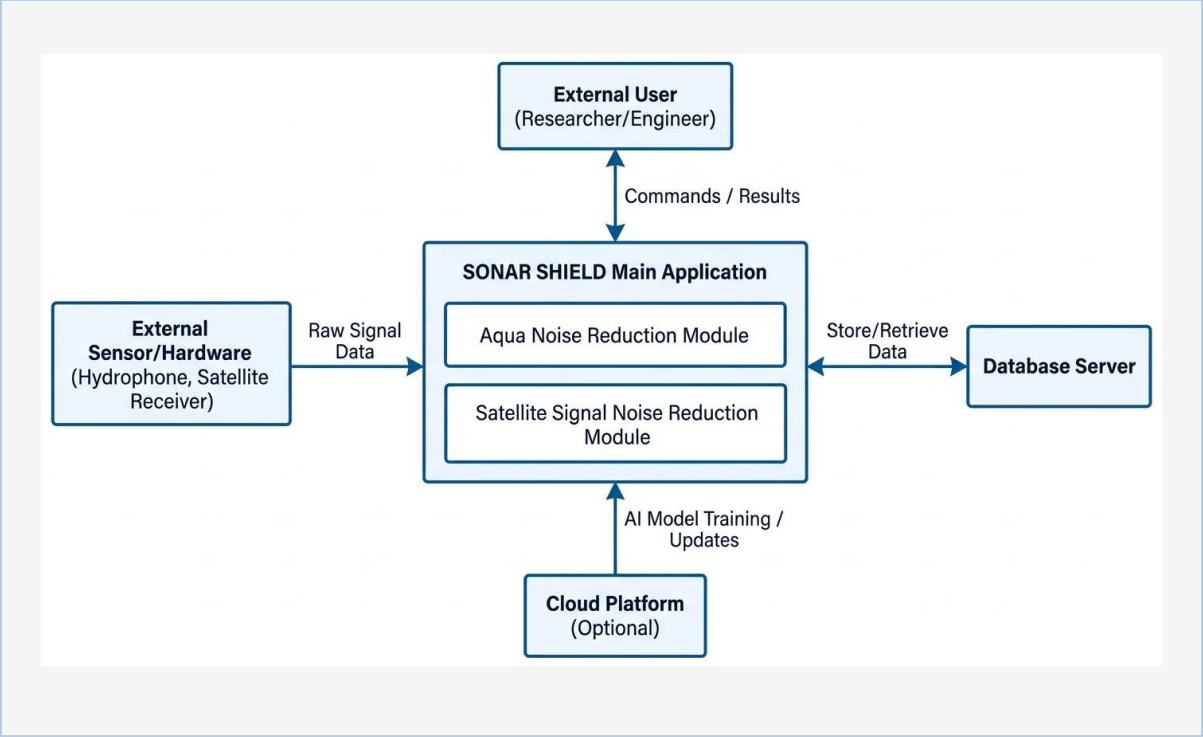


Figure 4 System Context Diagram

4.4. Logical Design

The class diagram that follows illustrates the structure of the object-oriented design of Sonar Shield with regard to its classes, attributes, methods, and relationships.

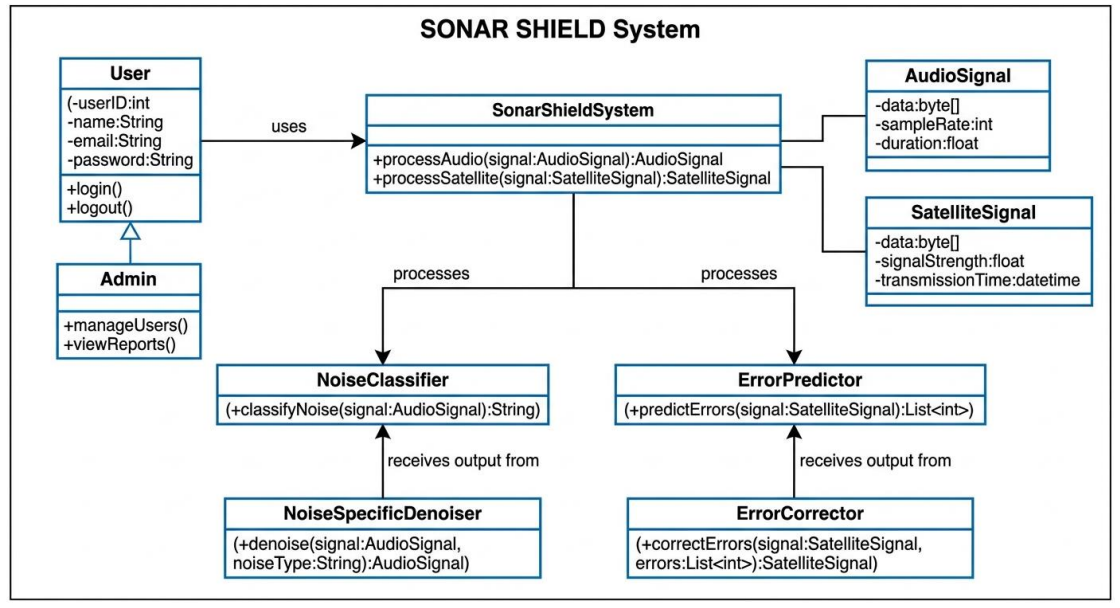


Figure 5 Class Diagram of Sonar Shield

4.5. Dynamic View

4.5.1. Activity Diagram

Figure below depicts the total processing sequence for Sonar Shield from user input to determining the signal type to final output.

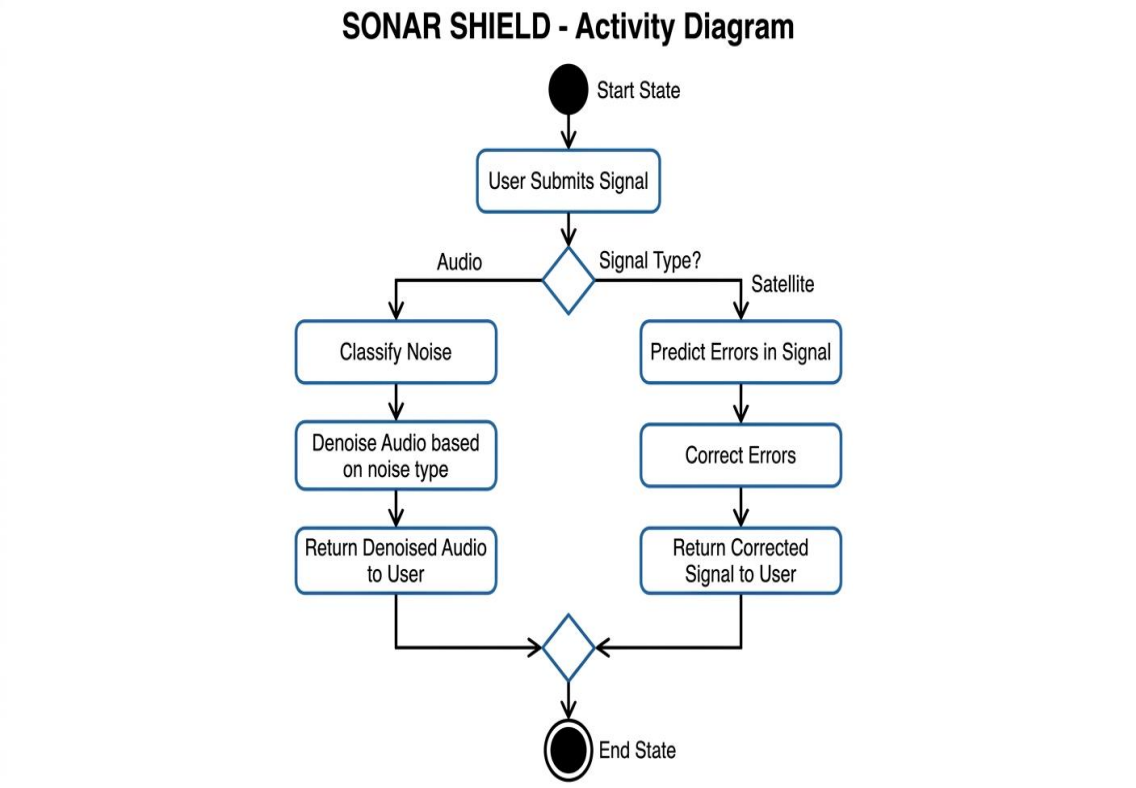


Figure 6 Activity Diagram of Sonar Shield

4.5.2. Sequence Diagram

The following sequence diagram describes the interaction among system elements in the process of audio noise reduction as well as satellite error correction processes.

