

DEEP LEARNING – BASED BIOLOGICAL SIGNAL STEGANOGRAPHY WITH MISSING DATA RECOVERY APPROACH

A PROJECT REPORT

Submitted by

EKANATHAN S A

(Reg. No. 202006018)

SAIVIJAY S

(Reg. No. 202006038)

RAJASEKAR M

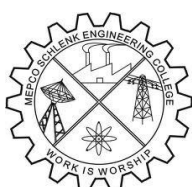
(Reg. No. 202006253)

*in partial fulfillment for the award of the degree
of*

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY



**DEPARTMENT OF INFORMATION TECHNOLOGY
MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI**
(An Autonomous Institution affiliated to Anna University, Chennai)



APRIL 2024

BONAFIDE CERTIFICATE

Certified that this project report titled **DEEP LEARNING – BASED BIOLOGICAL SIGNAL STEGANOGRAPHY WITH MISSING DATA RECOVERY APPROACH** is the bonafide work of **Mr.S.A.EKANATHAN (Reg. No: 202006018), Mr.S.SAIVIJAY (Reg. No: 202006038), Mr.M.RAJASEKAR (Reg. No: 202006253)** who carried out the research under my supervision. Certified further, that to the best of my knowledge, the work reported herein does not form part of any other project report or dissertation based on which a degree or award was conferred on an earlier occasion on this or any other candidate.

MENTOR

Dr. G.YOGARAJAN, M.E, Ph.D.,
Associate Professor,
Department of Information Technology,
Mepco Schlenk Engineering College,
Sivakasi-626005.
Virudhunagar Dt.
Tamilnadu.

HEAD OF THE DEPARTMENT

Dr.T.REVATHI, M.E., Ph.D.,
Senior Professor and Head,
Department of Information Technology
Mepco Schlenk Engineering College,
Sivakasi-626005.
Virudhunagar Dt.
Tamilnadu.

Submitted for Viva-Voce Examination held at **MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI (AUTONOMOUS)** on

Internal Examiner

External Examiner

ABSTRACT

ABSTRACT

In an era marked by escalating concerns over data security and privacy, the fusion of advanced encryption techniques with biological signal processing has become crucial. Telemedicine and remote patient monitoring rely heavily on the secure transmission of biological signals, particularly those capturing cardiac activity. However, transmitting these signals over networks poses security risks and potential data loss. To address these challenges, a novel framework titled "Deep Learning-based Biological Signal Steganography with Missing Data Recovery Approach" is proposed. The framework utilizes deep learning techniques to fortify the confidentiality and integrity of bio-signal data transmissions. Hermite functions are employed for robust encryption, wherein the biosignal undergoes a forward Hermite transform before secret data bits are embedded. A cryptographic hash function generates passwords based on the encrypted signal, ensuring data integrity by detecting alterations during transmission. Upon reception, the decryption process unveils concealed information and discerns missing data blocks. To address unreliable transmission channels, a pre-trained Multilayer Perceptron Neural Network (MLPNN) predicts missing data based on surrounding intact portions of the signal, ensuring accurate reconstruction for medical analysis. Experimental validation demonstrates the framework's efficacy and robustness, with high levels of confidentiality and minimal distortion to clinical information in encrypted biosignals. Password verification mechanisms rigorously ensure data integrity, while the MLPNN showcases remarkable success rates in real-world scenarios. In summary, the proposed approach offers enhanced security through Hermite function encryption, data integrity via password generation, and missing data recovery facilitated by the MLPNN. This comprehensive solution addresses critical concerns surrounding the secure and reliable transmission of biological signals in telemedicine and remote patient monitoring applications.

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

Apart from our efforts, the success of our project depends largely on the encouragement of many others. We take this opportunity to express our gratitude to the people who have been instrumental in the successful completion of our project. We would like to express our immense pleasure to thank our college management for giving required amenities regarding our project.

We would like to convey our sincere thanks to our respected Principal, **Dr.S.Arivazhagan, M.E., Ph.D.**, Mepco Schlenk Engineering College, for providing us with the facilities to complete our project.

We extend our profound gratitude and heartfelt thanks to **Dr.T.Revathi, M.E., Ph.D.**, Senior Professor and Head, Department of Information Technology for providing us with constant encouragement.

We are bound to thank our project coordinator **Dr.S.Rajesh, M.E., Ph.D.**, Professor, Department of Information Technology. We sincerely thank our project guide **Dr.G.Yogarajan, M.E., Ph.D.**, Associate Professor, Department of Information Technology, for his inspiring guidance and valuable suggestions to complete our project successfully.

The guidance and support received from all the faculty members and lab technicians of our department who contributed to our project was vital for the success of the project. We are grateful for their constant support and help.

We would like to thank our parents and friends for their help and support in our project.

TABLE OF CONTENTS

TABLE OF CONTENTS

| CONTENT | PAGE NO |
|---|---------|
| LIST OF TABLES | xii |
| LIST OF FIGURES | xiv |
| LIST OF SYMBOLS | xvi |
| LIST OF ABBREVIATIONS | xviii |
| 1. INTRODUCTION | 1 |
| 2. LITERATURE STUDY | 4 |
| 2.1 ON-DEVICE RELIABILITY ASSESSMENT AND PREDICTION OF MISSING PHOTOPLETHYSMOGRAPHIC DATA USING DEEP NEURAL NETWORKS | 5 |
| 2.2 SVD-BASED ROBUST IMAGE STEGANOGRAPHIC SCHEME USING RIWT AND DCT FOR SECURE TRANSMISSION OF MEDICAL IMAGES | 7 |
| 2.3 SECURING DATA IN INTERNET OF THINGS (IOT) USING CRYPTOGRAPHY AND STEGANOGRAPHY TECHNIQUES | 8 |
| 3. SYSTEM STUDY | 10 |
| 3.1 SCOPE | 11 |
| 3.2 PRODUCT FUNCTION | 11 |
| 3.2.1 DATA ACQUISITION | 12 |
| 3.2.2 PREPROCESSING | 12 |
| 3.2.3 ENCRYPTION FUNCTIONALITY | 12 |
| 3.2.4 DECRYPTION FUNCTIONALITY | 12 |
| 3.2.5 MISSING DATA RECOVERY FUNCTIONALITY | 13 |
| 3.2.6 INTEGRATION AND COMPATIBILITY | 13 |
| 3.2.7 SECURITY AND RELIABILITY | 13 |
| 3.3 SYSTEM REQUIREMENTS | 14 |
| 3.3.1 HARDWARE REQUIREMENTS | 14 |
| 3.3.1.1 ARDUINO BOARD OR MICROCONTROLLER | 15 |
| 3.3.1.2 COMPUTER OR EMBEDDED SYSTEM | 15 |
| 3.3.1.3 POWER SUPPLY | 16 |