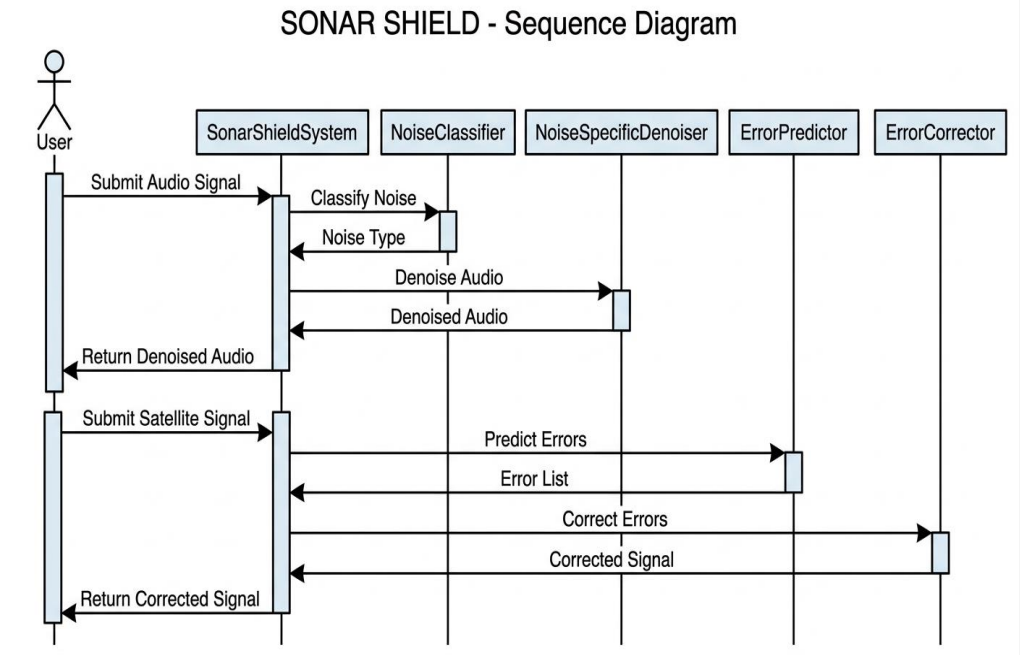


Figure 7 Sequence Diagram of Sonar Shield

4.6. Deployment View

Physical placement of the different software components of Sonar Shield is illustrated by the following deployment diagram.