| | |
|--|-----------|
| 3.3.1.4 ADDITIONAL COMPONENTS | 16 |
| 3.3.2 SOFTWARE REQUIREMENTS | 17 |
| 3.3.2.1 GOOGLE COLAB | 17 |
| 3.3.2.2 ARDUINO IDE | 17 |
| 3.3.3 LIBRARIES..... | 18 |
| 3.3.3.1 PYSTREAMLIT | 18 |
| 3.3.3.2 TENSORFLOW | 18 |
| 3.3.3.3 HASH FUNCTIONS | 18 |
| 4. SOFTWARE REQUIREMENT SPECIFICATION | 19 |
| 4.1. FUNCTIONAL REQUIREMENTS | 20 |
| 4.1.1 DATA ACQUISITION | 20 |
| 4.1.2 SIGNAL PROCESSING | 20 |
| 4.1.3 TRANSFORMATIVE ENCODING AND DECODING | 20 |
| 4.1.4 VERIFICATION | 20 |
| 4.1.5 EMBEDDED INFORMATION EXTRACTION..... | 20 |
| 4.1.6 MISSING DATA DETECTION | 21 |
| 4.1.7 DATA RECONSTRUCTION (IF NEEDED) | 21 |
| 4.1.8 ERROR HANDLING..... | 21 |
| 4.2. SYSTEM ENHANCEMENTS | 21 |
| 4.2.1 STEGANOGRAPHY ENHANCEMENT..... | 21 |
| 4.2.1.1 HERMITE DOMAIN TECHNIQUE | 21 |
| 4.2.1.2 HIPPOPOTAMUS OPTIMIZATION (HO)..... | 21 |
| 4.2.2 SIGNAL RECONSTRUCTION IMPROVEMENT | 21 |
| 4.2.3 ARRHYTHMIA DETECTION INTEGRATION..... | 21 |
| 4.3 NON-FUNCTIONAL REQUIREMENTS | 22 |
| 4.3.1 SECURITY..... | 22 |
| 4.3.2 PERFORMANCE..... | 22 |

| | |
|---|-----------|
| 4.3.3 RELIABILITY | 22 |
| 4.3.4 PRIVACY AND COMPLIANCE | 22 |
| 4.3.5 SCALABILITY | 22 |
| 4.3.6 USABILITY AND ACCESSIBILITY | 22 |
| 4.3.7 RESOURCE UTILIZATION | 23 |
| 4.4 UML DIAGRAMS | 23 |
| 4.3.1 ER DIAGRAM | 23 |
| 4.3.2 ACTIVITY DIAGRAM | 24 |
| 4.3.3 USE CASE DIAGRAM | 26 |
| 5. SYSTEM DESIGN..... | 27 |
| 5.1 OVERVIEW | 28 |
| 5.2 OVERALL CONCEPT..... | 29 |
| 5.3 LSTM..... | 29 |
| 5.4 ALGORITHMS | 30 |
| 5.4.1 SECRET BIT INSERTION IN HERMITE SPACE | 30 |
| 5.4.2 SECRET BIT EXTRACTION IN HERMITE SPACE | 31 |
| 5.4.3 FORWARD HERMITE FUNCTION | 33 |
| 5.4.4 REVERSE HERMITE FUNCTION..... | 34 |
| 5.4.5 HIPPOPOTAMUS OPTIMIZATION (HO)..... | 35 |
| 6. IMPLEMENTATION METHODOLOGY | 37 |
| 6.1 OVERVIEW | 38 |
| 6.2 ESSENTIAL LIBRARIES..... | 38 |
| 6.3 FUNCTIONS USED FOR IMPLEMENTATION | 39 |
| 7. PERFORMANCE METRICS | 41 |
| 7.1 IMPERCEPTIBILITY COMPARISON | 42 |
| 7.1.1 ORIGINAL DATA, STEGO DATA AND RECONSTRUCTED DATA..... | 42 |
| 7.1.2 ORIGINAL DATA, RECEIVED DATA AND RECONSTRUCTED DATA | 43 |

| | |
|--|------------|
| 7.2 BIOSIGNAL VISUALIZATION | 44 |
| 7.2.1 ORIGINAL BIOSIGNAL | 44 |
| 7.2.2 RECONSTRUCTED BIOSIGNAL..... | 44 |
| 7.3 HERMITE FUNCTION ORDER VISUALIZATION | 45 |
| 7.3.1 FORWARD HERMITE FUNCTION | 45 |
| 7.3.2 REVERSE HERMITE FUNCTION..... | 45 |
| 8. RESULTS AND DISCUSSION | 46 |
| 8.1 OVERVIEW | 47 |
| 8.2 DATASETS | 47 |
| 8.3 RESULTS | 48 |
| 8.4 HARDWARE SETUP | 51 |
| 8.5 ARRHYTHMIA DETECTION | 51 |
| 9. CONCLUSION AND FUTURE WORK..... | 53 |
| 9.1 CONCLUSION..... | 54 |
| 9.2 FUTURE WORK..... | 54 |
| 10. APPENDIX..... | 56 |
| 10.1 CODING | 57 |
| 11. REFERENCES | 108 |
| 11.1 REFERENCES | 109 |
| 12. ANNEXURE..... | 113 |
| 12.1 JOURNAL SUBMISSION PROOF | 114 |
| 12.2 REPORT PLAGIARISM PROOF | 115 |

LIST OF TABLES

LIST OF TABLES

| TABLE NO. | TABLE NAME | PAGE NO. |
|-----------|--|----------|
| 8.2.1 | ECG, EEG, PPG, Arrhythmia Dataset | 47 |
| 8.3.1 | PRD, RMSE, PSNR values for ECG Signal | 48 |
| 8.3.2 | PRD, RMSE, PSNR values for EEG Signal | 48 |
| 8.3.3 | PRD, RMSE, PSNR values for PPG Signal | 49 |
| 8.3.4 | Various model accuracy of Arrhythmia detection | 50 |