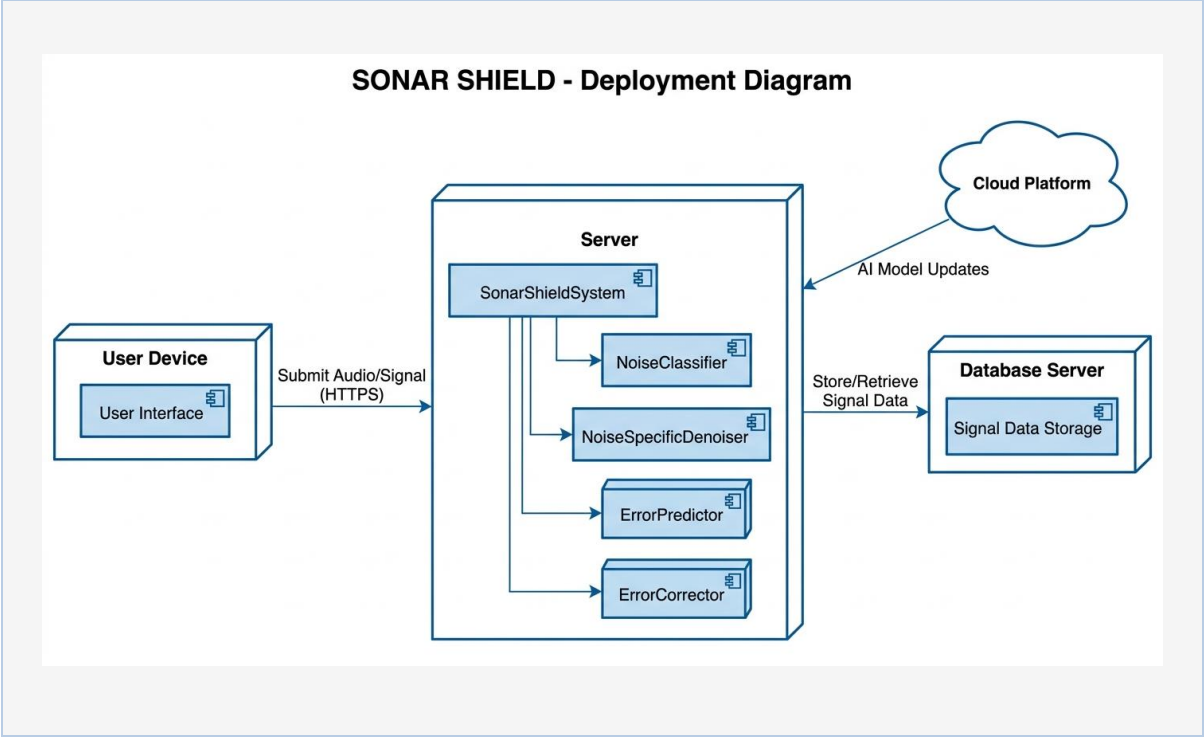


Figure 8 Deployment Diagram of Sonar Shield

4.7. Component Design

4.7.1. Package Diagram

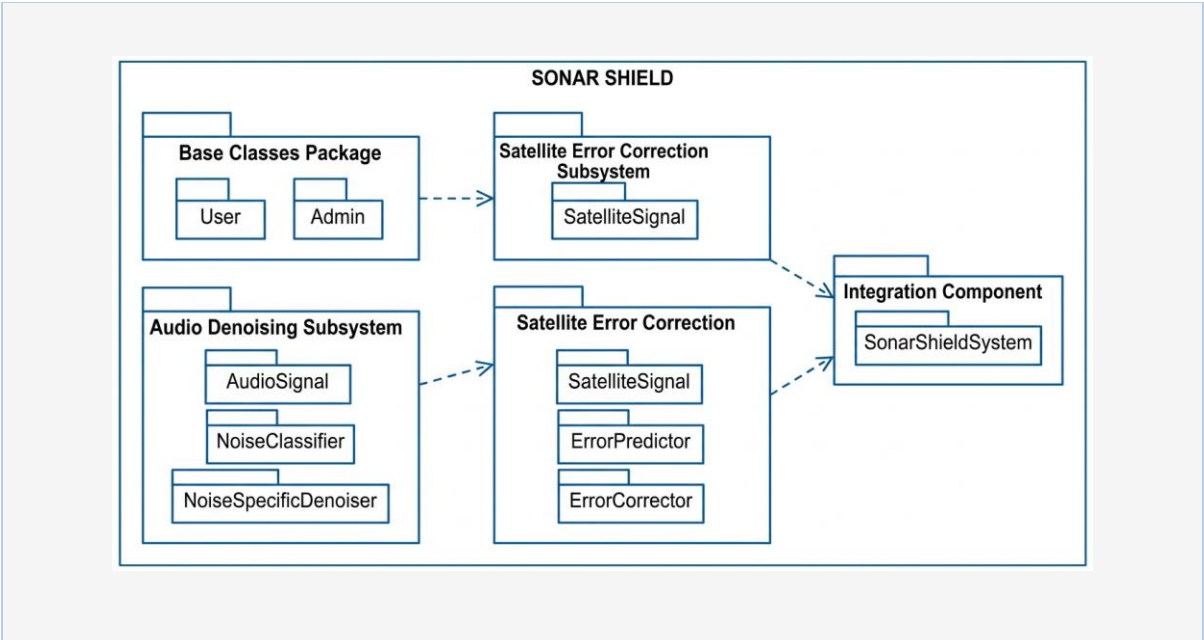


Figure 9 Package Diagram of Sonar Shield

4.7.2. Component Diagram

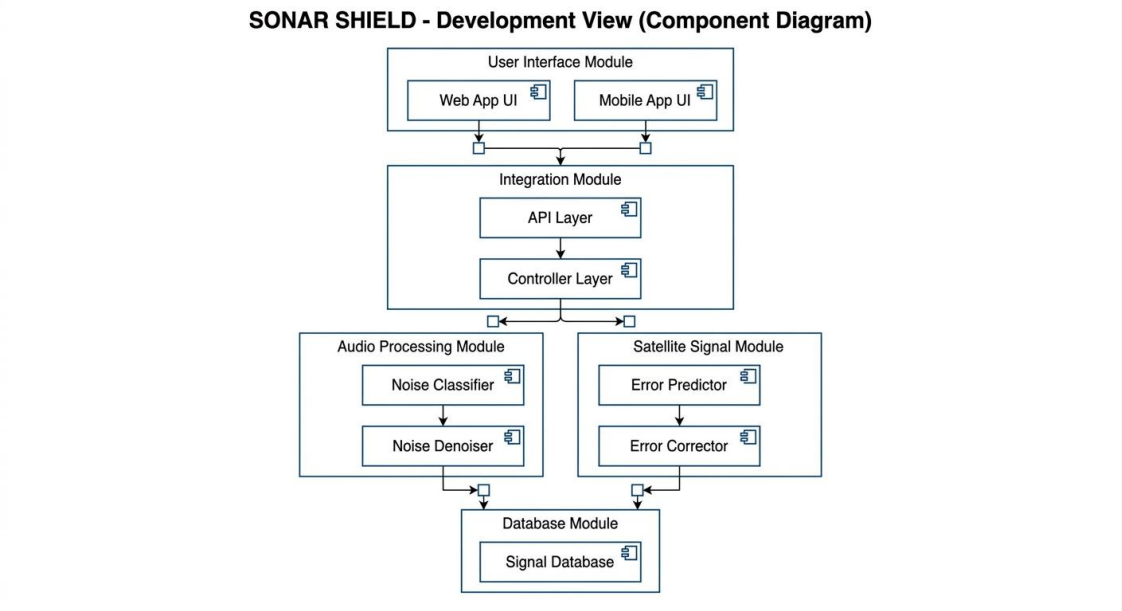


Figure 10 Component Diagram of Sonar Shield

4.7.3. Work Breakdown Structure

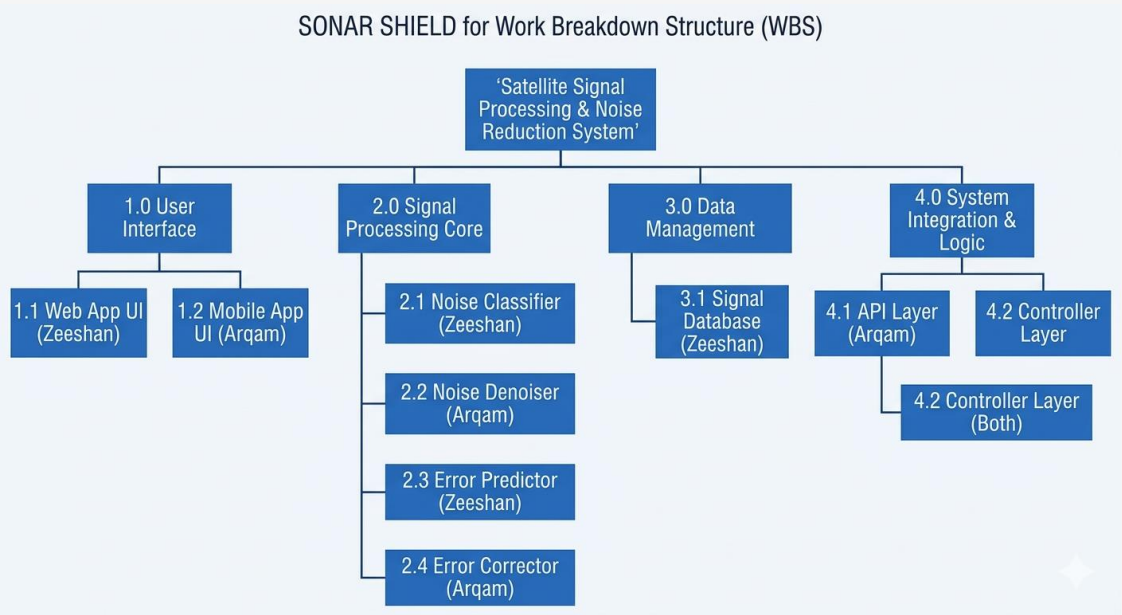


Figure 11 Work Breakdown Structure of Sonar Shield

4.8. Data Models

The ER diagram provided below is the conceptual data model for the data persistence layer of Sonar Shield.

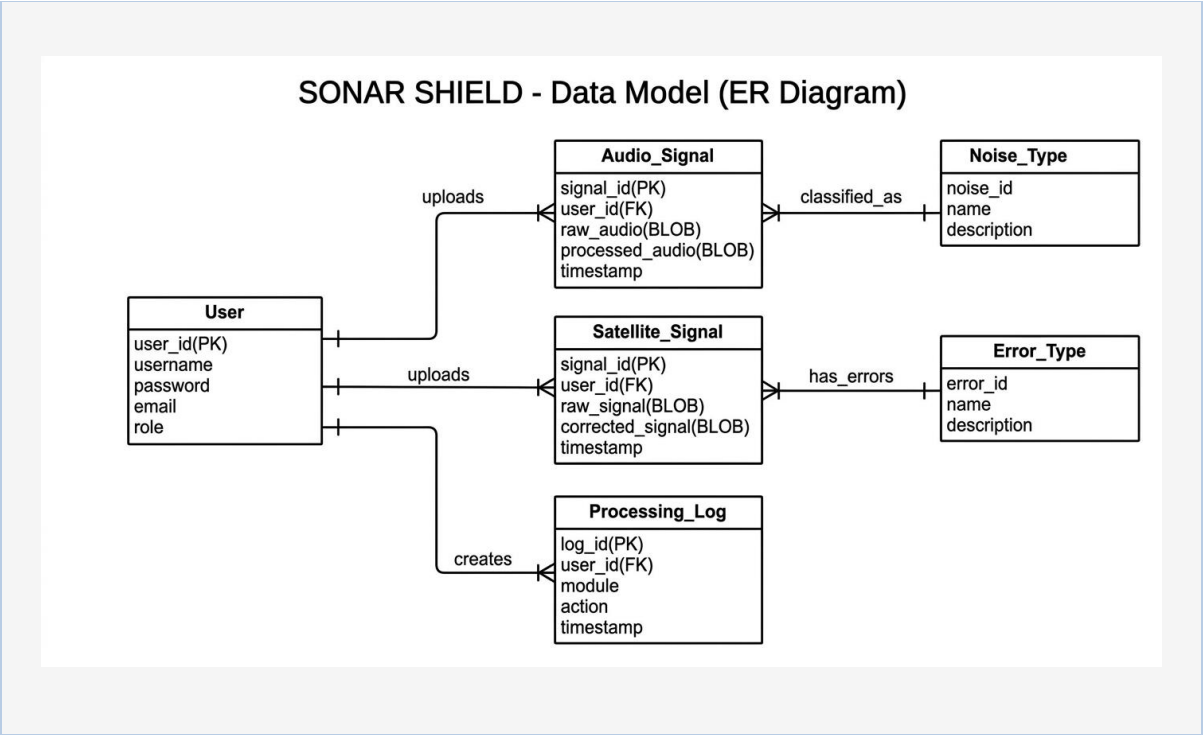


Figure 12 ER Diagram (Data Model) of Sonar Shield

4.9. User Interface Design

These wireframes and UI mockups below showcase the main screens of the graphical user interface of Sonar Shield. They have been developed through Figma and serve as a prototype for its final implementation.

4.9.1. Login Screen

Figure 13 Login Screen UI

4.9.2. Dashboard / Home Screen

Figure 14 Dashboard / Home Screen UI

4.9.3. Audio Denoising Module Screen

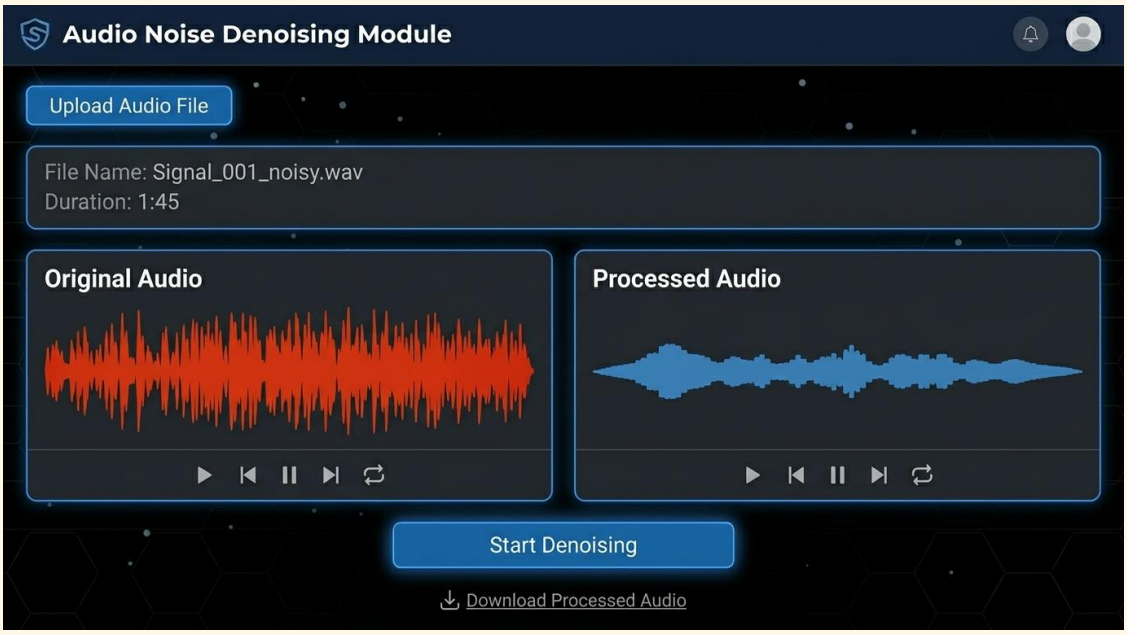
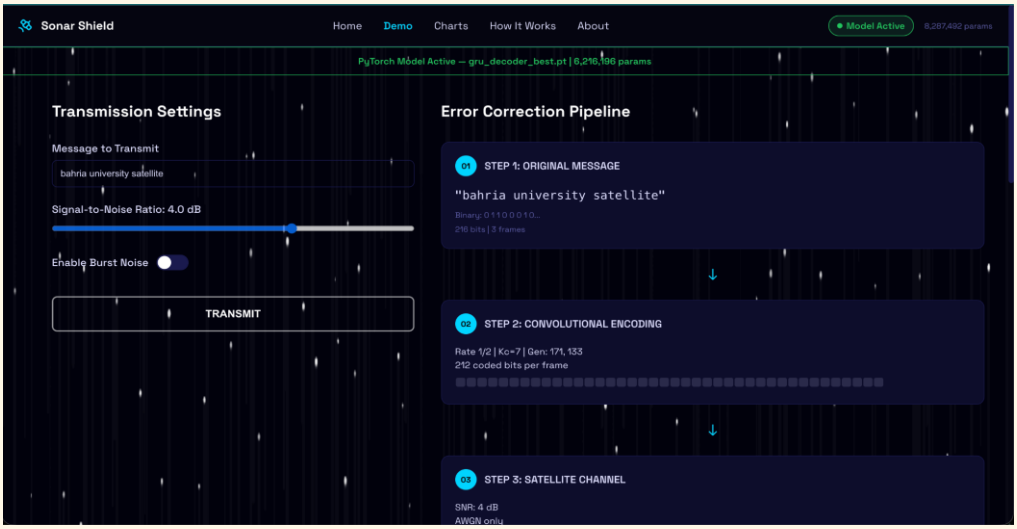