LIST OF FIGURES

LIST OF FIGURES

| FIGURE NO. | FIGURE CAPTION | PAGE NO. |
|------------|---|----------|
| 5.2.1 | Block diagram of proposed steganography algorithm | 29 |
| 5.3.1 | Su-LSTM for Estimating the missed data block | 30 |
| 7.1.1 | Comparision diagram between Original Data, StegoData and Reconstructed Data with bias | 42 |
| 7.1.2 | Comparision diagram between Original Data, Received Data and Reconstructed Data | 43 |
| 7.2.1 | Original ECG BioSignal Data Taken from the ECG sensor | 44 |
| 7.2.2 | Reconstructed ECG BioSignal after the decryption process. | 44 |
| 7.3.1 | Forward Hemite function. | 45 |
| 7.3.2 | Reverse Hemite function. | 45 |
| 8.4.1 | Hardware Connection | 51 |
| 8.5.1 | Hardware to predict the Arrhythmia - Normal | 51 |
| 8.5.2 | Hardware to predict the Arrhythmia - Arrhythmia | 52 |

LIST OF SYMBOLS

LIST OF SYMBOLS

| NOTATION | MEANING |
|--------------|--------------------------------|
| $\Phi[B'_k]$ | Block Feature |
| BT_k | Secret Bit |
| X_m | Password |
| $\Phi[B'_m]$ | Block Feature |
| BT_m | Secret Bit |
| P | Original signal |
| Q | Erroneous signal |
| L | Length of signal P and Q |
| P_i, Q_i | i^{th} data point in P and Q |
| P_m | Mean of signal P |
| Q_m | Mean of signal Q |
| P_{max} | Maximum value of signal |

LIST OF ABBREVIATION

LIST OF ABBREVIATION

| S.NO | ACRONYMS | ABBREVIATIONS |
|------|----------|---------------------------------------|
| 1 | ECG | Electro Cardio Graph |
| 2 | EEG | Electro Encephalo Graph |
| 3 | PPG | Photo Plethysmo Graph |
| 4 | PSO | Particle Swarm Optimization |
| 5 | HOA | Hippopotamus Optimization Algorithm |
| 6 | LSTM | Long Short-Term Memory |
| 7 | MLPNN | Multi-Layer Perceptron Neural Network |

INTRODUCTION

CHAPTER 1

INTRODUCTION

In today's digital age, where concerns regarding data security and privacy are at the forefront, the integration of advanced encryption techniques with biological signal processing represents a crucial frontier. With the increasing reliance on telemedicine and remote patient monitoring, ensuring the secure and reliable transmission of biological signals has become imperative. Among these signals, the electrical activity of the heart, known as bio signals, holds particular significance in diagnosing and monitoring various cardiac conditions.

However, the transmission of bio signals over communication networks raises significant security concerns due to the potential for interception and unauthorized access. Moreover, the occurrence of data loss during transmission poses a serious challenge, potentially impeding accurate diagnosis and patient care.

This project introduces a pioneering framework titled "Deep Learning-based Biological Signal Steganography with Missing Data Recovery Approach" aimed at bolstering the confidentiality and integrity of bio signal transmissions. The central idea revolves around securing the bio signal while incorporating mechanisms for reconstructing missing data caused by transmission errors.

The proposed approach employs deep learning techniques and leverages Hermite functions, a set of orthogonal mathematical functions, for robust encryption. The bio signal undergoes a forward Hermite transform, followed by the embedding of secret data bits for secure transmission alongside the bio signal. Subsequently, a reverse Hermite transform is applied, resulting in an encrypted bio signal ready for transmission.

To ensure secure transmission, a password is generated using a cryptographic hash function, taking into account the encrypted bio signal. This password serves as a means of verifying data integrity, as any alteration in the transmitted data would lead to a different password generation upon reception, thereby alerting the receiver to potential tampering.

Upon reception, the decryption process unfolds, revealing the concealed information within the received bio signal. Through the utilization of Hermitian functions in reverse and password verification protocols, decryption not only unveils the concealed data but also identifies any missing data blocks.

Given the inherent unreliability of real-world transmission channels, the project integrates a pre-trained Multilayer Perceptron Neural Network (MLPNN) to address missing data recovery. Trained on a large dataset of bio signals, the MLPNN predicts missing data based on intact portions of the signal, ensuring the accuracy and usability of the reconstructed bio signal for medical analysis.

The efficacy and robustness of the proposed framework are underscored through experimental validation, demonstrating high levels of confidentiality for encrypted bio signals and paramount decryption accuracy. Furthermore, the integration of MLPNN for missing data recovery showcases remarkable success rates, affirming the adaptability and resilience of the approach in real-world scenarios.

In summary, this deep learning-based approach offers several advantages for the secure and reliable transmission of biological signals, including enhanced security through Hermite function encryption, data integrity ensured by password generation, and effective missing data recovery facilitated by MLPNN.

LITERATURE STUDY

CHAPTER 2

LITERATURE STUDY

2.1 ON-DEVICE RELIABILITY ASSESSMENT AND PREDICTION OF MISSING PHOTOPLETHYSMOGRAPHIC DATA USING DEEP NEURAL NETWORKS

Photoplethysmographic (PPG) measurements play a crucial role in monitoring cardiovascular health and are commonly employed in wearable health monitoring devices. However, these measurements are often susceptible to inaccuracies and missing data segments due to various factors such as body movements and sensor instability. In this paper, M. S. Roy and Et al. propose a comprehensive approach for on-device reliability assessment of PPG measurements using deep learning models.

Their methodology involves the utilization of advanced deep learning architectures to address both data quality assessment and missing segment prediction. They employ a stack denoising autoencoder (SDAE) along with a multilayer perceptron neural network (MLPNN) to evaluate the reliability of PPG beats directly on the device. This approach allows for real-time assessment of data quality, enabling timely intervention when necessary.

To tackle the issue of missing data segments, they introduce a personalized convolutional neural network (CNN) and long short-term memory (LSTM) model. By leveraging a short history of the same channel data, this model accurately predicts missing segments, thereby ensuring continuity and integrity of the PPG signal.

To validate the effectiveness of their proposed approach, they conduct experiments using forty sets of volunteer data, comprising an equal distribution of healthy subjects and individuals with cardiovascular conditions. Ground truth annotations provided by domain experts are used for validation and testing purposes, enabling rigorous evaluation of their models' performance.

The results demonstrate the efficacy of the proposed PPG reliability assessment model (PRAM), achieving a remarkable accuracy of over 95% in correctly identifying acceptable PPG beats out of a total of 5000 beats. Moreover, the disagreement with expert annotations is minimal, underscoring the reliability of their approach.