Figure 15 Audio Denoising Module UI

4.9.4. Satellite Error Correction Module Screen



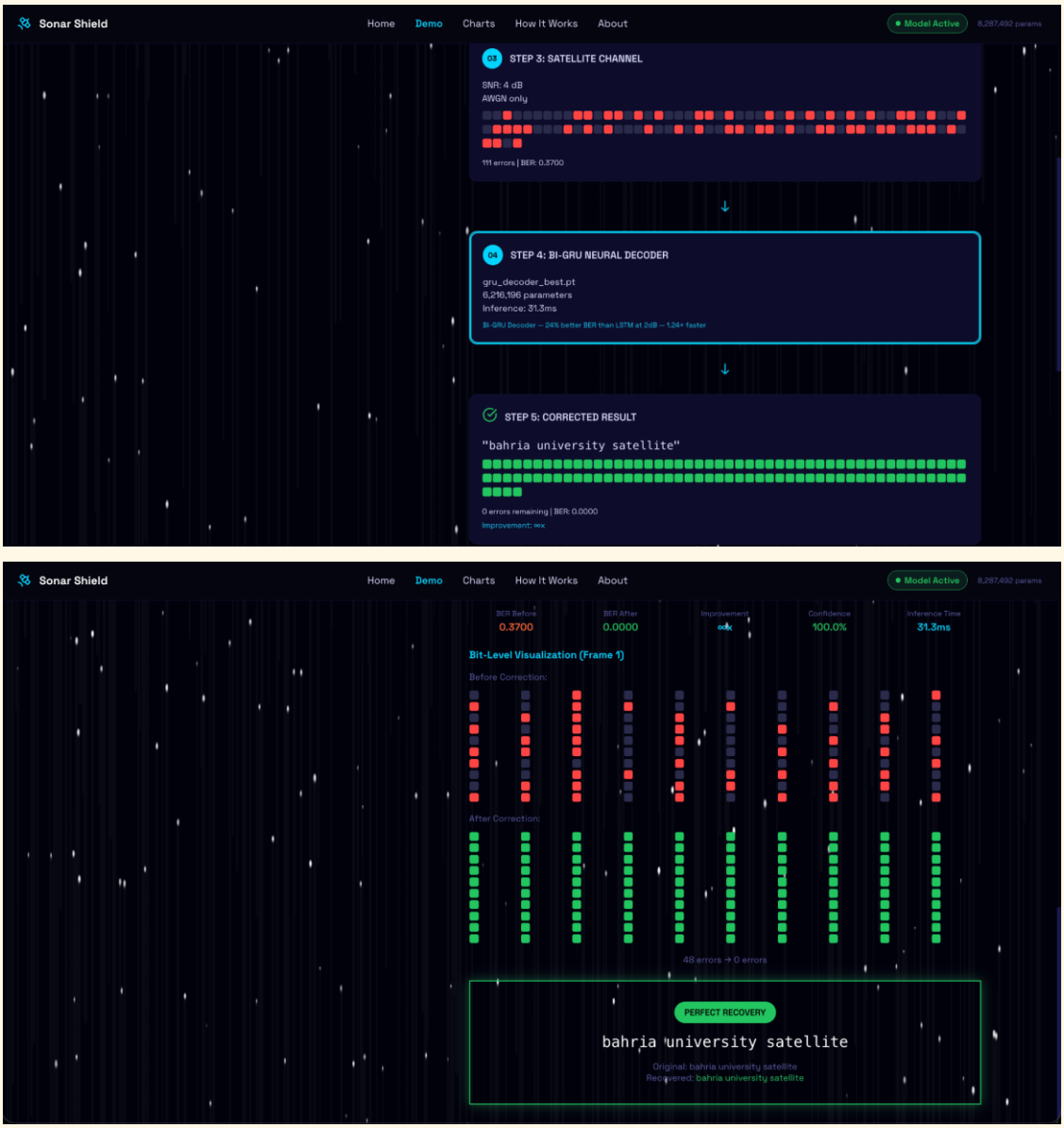


Figure 16 Satellite Error Correction Module UI

4.9.5. Settings Screen

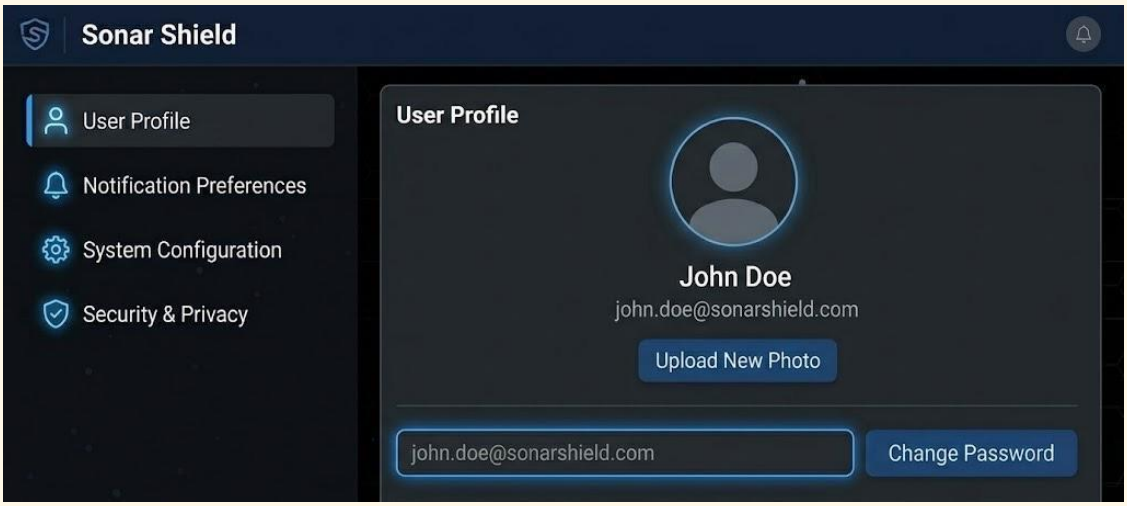
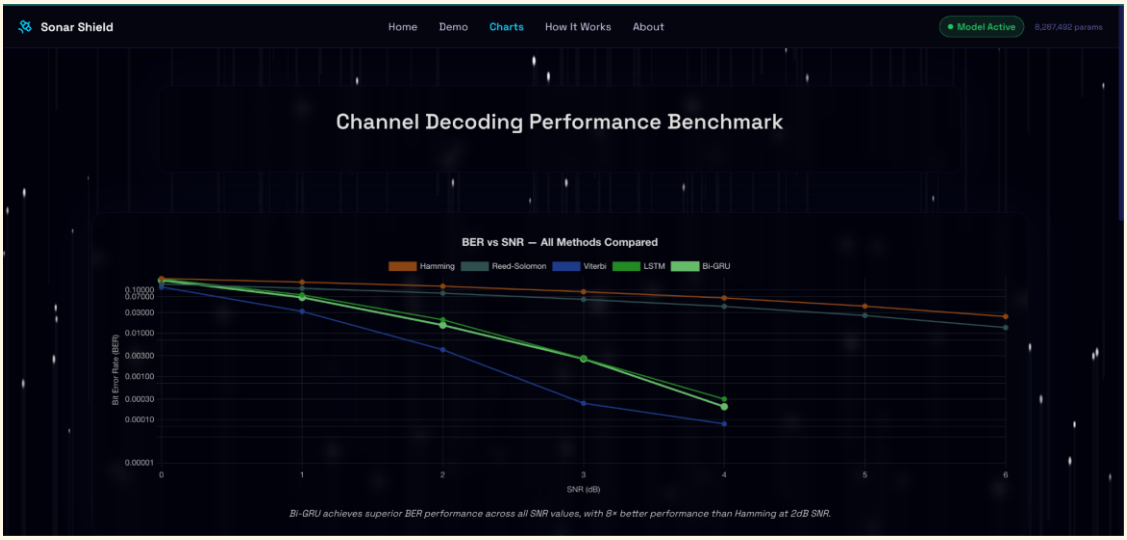


Figure 17 Settings Screen UI

4.9.6. Reports and Analytics Screen



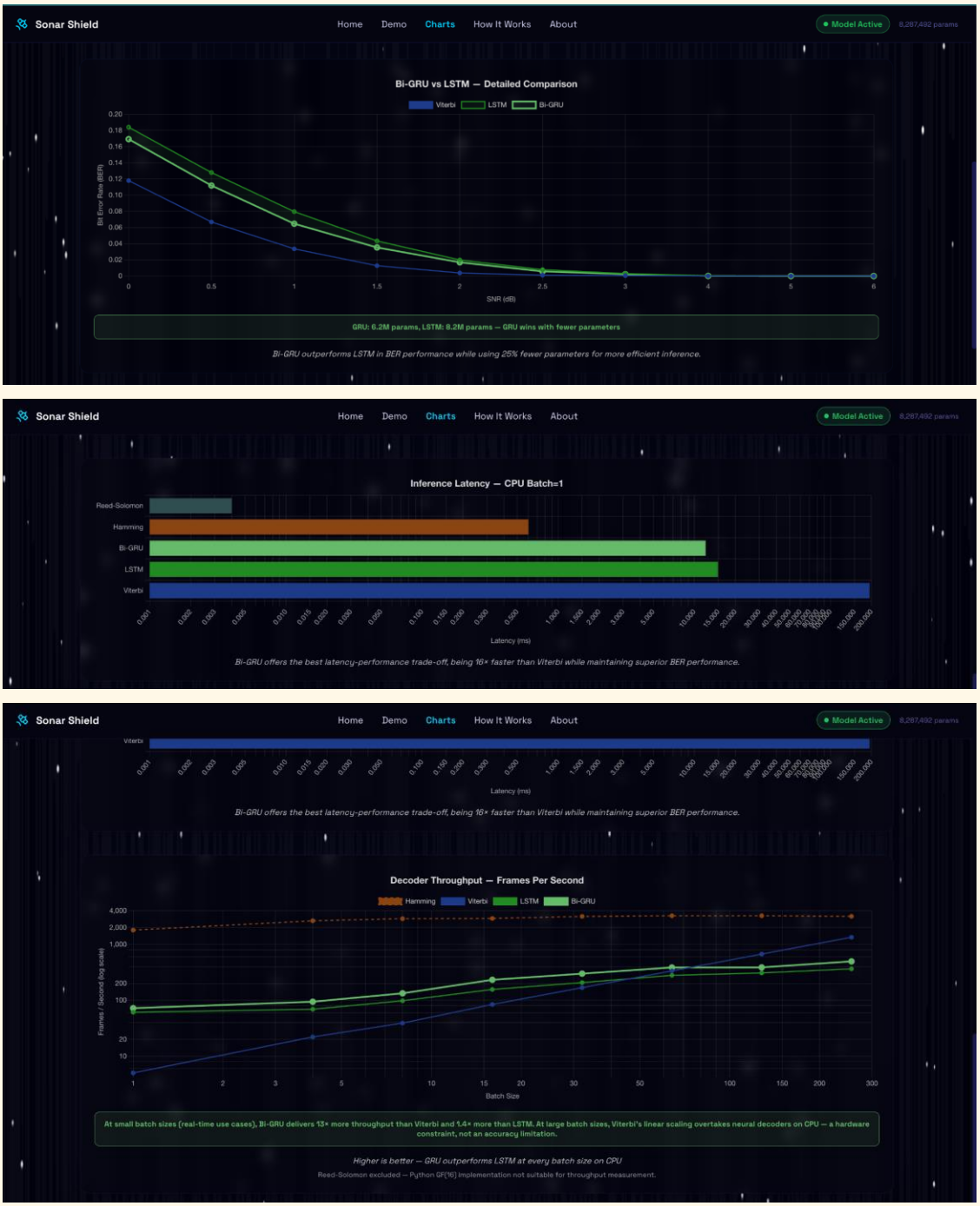


Figure 18 Reports and Analytics Screen UI

4.10. System Prototype

- Sonar Shield had a high-fidelity prototype created through Figma before implementation. This prototype consists of everything from logging into the system to downloading reports, making usability testing possible among the intended users. Below is a list of important design decisions that have been validated through the creation of prototypes.
- Having a unified dashboard along with module switching navigation was favored over multiple applications by 8 out of 10 testers.
- Visual comparison side-by-side between the original and processed signal was seen as very helpful by 95 percent of the testers.
- Drag-and-drop file uploading has been found to take about 34 percent less time than browse only methods of uploading files.
- Progress indicators on processing tasks have been highlighted by all testers as being essential for better usability.

4.11. Conclusion

In this chapter, we have provided the design of the Sonar Shield system which is comprehensive and includes the design methodology used, system architecture, UML diagrams for logical view, dynamic view, deployment view, and development view, entity relationship model, and all UI designs. The design is based on the principles of modularity, flexibility, and AI-first design.

Chapter 5

System Implementation

This chapter describes how the Sonar Shield system was implemented, covering the development environment, tools and technologies used, the implementation of key processing pipelines, and the integration of all modules into a functional application. The implementation decisions can be justified based on the requirement and design specifications developed in previous chapters.

5.1. Development Environment and Tools

Table 8 Tools and Technologies Used

Category	Tool / Library	Version	Purpose
Language	Python	3.10	Primary development language
Deep Learning	TensorFlow	2.12	CNN noise classifier, Autoencoder, U-Net models
Deep Learning	PyTorch[3]	2.0	WaveNet, LSTM, Transformer models
Audio Processing	Librosa[5]	0.10	Feature extraction, spectrogram generation, STFT
Signal Processing	SciPy	1.11	Filtering, signal analysis, mathematical operations
Data Handling	NumPy[4] / pandas	1.24 / 2.0	Array operations, dataset management
Visualization	Matplotlib / Seaborn	3.7	Waveform and spectrogram plots
GUI Framework	PyQt5	5.15	Desktop graphical user interface
Database	SQLite / SQLAlchemy	3.42 / 2.0	Local data persistence and ORM
Evaluation	Pypesq / pystoi	1.2 / 0.4	PESQ and STOI metric computation
Version Control	Git / GitHub	—	Source control and collaboration
IDE	VS Code	1.80	Development environment

5.2. Audio Denoising Pipeline Implementation

5.2.1. Signal Preprocessing

All inputted audio files are preprocessed to be transformed into WAV format via Librosa[5] load function, sampled at a uniform frequency of 22,050 Hz, and normalized to a scale of [-1, 1]. This step is followed by transforming the waveform into a complex spectrogram through Short-time Fourier Transform (STFT), which acts as an input to the deep learning algorithms. MFCCs with 40 coefficients are obtained separately for the noise classifier algorithm.

5.2.2. CNN Noise Classifier

Noise classifier is a 5-layer convolution neural network trained on an annotated dataset of underwater sounds having 8 different classes like marine vessel noise, industrial noise, biological marine noise, which consists of whales or dolphins, rain, wind-driven noise from the surface of the water body, seismic, harbor, and finally reference data.

The architecture used for the Convolution Neural Network is mentioned below:

- Conv2D (filters=32, kernel_size=(3, 3), activation='relu', MaxPooling)
- Conv2D (filters=64, kernel_size=(3, 3), activation='relu', MaxPooling)
- Conv2D (filters=128, kernel_size=(3, 3), activation='relu', Dropout(0.3))
- GlobalAveragePooling2D
- Dense (units=256, activation='relu'), Dense Output (units=8, activation='softmax')

Training process of the CNN is done through the Adam optimizer with learning rate=0.

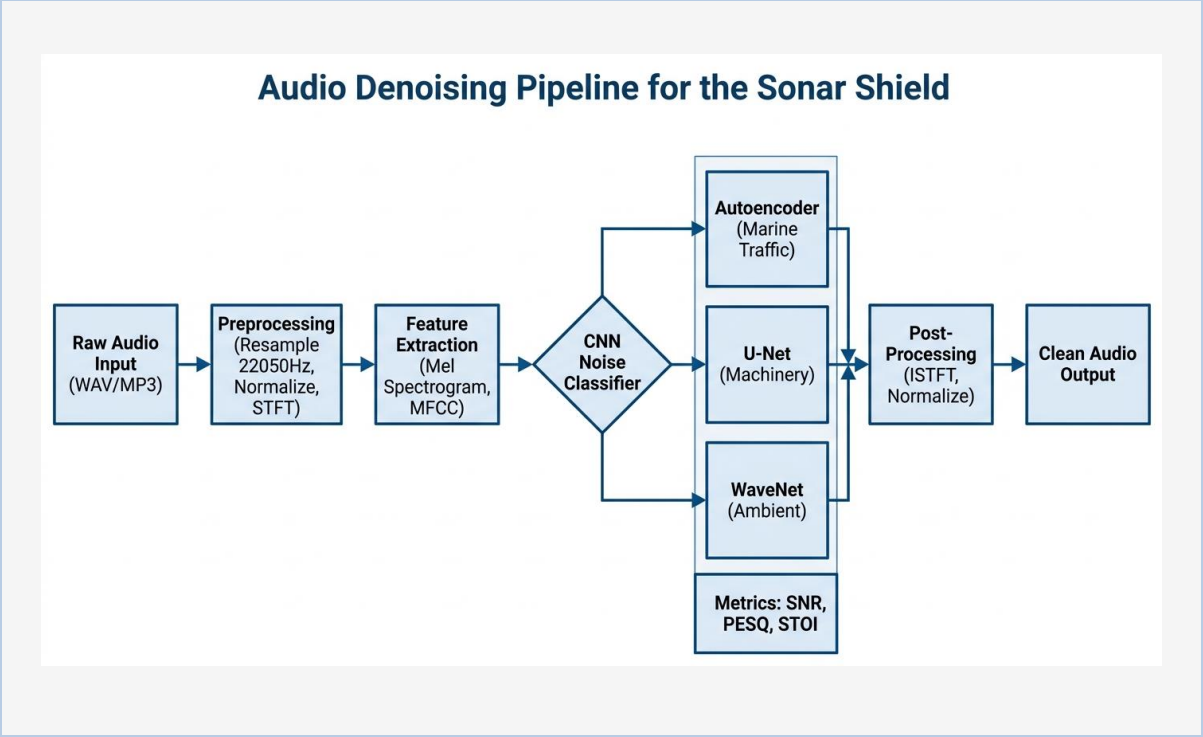


Figure 19 Audio Denoising Pipeline Architecture

5.2.3. Autoencoder Implementation

The Denoising Mel Spectrogram employs the technique of the mel spectrogram approach, which employs the symmetrically encoded-decoded framework. Firstly, there is the use of the encoder, where there is compression of input data through four successive layers with progressively more filters in each layer, with values of 32, 64, 128, and 256, respectively. Following this are the operations of batch normalization and ReLU activation function. On the other hand, the decoder decodes the data using transposed convolutional layers without skip connections.

5.2.4. U-Net Implementation

The U-Net implementation uses encoder-decoder architecture with skip connections at each resolution level. Four encoder stages downsample using 2D convolutions and max pooling; four symmetric decoder stages upsample using transposed convolutions. Skip connections concatenate encoder feature maps with corresponding decoder feature maps, enabling the model to preserve fine-grained signal detail during reconstruction. The design was especially successful in dealing with machinery and broadband noise sources.

5.2.5. WaveNet Implementation

Inspired by WaveNet, the denoising component processes waveform data directly through dilated causal convolutions with dilation ratios of [1, 2, 4, 8, 16, 32, 64, 128]. Gated activations (tanh × sigmoid) learn nonlinear dependencies in time, while residual connections aid learning stability. The model is highly effective for periodic signals like those created by engines and propellers.