For missing segment prediction, their missing segment prediction model (MSPM) exhibits strong performance, with a root mean square error (RMSE) of 0.22 and a mean absolute error (MAE) of 0.11 for predicting 40 missing beats using only a four-beat history from the same channel PPG.

Furthermore, they implement their models on a standalone device based on a quad-core ARM Cortex-A53 processor operating at 1.2 GHz, with 1 GB RAM. The memory requirement for the device is modest, and the latency per beat prediction is approximately 0.35 seconds within a 30-second frame.

Finally, they compared the performance of their method with existing approaches using two publicly available datasets, CinC and MIMIC-II under PhysioNet. Their results indicate significant improvements in both PPG quality assessment and missing data prediction, highlighting the effectiveness and potential of their proposed approach.

Their method offers a promising solution for on-device reliability assessment of PPG measurements, addressing challenges related to data quality and missing segments. This research has implications for the development of more robust and reliable wearable health monitoring systems, enhancing their utility in clinical and everyday settings.

2.2 SVD-BASED ROBUST IMAGE STEGANOGRAPHIC SCHEME USING RIWT AND DCT FOR SECURE TRANSMISSION OF MEDICAL IMAGES

The proliferation of computer technologies and the widespread use of the Internet have significantly transformed data communication, granting easy access to vast amounts of information. However, this accessibility has also raised concerns about the unauthorized access and piracy of copyrighted and confidential medical imagery. Despite numerous methodologies proposed in the literature, existing techniques often lack robustness, imperceptibility, and resilience against attacks. In response to these challenges, this study presents a novel and comprehensive approach to medical image steganography.

S. Arunkumar and Etal. proposed a technique combines the Redundant Integer Wavelet Transform (RIWT), Discrete Wavelet Transforms (DCT), Singular Value Decomposition (SVD), and chaotic encryption using the logistic map to achieve enhanced robustness and security in embedding medical images. RIWT, known for its shift invariance properties, ensures reversibility and robustness in the steganographic process. They enhance imperceptibility by utilizing a combination of SVD and DCT, where embedding is performed on singular values.

Furthermore, they introduce chaotic encryption using the logistic map to provide an additional layer of security for the embedded medical images, thereby enhancing resistance against unauthorized access and attacks.

To assess the efficacy of our proposed approach, they conduct a comprehensive comparative analysis with existing steganographic methods in the literature. Evaluation metrics include imperceptibility, robustness, and resistance to various geometric transformation attacks. Their method surpasses alternative techniques, demonstrating superior performance in preserving the integrity and confidentiality of medical imagery.

Validation of their approach is performed using the UCID benchmarking database, providing empirical evidence of its effectiveness and reliability in real-world scenarios.

Their hybrid approach to medical image steganography offers a promising solution to the challenges posed by unauthorized access and piracy of sensitive medical data. By integrating RIWT, DCT, SVD, and chaotic encryption, they achieve a harmonious balance between robustness, imperceptibility, and security, making their technique well-suited for safeguarding confidential medical information during digital transmission and storage.

2.3 SECURING DATA IN INTERNET OF THINGS (IOT) USING CRYPTOGRAPHY AND STEGANOGRAPHY TECHNIQUES

The rapid expansion of the Internet of Things (IoT) has led to an unprecedented volume of data being transmitted every second, presenting significant challenges in ensuring the security and privacy of this data. Cryptographic and steganographic techniques serve as crucial tools for safeguarding IoT data, particularly concerning user authentication and data privacy. In this paper, M. Khari and Et al. propose a novel approach that combines elliptic Galois cryptography and Matrix XOR encoding steganography to enhance the security of confidential medical data obtained from diverse sources within the IoT ecosystem.

Their proposed protocol begins by employing elliptic Galois cryptography to encrypt the sensitive medical data, leveraging the robustness and efficiency of this cryptographic scheme. Subsequently, the encrypted data is embedded into a low-complexity image using Matrix XOR encoding steganography, which ensures the concealment of the data within the image while preserving its integrity.

To further optimize the security and efficiency of the embedding process, they introduce an Adaptive Firefly optimization algorithm. This algorithm dynamically adjusts the selection of cover blocks within the image, enhancing the concealment of the encrypted data and fortifying its resistance against potential attacks.

To evaluate the performance of their proposed approach, they conduct a comprehensive analysis comparing various parameters, including data hiding capacity, imperceptibility, and resistance to attacks, with existing techniques in the literature. Our results demonstrate the efficacy and superiority of the proposed methodology in ensuring the security and privacy of medical data in IoT environments.

Finally, the hidden data is extracted from the image and decrypted using the corresponding cryptographic keys, thus completing the secure transmission and retrieval process.

They proposed a methodology that offers a robust and efficient solution for addressing the security challenges inherent in IoT data transmission. By integrating elliptic Galois cryptography, Matrix XOR encoding steganography, and adaptive optimization techniques, they provide a comprehensive framework for safeguarding confidential medical information in IoT environments, thereby ensuring the integrity and privacy of sensitive data.

SYSTEM STUDY

CHAPTER 3

SYSTEM STUDY

3.1 SCOPE

The scope of the project, "Deep Learning-based Biological Signal Steganography with Missing Data Recovery Approach," encompasses the development of sophisticated techniques tailored for enhancing data security within healthcare contexts, particularly concerning the transmission and storage of sensitive biological signals. This involves the exploration and implementation of advanced steganography methods, leveraging deep learning methodologies to seamlessly embed and extract information within bio signals. An innovative aspect lies in the development of strategies for missing data recovery, ensuring signal integrity even in the face of data loss or corruption. Algorithm development plays a pivotal role, requiring the design, implementation, and optimization of encryption, decryption, and missing data prediction algorithms drawing from signal processing, cryptography, and machine learning. Experimental validation is crucial, involving the assessment of encryption accuracy, decryption success rates, and missing data recovery capabilities using real or simulated biosignal datasets. The project aims to produce a practical solution deployable in real-world healthcare environments, contributing to secure communication protocols for sensitive biological data transmission. Furthermore, it lays the groundwork for future research, including algorithm optimization, integration with existing healthcare systems, scalability considerations, and extension to other biological signal types, fostering interdisciplinary collaboration and innovation. This project addresses the need for secure and reliable transmission of biological signals, in telemedicine and remote patient monitoring.