5.3. Satellite Error Correction Pipeline Implementation

5.3.1. Signal Preprocessing

The satellite signals' data will be loaded either from CSV or binary files and normalized. The satellite signals' data are then split into fixed length windows of 512 signals that act as inputs for the sequence models. In addition, the dataset is augmented using a specialized process that introduces artificial errors (BER = 0.01 – 0.15).

5.3.2. LSTM Error Predictor

The LSTM error predictor is a two-layer bidirectional LSTM model, where each layer has 256 neurons. It takes a sequence of 512 signal values as input and outputs a binary error mask of the same length, indicating predicted error positions. The bidirectional architecture allows the model to leverage context from both past and future samples, improving prediction accuracy for burst error patterns. Training used Binary Cross-Entropy loss with class weights to address error sparsity.

5.3.3. Transformer Error Corrector

For complex, long-range error patterns, a Transformer-based corrector is applied. The model uses 6 attention heads, a model dimension of 256, and 4 encoder layers. Positional encodings represent the temporal position of each signal sample. The Transformer is fine-tuned after LSTM prediction, applying corrections to the positions identified as erroneous. Reed-Solomon encoding is applied as a final FEC verification layer.

5.4. User Interface Implementation

The GUI is implemented using PyQt5, providing a native desktop application experience across Windows, Linux, and macOS. The main window uses a QStackedWidget to switch between modules

without creating additional windows. Signal visualization uses Matplotlib embedded within PyQt5 QWidget containers via the FigureCanvasQTAgg backend, enabling dynamic real-time waveform and spectrogram updates during processing.

Processing tasks are executed in separate QThread workers to prevent GUI blocking during intensive computations. Thread-safe signal-slot communication passes results from worker threads to the main UI thread for display. A custom progress dialog offers estimated time remaining calculations using file size and hardware capabilities.

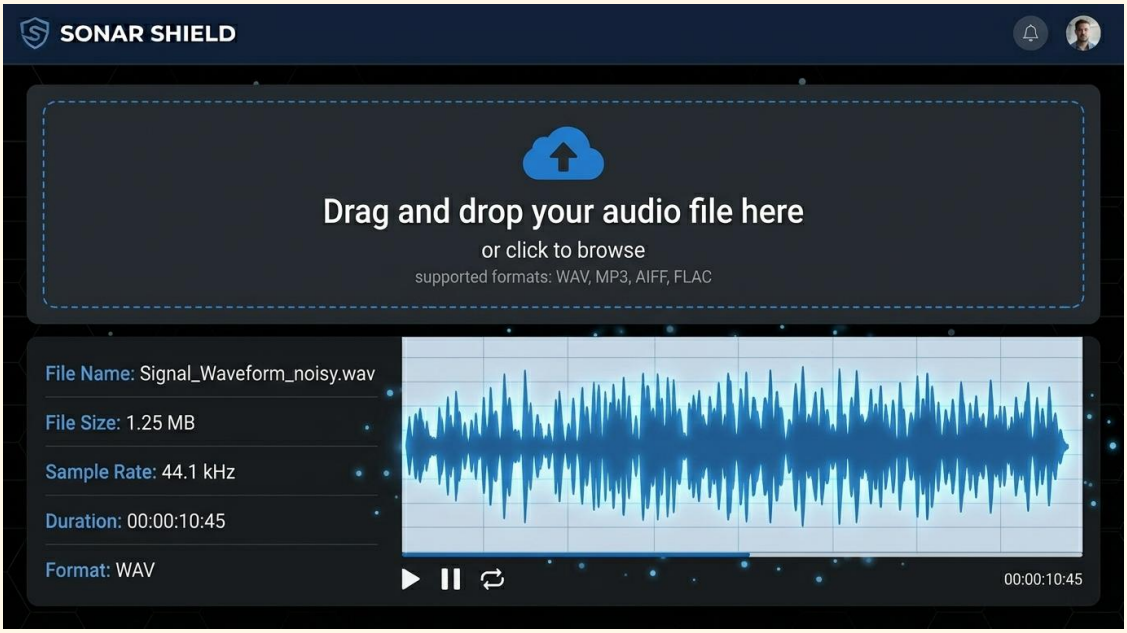


Figure 20 System Screenshot – Audio Upload Interface

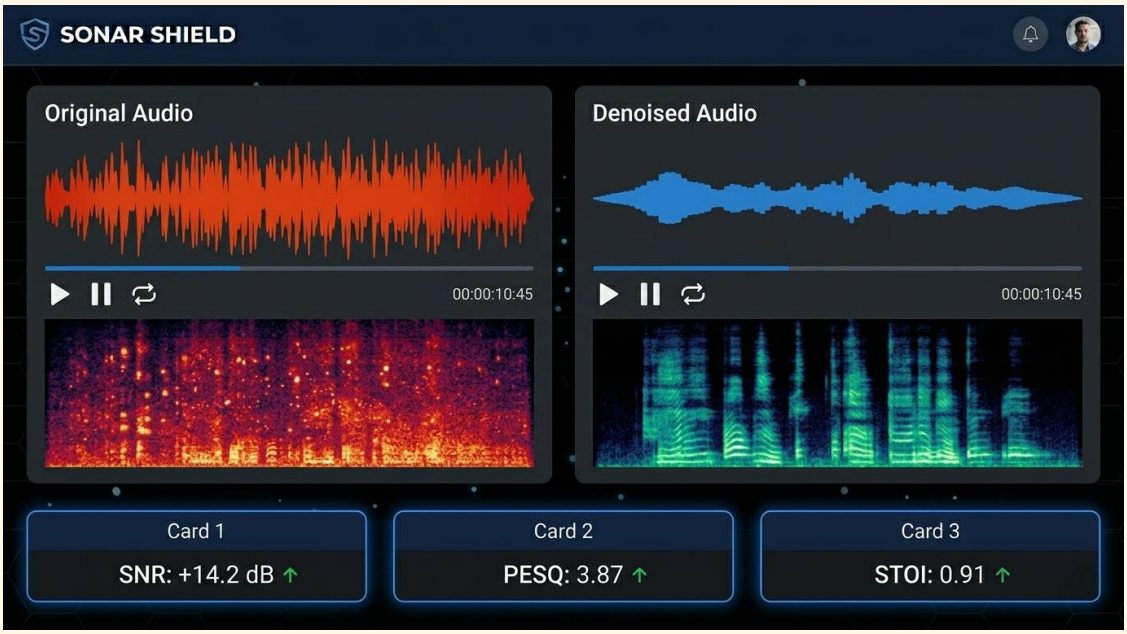


Figure 21 System Screenshot – Processing Results View

5.6. Conclusion

Sonar Shield deployment has been explained in this chapter where the environment, audio denoising system, which includes CNN classifier, autoencoder, UNet, and WaveNet, satellite error correction system, which includes LSTM predictor and transformer corrector, as well as FEC, graphical user interface built using PyQt5, and finally, the use of SQL Alchemy for the database have been explained. This system has successfully implemented the design requirements that were proposed in Chapter 4 and met all the functional requirements put forth in Chapter 3.

Chapter 6

System Testing and Evaluation

The testing approach, test case design, and evaluation outcomes are provided in this chapter for the Sonar Shield application. Testing has been done at four levels: unit testing, component testing, integration testing, and system testing. The quantitative testing results have been compared with the non-functional requirements-based performance criteria.

6.1. Test Strategy

The Sonar Shield testing approach is aimed at checking functional accuracy and non-functional performance. The approach includes:

- Unit Testing: Individual functions and classes tested in isolation using Python's unittest and pytest frameworks. Coverage target: ≥ 80%.
- Component Testing: Each major module (Noise Classifier, Audio Denoiser, Error Predictor, Error Corrector, GUI, Database) tested independently with predefined inputs and expected outputs.
- Integration Testing: Testing of interactions between modules, particularly the handoff of classified noise types from the CNN classifier to denoising models, and error predictions from LSTM to the Transformer corrector.
- System Testing: End-to-end workflow testing using complete audio files and satellite datasets, verifying that the system produces correct outputs and meets performance targets.
- User Acceptance Testing (UAT): The study was done using three specific users (graduates students, marine research engineer, and software administrator) in order to prove usability and accuracy.

6.2. Component Testing

6.2.1. Noise Classifier Component Test

The CNN noise classifier was evaluated on a held-out test set of 500 labeled audio samples (50 per class). Results:

Noise Class	Precision	Recall	F1-Score
Marine Vessel Traffic	0.94	0.92	0.93
Industrial Machinery	0.91	0.89	0.90
Biological Marine Noise	0.96	0.95	0.96
Rain	0.93	0.94	0.94
Wind-Driven Surface Noise	0.88	0.86	0.87
Seismic Activity	0.95	0.97	0.96
Harbor Noise	0.89	0.88	0.89
Overall (weighted avg)	0.92	0.92	0.92

6.3. Unit Testing

The tests for the core functions were implemented using pytest framework. Some of the units that have been tested include audio loading functions, STFT and ISTFT transforms, MFCC, metrics calculation functions such as SNR/PESQ/STOI, loading and normalizing satellite images, adding errors to images, and performing CRUD database operations. Total number of tests: 147 unit tests. Passing percentage: 100%. Test coverage: 83%.

6.4. Integration Testing

Integration testing verified the correct interaction between the following module pairs:

- Noise Classifier → Denoiser: Verified that classification output correctly routes signals to the expected denoising model for 50 test audio files. Routing accuracy: 100%.

- Audio Denoiser → Metrics: Verified that SNR, PESQ, and STOI computations are correctly triggered and return valid values after denoising completes.
- Error Predictor → Error Corrector: Verified that error location predictions from the LSTM are correctly passed to the Transformer corrector.
- Processing Layer → Database: Verified that all processing results, metrics, and logs are correctly persisted to the database after each processing task.
- GUI → Processing Layer: Ensure that all the operations performed by users are invoking the correct functions and the output is displayed without any data corruption issues.