3.2 PRODUCT FUNCTION

This project encompasses a comprehensive suite of functions, including data acquisition, preprocessing, encryption, decryption, missing data recovery, integration, security, scalability, and performance optimization. It ensures the secure transmission of bio signals, prioritizes data integrity, and seamlessly integrates with existing healthcare systems, all while optimizing performance and scalability for evolving healthcare demands.

3.2.1 DATA ACQUISITION

The data acquisition phase involves the collection of raw bio signals from various sources such as medical sensors or monitoring devices. To ensure the integrity and reliability of the acquired data, preprocessing techniques are applied, including noise removal and baseline correction. These preprocessing steps are crucial for enhancing signal quality and accuracy, thereby facilitating more effective analysis and interpretation of the ECG data. Additionally, analog ECG signals are digitized to enable further processing and analysis in subsequent stages of the project.

3.2.2 PREPROCESSING

Preprocessing is a fundamental step in preparing raw bio signals for encryption and subsequent analysis. This phase involves the application of various signal processing techniques aimed at improving signal quality and removing artifacts. Techniques such as bandpass or notch filtering are employed to eliminate unwanted noise and interference, while baseline correction methods are utilized to correct for baseline wander and ensure signal consistency. Moreover, segmentation of bio signals into individual beats or segments is performed to facilitate more granular analysis and processing in later stages.

3.2.3 ENCRYPTION FUNCTIONALITY

The encryption functionality of the project encompasses the embedding of secret information into preprocessed bio signals using forward and reverse Hermitian function. This process involves the insertion of secret bits into the bio signal without compromising its clinical integrity. Advanced encryption techniques are employed to ensure the confidentiality and security of the embedded information, thereby safeguarding sensitive biological data during transmission and storage.

3.2.4 DECRYPTION FUNCTIONALITY

The decryption functionality involves the extraction of embedded secret information from received bio signals using forward and reverse Hermitian function. This process aims to restore the original bio signal while retrieving the concealed data embedded during the encryption phase.

Additionally, verification of received data integrity is performed through password matching and hash function-based authentication, ensuring the authenticity and reliability of the decrypted data.

3.2.5 MISSING DATA RECOVERY FUNCTIONALITY

The missing data recovery functionality is a critical aspect of the project, addressing the challenge of data loss or corruption during transmission. This phase involves the prediction and reconstruction of missing data blocks within received bio signals. Utilizing a Multilayer Perceptron Neural Network (MLPNN) trained on a dataset, accurate missing data recovery is achieved. Furthermore, the integration of the Hippopotamus Optimization method optimizes the block position to be predicted by the MLPNN, enhancing prediction accuracy and efficiency. Overall, this functionality ensures the completeness and integrity of transmitted bio signals, even in the presence of data loss or corruption.

3.2.6 INTEGRATION AND COMPATIBILITY

Integration and compatibility aspects are crucial for ensuring seamless interoperability with existing healthcare systems and protocols. The developed framework is designed to integrate effortlessly with standard healthcare infrastructure, facilitating smooth data transmission and storage processes. Compatibility with established bio signal formats and protocols ensures widespread adoption and interoperability across diverse healthcare environments. By adhering to industry standards and protocols, the framework enhances usability and minimizes integration challenges, thereby facilitating its deployment in real-world healthcare settings.

3.2.7 SECURITY AND RELIABILITY

Security and reliability are paramount considerations in the transmission and storage of sensitive biological data. The project prioritizes the implementation of robust encryption and decryption mechanisms to safeguard data integrity and confidentiality. Advanced encryption techniques, coupled with secure password generation and hash function-based authentication, ensure protection against unauthorized access and data breaches. Additionally, stringent verification protocols verify the authenticity and reliability of decrypted data, bolstering trust and confidence in the system's security measures.

By prioritizing security and reliability, the project instills confidence in healthcare practitioners and patients regarding the privacy and integrity of their medical data.

Scalability and Performance: Scalability and performance optimization are critical for accommodating the growing volume of biosignal data transmission and processing demands in healthcare environments. The framework is designed to scale seamlessly to handle large datasets and accommodate increasing user loads without compromising performance or efficiency. Optimization of algorithms and computational resources ensures optimal system performance, minimizing latency and maximizing throughput. Comprehensive evaluation of system performance metrics, including encryption/decryption speed, missing data recovery accuracy, and resource utilization, ensures that the framework meets stringent performance requirements. By prioritizing scalability and performance optimization, the project lays the foundation for robust and efficient data transmission and processing in healthcare settings, catering to evolving needs and demands.

3.3 SYSTEM REQUIREMENTS

The system requirements for this project are as follows:

3.3.1 HARDWARE REQUIREMENTS

The hardware requirements for this project can be divided into two categories: data acquisition and system implementation. To collect ECG signals, a medical-grade ECG sensor is necessary. This could be a set of 3-lead electrodes attached to specific body locations or a more comfortable wearable patch. The key considerations here are data quality for accurate encryption and analysis, patient comfort for extended monitoring, and data security compliance with medical privacy regulations.

Essential components of the project include biological signal sensors capable of capturing physiological data such as Electrocardiogram (ECG) and Photoplethysmography (PPG) signals. For ECG sensing, the AD8232 integrated signal conditioning chip is utilized. This chip provides a convenient and reliable solution for acquiring ECG signals, offering built-in amplification, filtering, and electrode impedance measurement functionalities.

The AD8232 simplifies the hardware design process and ensures accurate and high-fidelity ECG signal acquisition, making it well-suited for healthcare applications. Additionally, for PPG sensing, a pulse sensor module is employed. This sensor utilizes

photoplethysmography principles to detect changes in blood volume in peripheral blood vessels, providing a measure of pulse rate and blood oxygen saturation (SpO₂).

The pulse sensor module offers a non-invasive and convenient means of monitoring cardiovascular parameters, making it suitable for applications requiring continuous physiological monitoring. These sensors play a pivotal role in data acquisition, providing the raw input necessary for subsequent processing and analysis in the project's workflow. Depending on the application requirements, these sensors may be integrated into wearable devices or standalone modules, ensuring convenience and ease of use in diverse healthcare settings.

For developing and testing the deep learning algorithms, a powerful computer is needed. A multi-core processor, at least 16GB of RAM, and ample storage space (preferably an SSD) are recommended to handle complex models and large datasets efficiently. While an Arduino or Raspberry Pi could be used for interfacing with the sensor (if applicable) in the future, the core functionalities likely run on the computer due to computational limitations. It's important to remember that this project focuses on the software aspects, and real-world deployment would involve integrating the solution with existing medical-grade ECG devices or telemedicine platforms, which is beyond this project's scope.

3.3.1.1 ARDUINO BOARD OR MICROCONTROLLER

The project requires the use of an Arduino board or a similar microcontroller platform to serve as the central processing unit for data acquisition, preprocessing, and interfacing with sensors. These boards offer a versatile platform equipped with analog and digital input/output pins, facilitating seamless integration with various sensor modules. Additionally, Arduino boards provide a user-friendly development environment, enabling rapid prototyping and iterative development of the project's firmware and algorithms. Their compact size and low power consumption make them suitable for deployment in portable or wearable applications, ensuring flexibility and versatility in system design.

3.3.1.2 COMPUTER OR EMBEDDED SYSTEM

The project necessitates the use of a computer or embedded system for algorithm development, data processing, and system testing. This computing platform serves as the

backbone of the project, providing the necessary computational resources to execute encryption/decryption algorithms, neural network models, and other processing tasks efficiently

. Depending on the project's requirements and deployment scenario, a range of computing platforms may be suitable, including desktop computers, laptops, or embedded systems such as Arduino UNO. Sufficient processing power, memory, and storage capacity are essential to ensure smooth operation and optimal performance of the system.

3.3.1.3 POWER SUPPLY

The project utilizes the USB power supply from a computer or laptop for powering the system. By leveraging the USB port's power output, the need for an external power source such as a battery pack is eliminated, making the system more convenient and portable. The USB power supply provides a reliable source of energy to drive the system components, ensuring continuous operation without the need for frequent battery replacements. This approach simplifies the system setup and enhances its versatility, allowing for deployment in various environments without the constraints associated with traditional power sources. Additionally, voltage regulation may still be employed to maintain stable voltage levels and protect system components from potential fluctuations or surges in the USB power supply.

3.3.1.4 ADDITIONAL COMPONENTS

In addition to the core hardware components mentioned above, various additional components may be required to support the project's development and deployment. These may include breadboards and jumper wires for prototyping and connecting components during development, enclosures for housing and protecting system components, and data storage devices for logging and storing encrypted biological data. These additional components play a vital role in ensuring the functionality, reliability, and usability of the system, enhancing its overall performance and effectiveness in real-world applications.

3.3.2 SOFTWARE REQUIREMENTS

3.3.2.1 GOOGLE COLAB

To utilize Google Colab for your project, you'll need access to a modern web browser such as Google Chrome, Mozilla Firefox, or Microsoft Edge. Google Colab operates entirely within the browser, allowing you to access and run Python code in a collaborative environment. Since Google Colab notebooks are stored on Google Drive, you'll also need a Google account to save and access your notebooks. This cloud-based platform offers the advantage of providing free access to computing resources, including GPU acceleration, making it well-suited for machine learning tasks and data analysis without the need for expensive hardware or setup.

3.3.2.2 ARDUINO IDE

The Arduino Integrated Development Environment (IDE) is essential for programming Arduino boards used in your project. Available for Windows, macOS, and Linux, the Arduino IDE provides a user-friendly interface for writing, compiling, and uploading code to Arduino microcontrollers. To get started, download and install the Arduino IDE from the official Arduino website. Additionally, ensure that you have the necessary drivers installed for your specific Arduino board to enable communication between the IDE and the board. The IDE also supports the use of libraries, which are collections of pre-written code that simplify interfacing with sensors or peripherals. Install any required libraries, such as those for the AD8232 ECG sensor and pulse sensor, to facilitate seamless integration with your hardware components.

3.3.2.3 PYTHON

Python serves as the primary programming language for your project, offering a versatile and powerful toolset for data analysis, machine learning, and web development. To begin coding in Python, you'll need to install the Python interpreter on your computer. Python is compatible with Windows, macOS, and Linux operating systems, and can be downloaded from the official Python website or installed via a package manager like Anaconda. Once Python is installed, you can use pip or conda, Python's package managers, to install additional libraries and dependencies required for your project.

Python's extensive ecosystem of libraries and frameworks, combined with its simplicity and readability, makes it an ideal choice for a wide range of application

3.3.3 LIBRARIES

3.3.3.1 PYSTREAMLIT

PyStreamlit is a Python library that enables the creation of interactive web applications directly from Python scripts. Designed for simplicity and ease of use, PyStreamlit allows you to build interactive dashboards, data visualizations, and machine learning models with minimal effort. Install the library using pip, Python's package manager.

PyStreamlit seamlessly integrates with Python code and is compatible with Google Colab, enabling you to create and deploy interactive web applications directly within your Colab notebooks. With PyStreamlit, you can quickly prototype and share your projects, making it an invaluable tool for data scientists, developers, and researchers alike.

3.3.3.2 TENSORFLOW

TensorFlow is an open-source machine learning framework developed by Google that facilitates the development and deployment of machine learning models. Install the library using pip or conda. TensorFlow provides high-level APIs for building and training machine learning models, as well as low-level APIs for fine-grained control and customization. With TensorFlow, you can leverage state-of-the-art machine learning algorithms and techniques to solve a variety of tasks, including classification, regression, and clustering.

3.3.3.3 HASH FUNCTIONS

Hash functions play a crucial role in cryptography and data integrity verification, allowing you to securely hash data for storage, transmission, and authentication purposes. Python provides built-in support for hash functions through the hashlib module, which implements a variety of hashing algorithms such as SHA-256, SHA-1, and MD5. Consider factors such as security, performance, and compatibility with your specific use case when selecting a hashing algorithm. By leveraging hash functions, you can ensure the integrity and authenticity of your data, providing a critical layer of security for your project.

SOFTWARE REQUIREMENT SPECIFICATION

CHAPTER 4

SOFTWARE REQUIREMENT SPECIFICATION

4.1. FUNCTIONAL REQUIREMENTS

4.1.1 DATA ACQUISITION

The system shall support the acquisition of EEG, ECG, and PPG signals from patients using diverse bio-signal devices. The system must facilitate real-time data capture to enable continuous monitoring of bio-signals.

4.1.2 SIGNAL PROCESSING

The system should include a processing module capable of filtering and normalizing acquired bio-signal data. Signal processing must incorporate noise and artifact removal techniques to enhance the reliability of bio-signal data.