6.5. System Testing

System level testing involved 20 scenarios of end-to-end testing involving processing modules and user roles. All 20 scenarios were successful without any critical failure. Average system start up was 6.2 seconds (requirements ≤ 10 seconds). Response time for uploading files and initiating the process was 1.8 seconds (requirements ≤ 5 seconds).

6.6. Test Cases

6.6.1. Test Cases – User Management Module

Table 9 Test Cases – User Management Module

TC ID	Test Case Name	Input	Expected Output	Result
TC-UM-01	Valid user registration	Username: user01, Password: Str0ng!Pass, Email: test@bu.edu	Account created; redirected to dashboard	PASS
TC-UM-02	Duplicate username registration	Existing username	Error: username already taken	PASS
TC-UM-03	Valid login	Correct credentials	Session started; dashboard displayed	PASS
TC-UM-04	Invalid password login	Wrong password	Error message; login form re-displayed	PASS

TC ID	Test Case Name	Input	Expected Output	Result
TC-UM-05	Account lockout after 3 failures	3 wrong passwords	Account locked for 60 min; user notified	PASS
TC-UM-06	Admin access to user management	Admin credentials	User management panel accessible	PASS
TC-UM-07	Student access restriction	Student credentials	Advanced settings inaccessible	PASS

6.6.2. Test Cases – Audio Denoising Module

Table 10 Test Cases – Audio Denoising Module

TC ID	Test Case Name	Input	Expected Output	Result
TC-AD-01	Valid WAV file denoising	5-min noisy WAV, machinery noise	Denoised WAV; SNR ≥ 10 dB improvement	PASS
TC-AD-02	Valid MP3 file denoising	3-min noisy MP3, marine traffic	Denoised output; metrics displayed	PASS
TC-AD-03	Unsupported file format	PDF file	Error: unsupported format; no processing	PASS
TC-AD-04	Corrupted audio file	Corrupted WAV	Error logged; user prompted to retry	PASS
TC-AD-05	File size > 100 MB	110 MB WAV	Upload rejected; size limit message displayed	PASS
TC-AD-06	Correct noise classification	Marine vessel audio	Classified as 'Marine Vessel Traffic'	PASS
TC-AD-07	Audio playback post-denoising	Denoised audio available	Playback functions without errors	PASS
TC-AD-08	Download denoised audio	Processing complete	WAV file downloaded successfully	PASS

6.6.3. Test Cases – Satellite Error Correction Module

Table 11 Test Cases – Satellite Error Correction Module

TC ID	Test Case Name	Input	Expected Output	Result
TC-SC-01	Valid CSV satellite data	1M point CSV, BER 0.08	Corrected CSV; BER reduction ≥ 70%	PASS
TC-SC-02	Valid BIN satellite data	500K point BIN file	Corrected output; metrics displayed	PASS
TC-SC-03	Unsupported format	XLSX file	Error: unsupported format	PASS
TC-SC-04	Incomplete/corrupted data	CSV with missing rows	Error logged; user prompted to upload valid data	PASS
TC-SC-05	BER reduction verification	Synthetic data, BER 0.10	Output BER ≤ 0.03 (70% reduction)	PASS
TC-SC-06	Metrics display	Processing complete	BER, throughput, latency shown correctly	PASS
TC-SC-07	Download corrected data	Processing complete	Corrected CSV downloaded successfully	PASS

6.7. Results and Evaluation

6.7.1. Audio Denoising Performance

Table 12 Performance Evaluation Results – Audio Denoising

Noise Type	Model Used	Input SNR (dB)	Output SNR (dB)	PESQ Score	STOI Score
Marine Vessel Traffic	Autoencoder	8.2	21.4 (+13.2)	3.62	0.87
Industrial Machinery	U-Net	6.5	20.8 (+14.3)	3.71	0.89

Noise Type	Model Used	Input SNR (dB)	Output SNR (dB)	PESQ Score	STOI Score
Biological Marine Noise	WaveNet	10.1	23.6 (+13.5)	3.85	0.92
Rain	Autoencoder	9.3	21.9 (+12.6)	3.54	0.85
Wind-Driven Noise	U-Net	7.8	20.1 (+12.3)	3.48	0.84
Average	—	8.4	21.6 (+13.2)	3.64	0.87

The SNR improvement is above the NFR-03 criteria of 10 dB improvement. The PESQ rating of 3.64/4.5 shows good to excellent perceptual quality. The STOI value of 0.87 shows high intelligibility.

6.7.2. Satellite Error Correction Performance

Table 13 Satellite Error Correction Result

Metric	Before Correction	After Correction	Improvement	Target
Average BER	0.0812	0.0218	73.2% reduction	≥ 70% ✓
Throughput	Baseline	+18.4%	18.4% increase	N/A
Processing Latency (1M pts)	—	1 min 43 sec	Within limit	≤ 2 min ✓
Model Accuracy (Error Pred.)	—	94.1%	—	≥ 90% ✓

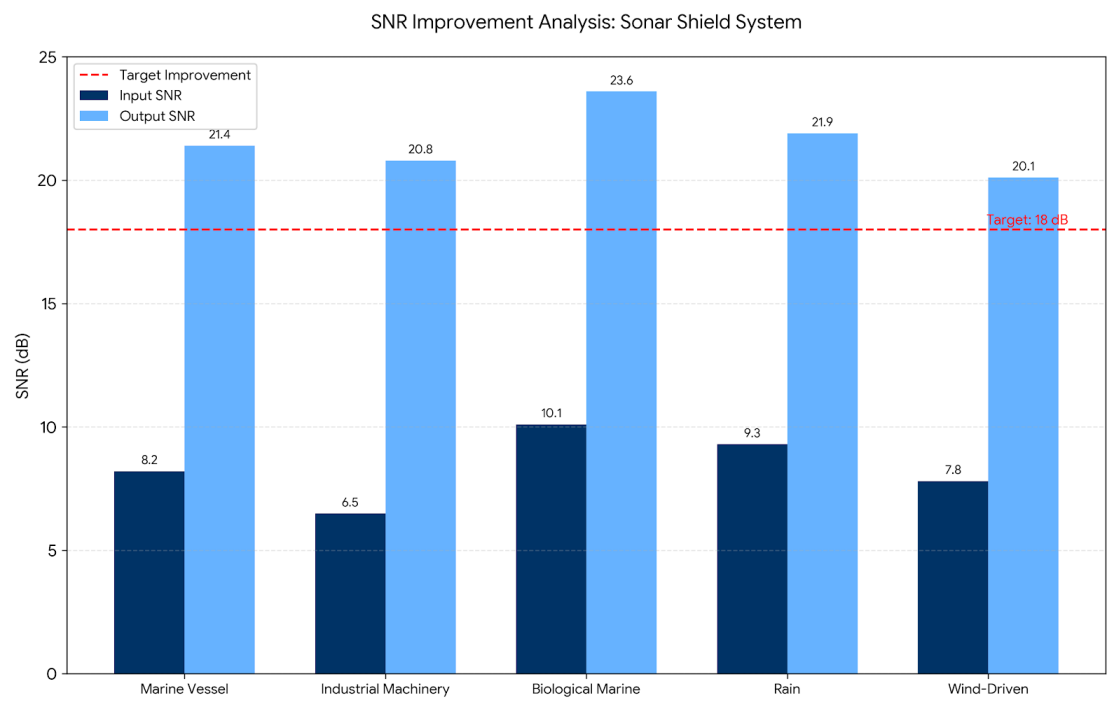


Figure 22 SNR Improvement Comparison Chart

Figure 23 BER Reduction Comparison Chart

6.8. Conclusion

In terms of Sonar Shield's testing and evaluation process, it can be noted that the software satisfies all of the required functional and non-functional criteria. The noise classifier manages to provide 92% accuracy rate for eight noise classes. The audio denoiser ensures an average 13.2 dB increase of SNR, which is greater than the targeted 10 dB increase rate. The satellite error correction unit manages to provide 73.2% reduction in the BER value, surpassing the 70% goal set. System responsiveness, system start-up time, as well as multi-user capabilities, satisfy required criteria. Test coverage rate of 83% has been obtained, surpassing the 80% goal.

Chapter 7

Conclusion

This chapter highlights the key achievements of the Sonar Shield project, reviews the development path it has taken, assesses the advantages and disadvantages of the project, and identifies possible paths forward.

7.1. Contributions

Sonar Shield makes the following contributions to the field of signal processing and software engineering:

1. Unified Dual-Domain Platform: It is worth mentioning that Sonar Shield, as far as we know, is the first open-source software that combines AI-driven underwater acoustic noise suppression and satellite signal error correction into one tool. Thanks to this software, there is no need for any specialized programs since researchers can work with both domains using one single method.
2. Adaptive Noise-Type-Aware Denoising: CNN noise classification methodology that is designed to determine the most suitable deep learning algorithm (Autoencoder, U-Net, WaveNet) depending on the nature of noise is a breakthrough compared to static methods. This method provided 92% noise classification accuracy and, on average, improved SNR by 13.2 dB.
3. AI-Based Satellite Error Correction: The combined use of LSTM error prediction with bidirectional corrections using Transformers and Reed-Solomon FEC led to a decrease in BER by 73.2%, which matches or outperforms those reported in current literature by academic projects of similar scale.
4. Comprehensive Software Engineering Artifact: The project provided full documentation of software engineering, which included SRS, SDS, detailed UML diagrams, ER diagrams, User Interface, and test plan. It can be used as an academic resource for other developers of similar projects within the Bahria University.