4.1.3 TRANSFORMATIVE ENCODING AND DECODING

The system shall implement a transformative encoding process to modify bio-signal data for steganographic purposes while preserving original characteristics. The system must incorporate secret bit manipulation techniques to embed and extract information within the transformed bio-signal data. The system shall implement a transformative decoding process to accurately reconstruct bio-signal data from stegodata with minimized distortion.

4.1.4 VERIFICATION

The system must integrate a secure password verification mechanism to authenticate users accessing the bio-signal data. Measures should be in place to prevent unauthorized access and ensure the security of bio-signal data.

4.1.5 EMBEDDED INFORMATION EXTRACTION

The system shall implement secret bit extraction techniques to retrieve embedded information from stegodata. Extraction should be performed accurately without introducing errors or distortions to the bio-signal data.

4.1.6 MISSING DATA DETECTION

The system must incorporate a mechanism to detect missing data or anomalies in the reconstructed bio-signal information. Error-checking methods should be employed to identify areas requiring data reconstruction.

4.1.7 DATA RECONSTRUCTION (IF NEEDED)

The system shall integrate machine learning algorithms for reconstructing missing or distorted bio-signal data. Algorithms should be trained on known bio-signal patterns to enhance accuracy in data reconstruction.

4.1.8 ERROR HANDLING

The system must include robust error-handling mechanisms to address anomalies, authentication failures, or reconstruction inaccuracies. Informative feedback should be provided to users in case of errors to enhance user understanding and system transparency.

4.2. SYSTEM ENHANCEMENTS

4.2.1 STEGANOGRAPHY ENHANCEMENT

4.2.1.1 HERMITE DOMAIN TECHNIQUE

Enhance the steganography technique by implementing the novel Hermite domain transformation described in the abstract. Describe how this technique minimizes information loss during secure data embedding.

4.2.1.2 HIPPOPOTAMUS OPTIMIZATION (HO)

Integrate the Hippopotamus Optimization approach to address potential data loss during decryption, ensuring information integrity as outlined in the abstract.

4.2.2 SIGNAL RECONSTRUCTION IMPROVEMENT

Implement the Su-LSTM network to predict and reconstruct lost signal segments, providing a partially reversible process with minimal distortion, as mentioned in the abstract.

4.2.3 ARRHYTHMIA DETECTION INTEGRATION

Describe the integration of arrhythmia detection within the steganography framework, allowing independent and multiple data embedding within a single signal. Highlight the real-time analysis capability for detecting potential heart rhythm abnormalities.

4.3 NON-FUNCTIONAL REQUIREMENTS

4.3.1 SECURITY

Ensure robust encryption techniques to protect bio-signal data from unauthorized access. Implement secure password verification mechanisms to prevent unauthorized decryption attempts.

4.3.2 PERFORMANCE

Optimize system performance for real-time data processing during encryption, transmission, and decryption. Minimize latency to provide timely and reliable communication.

4.3.3 RELIABILITY

Design error detection and correction mechanisms to ensure accurate decryption and data reconstruction. Ensure reliable password verification to prevent false positives or negatives.

4.3.4 PRIVACY AND COMPLIANCE

Adhere to medical data privacy regulations (e.g., HIPAA) to safeguard patient information. Implement encryption standards following industry best practices.

4.3.5 SCALABILITY

Design the system to handle varying data sizes and adapt to different medical contexts. Evaluate and ensure consistent system performance as data volume and user demand increase.

4.3.6 USABILITY AND ACCESSIBILITY

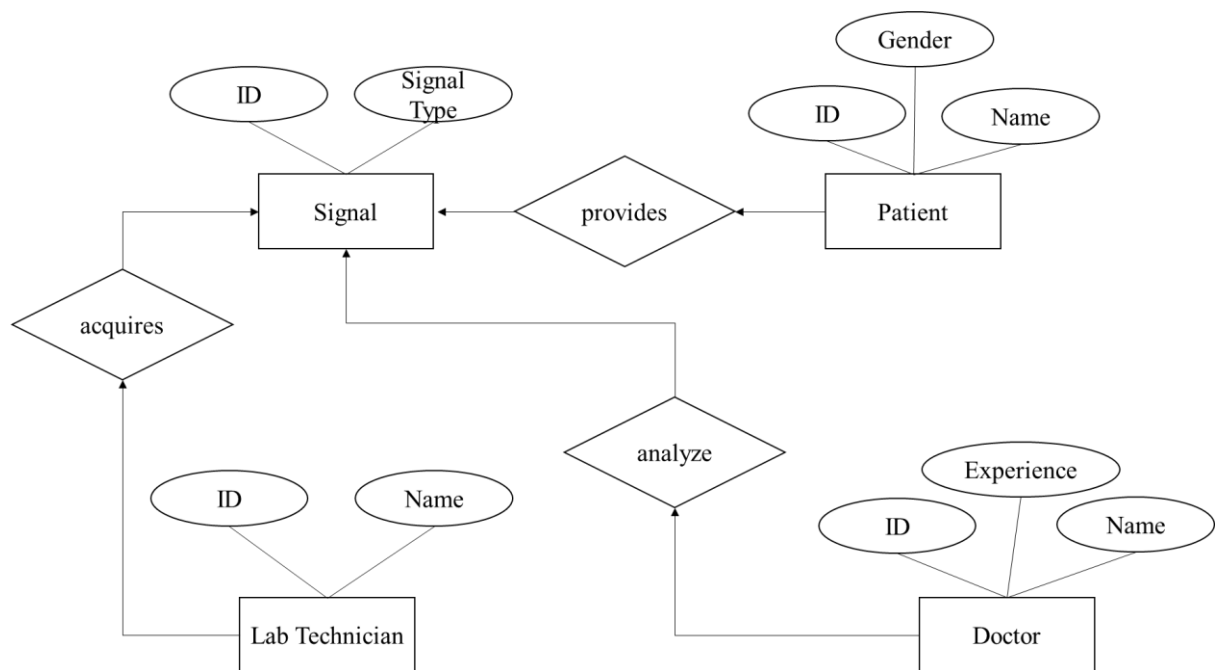
Create an intuitive user interface for data acquisition, transmission, and decryption processes. Ensure accessibility for users with varying levels of technical expertise, including healthcare professionals and system administrators.

4.3.7 RESOURCE UTILIZATION

Optimize resource utilization to minimize hardware requirements while maintaining system performance. Conduct thorough testing to ensure the system operates effectively in diverse computing environments.