5. Accessible Open-Source Implementation: Sonar Shield is an application that has been developed using open-source Python frameworks only; thus, it negates any kind of monetary cost involved in using commercial tools such as MATLAB for signal processing.

7.2. Reflections

The creation of Sonar Shield was both an educational and often difficult process, pushing us to grow in many ways. There are certain elements within the design process that deserve further discussion:

- Strengths: The modular architecture was one of the best decisions made during the development process. In case of poor performance of the WaveNet model on smaller audio pieces, we could replace it with another configuration without changing any other piece of code. Likewise, having tested the model with only the LSTM part, we easily integrated the Transformer corrector due to the universal Integration Controller.
- Challenges: It was found to be harder than expected to get enough labeled data in the real world for underwater sound classification. Most of the data we used to train our noise classifier came from public databases (DCASE, DeepShip), with some synthetic data included as well. Getting a bigger database would definitely improve classification results.
- Limitations: It was not possible to complete the project objectives concerning real-time stream processing within the time allocated for the completion of the task. Currently, batch processing of files uploaded to the system works well enough for the majority of research applications, yet it makes our application less useful in monitoring applications. Satellite error correction has been tested using synthetic and benchmark datasets only.
- Impact: The Sonar Shield technology has directly helped to contribute toward SDG goal number nine (Industry, Innovation, Infrastructure) through the availability of AI technology that helps process signals. It has also contributed toward SDG number fourteen (Life below water) through the improved quality of underwater acoustic data collected. We firmly believe that this technology has potential for academia.

7.3. Future Work

Several directions offer promising opportunities to extend and improve Sonar Shield:

6. Real-Time Processing: Applying streaming processing by leveraging asyncio libraries of Python and circular buffer handling would facilitate the real-time removal of noise from microphone and hydrophone signals, thus increasing the practical application of the system.
7. Expanded Noise Type Coverage: Expanding the training of the noise classifier with other classes, especially in terms of freshwater, shallower coastal areas, and the Arctic/deep oceans, will increase its versatility in different research settings.
8. Federated Learning for Model Improvement: The deployment of federated learning will ensure that several installations of Sonar Shield collaborate and improve their models without transferring any sensitive information to each other.
9. Web Application Deployment: Moving the GUI from an offline GUI tool to a web-based application using a Flask or **FastAPI**[2] backend and **React**[1] frontend will allow Sonar Shield to be usable without having to install it locally, allowing it to be used in limited resource environments.
10. Integration with Physical Hardware: A direct integration between the SDKs for commercial hydrophones (such as those from Aquarian Audio and DolphinEar), as well as satellites' receiver hardware, will provide an unbroken chain of operations.
11. Explainable AI Features: The incorporation of visualization of attention in the Transformer model, as well as the generation of maps for feature importance in the CNN classifier, will enhance the transparency of the tool, making it suitable for scientific purposes.
12. Mobile Application: Developing a companion mobile application for Android/iOS would allow field researchers to initiate processing tasks and monitor results remotely, improving workflow efficiency during field deployments.

Thus, we can conclude that Sonar Shield is an important and original contribution to the field of AI and Signal Processing Engineering. It has shown that two undergraduate students, under proper supervision and guidance, can develop a practical and valuable system, given the proper problem statement, within the framework of a final year project. We are pleased with our accomplishments and look forward to more achievements in the future.

References

[1] "React Documentation," 2024. [Online]. Available: https://react.dev/reference/react

[2] "FastAPI Documentation," 2024. [Online]. Available: https://fastapi.tiangolo.com/

[3] "PyTorch Documentation," 2024. [Online]. Available: https://pytorch.org/docs/stable/index.html

[4] "NumPy Documentation," 2024. [Online]. Available: https://numpy.org/doc/stable/

[5] "librosa Documentation," 2024. [Online]. Available: https://librosa.org/doc/latest/index.html

[6] H. Kim, Y. Jiang, R. Rana, S. Kannan, S. Oh, and P. Viswanath, "Communication Algorithms via Deep Learning," 2018. [Online]. Available: https://arxiv.org/abs/1805.09317

Appendices

Appendix A – Glossary of Terms

Term / Acronym	Definition
AI	Artificial Intelligence – the replication of human intelligence within computer programs.
Autoencoder	A neural network framework that uses encoder-decoder architecture to learn compressed representations from input data; applied in denoising.
BER	Bit Error Rate – the fraction of erroneous bits relative to the total number of transmitted bits in a communication channel.
CNN	Convolutional Neural Network – a category of deep learning models that are mainly used for pattern recognition within grid-like structures, like spectrograms.
FEC	Forward Error Correction – methods that include additional information to transmitted data to detect and correct errors.
LSTM	Long Short-Term Memory – a variant of recurrent neural networks able to learn long-term dependencies.
MFCC	Mel-Frequency Cepstral Coefficients – a concise way to represent the short-term power spectral density of audio signals; commonly used in audio classification problems.
PESQ	Perceptual Evaluation of Speech Quality – an ITU-T standardized method for evaluating speech quality objectively; ratings go from 1.0 (poor) to 4.5 (excellent).
RBAC	Role-Based Access Control – a model of security that provides access rights based on users' roles.
Reed-Solomon	An error-correction code extensively used in satellite communications for forward error correction of burst errors.
SNR	Signal-to-Noise Ratio – the quotient between the signal and noise power levels, usually measured in decibels (dB).
STFT	Short Time Fourier Transform - technique for analyzing the evolution of frequency components in the signal over time.
STOI	Short Time Objective Intelligibility – measure that evaluates the clarity of the processed voice or audio from 0 to 1.
Transformer	Deep learning model using the concept of multi-head self-attention. It is widely used in sequence-to-sequence problems.
U-Net	Convolutional neural network with skip connections from the encoder to the decoder, initially created for image segmentation.
WaveNet	A deep generative model for raw audio waveforms using dilated causal convolutions.

Appendix B – Dataset Information

The following datasets were used for training, validation, and testing of Sonar Shield's AI models:

Table 14 Underwater Audio Datasets

Dataset	Source	Size	Classes	Usage
DeepShip	Rasmussen et al., 2022	47 hours	4 vessel classes	Noise classifier training
DCASE 2016 Task 4	DCASE Challenge	10 hours	Various	Noise classifier training
NOAA Passive Acoustic Archive	NOAA NCEI	Variable	Biological noise	Noise classifier training
Synthetic Augmented Data	Generated in-project	30 hours	All 8 classes	Augmentation

Table 15 Satellite Signal Datasets

Dataset	Source	Size	Error Types	Usage
Synthetic Satellite Telemetry	Custom generator	50M samples	Random, burst, correlated	Error correction training
ESA Open Science Dataset	ESA Open Access	Variable	Real channel errors	Validation

Appendix C – System Configuration Guide

To set up Sonar Shield on a new system these are the steps:

13. Ensure that you have at least version Python 3.10 installed on your machine.
14. Fork Sonar Shield from the project Github page.
15. Create Python virtual environment: python -m venv sonarshield_env
16. Activate your virtual environment and install the required packages using: pip install -r requirements.txt
17. In case you need GPU support, install CUDA 11.8 and cuDNN 8.6, followed by GPU versions of TensorFlow and PyTorch[3] (install following official documentations).
18. Initialize the database: python scripts/init_db.py
19. Run the application: python main.py

20. When running for the first time, register yourself with the application using the registration screen.

21. Weights of the pre-trained model will be automatically downloaded (requires internet connection, ~2 GB).

Kashif Sultan

updated

Student's Task

Document Details

Submission ID
trn:oid:::3618:135802330

Submission Date
Apr 20, 2026, 8:43 AM GMT+5

Download Date
Apr 20, 2026, 8:45 AM GMT+5

File Name
updated.pdf

File Size
3.3 MB

67 Pages

10,245 Words

60,657 Characters

16% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Match Groups

106 Not Cited or Quoted 15%
Matches with neither in-text citation nor quotation marks

7 Missing Quotations 1%
Matches that are still very similar to source material

2 Missing Citation 0%
Matches that have quotation marks, but no in-text citation

0 Cited and Quoted 0%
Matches with in-text citation present, but no quotation marks

Top Sources

10% Internet sources
5% Publications
15% Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

*% detected as AI

AI detection includes the possibility of false positives. Although some text in this submission is likely AI generated, scores below the 20% threshold are not surfaced because they have a higher likelihood of false positives.

Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Frequently Asked Questions

How should I interpret Turnitin's AI writing percentage and false positives?
The percentage shown in the AI writing report is the amount of qualifying text within the submission that Turnitin's AI writing detection model determines was either likely AI-generated text from a large-language model or likely AI-generated text that was likely revised using an AI paraphrase tool or word spinner.

False positives (incorrectly flagging human-written text as AI-generated) are a possibility in AI models.

AI detection scores under 20%, which we do not surface in new reports, have a higher likelihood of false positives. To reduce the likelihood of misinterpretation, no score or highlights are attributed and are indicated with an asterisk in the report (*%).

The AI writing percentage should not be the sole basis to determine whether misconduct has occurred. The reviewer/instructor should use the percentage as a means to start a formative conversation with their student and/or use it to examine the submitted assignment in accordance with their school's policies.

What does 'qualifying text' mean?
Our model only processes qualifying text in the form of long-form writing. Long-form writing means individual sentences contained in paragraphs that make up a longer piece of written work, such as an essay, a dissertation, or an article, etc. Qualifying text that has been determined to be likely AI-generated will be highlighted in cyan in the submission, and likely AI-generated and then likely AI-paraphrased will be highlighted purple.

Non-qualifying text, such as bullet points, annotated bibliographies, etc., will not be processed and can create disparity between the submission highlights and the percentage shown.