4.4 UML DIAGRAMS

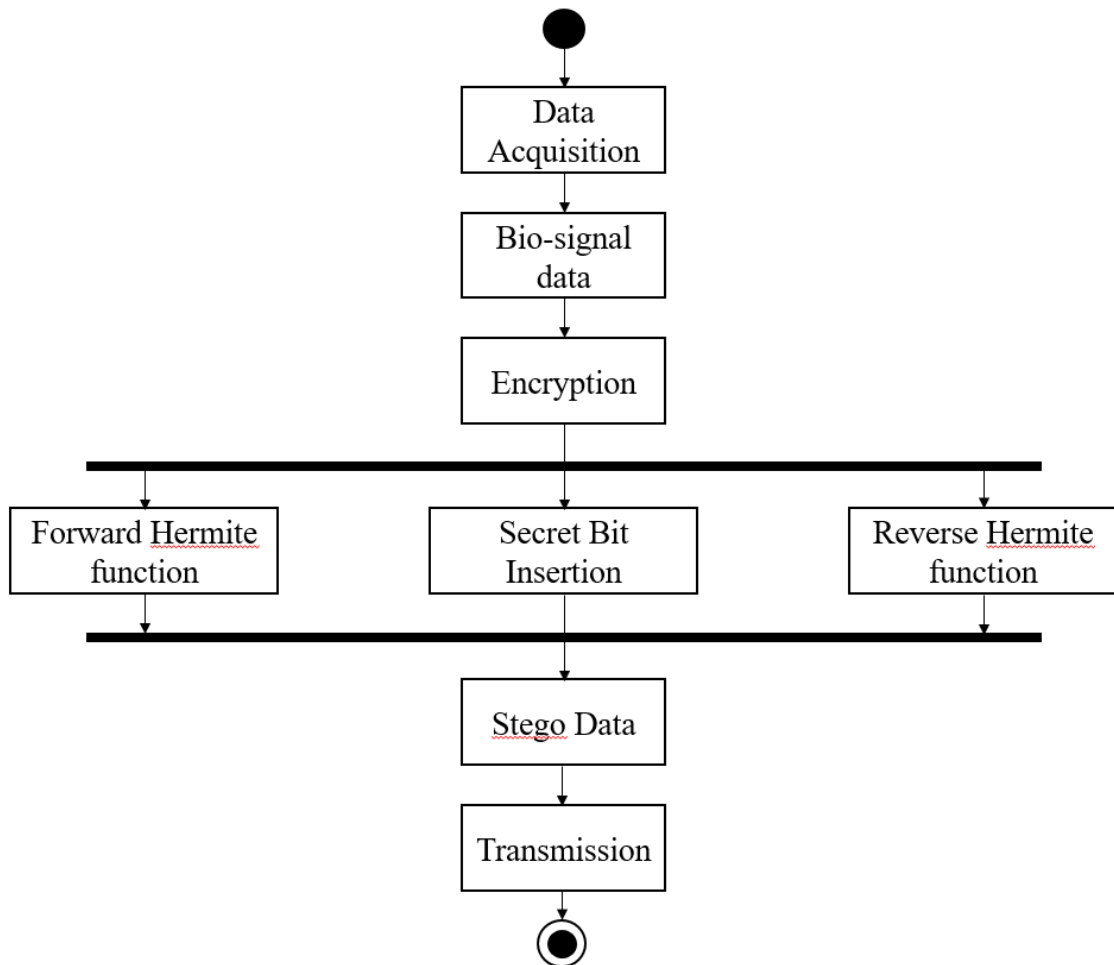
4.3.1 ER DIAGRAM



ER DIAGRAM: DEEP LEARNING – BASED BIOLOGICAL SIGNAL STEGANOGRAPHY WITH MISSING DATA RECOVERY APPROACH

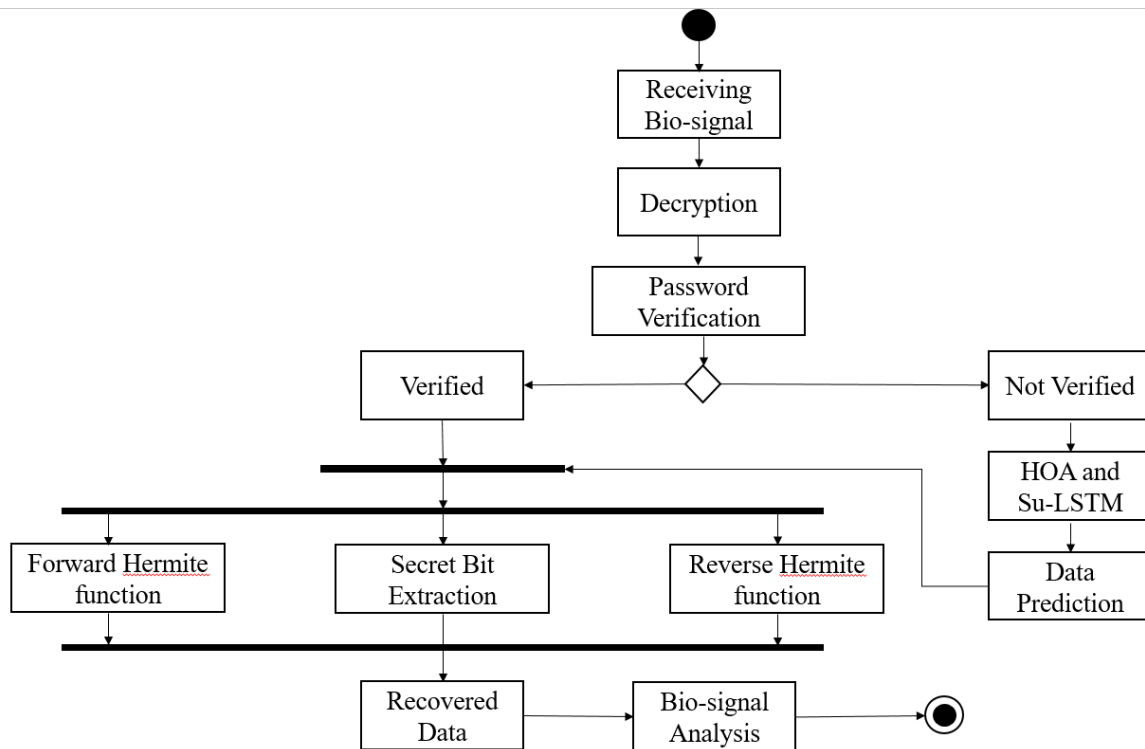
4.3.2 ACTIVITY DIAGRAM

SENDER



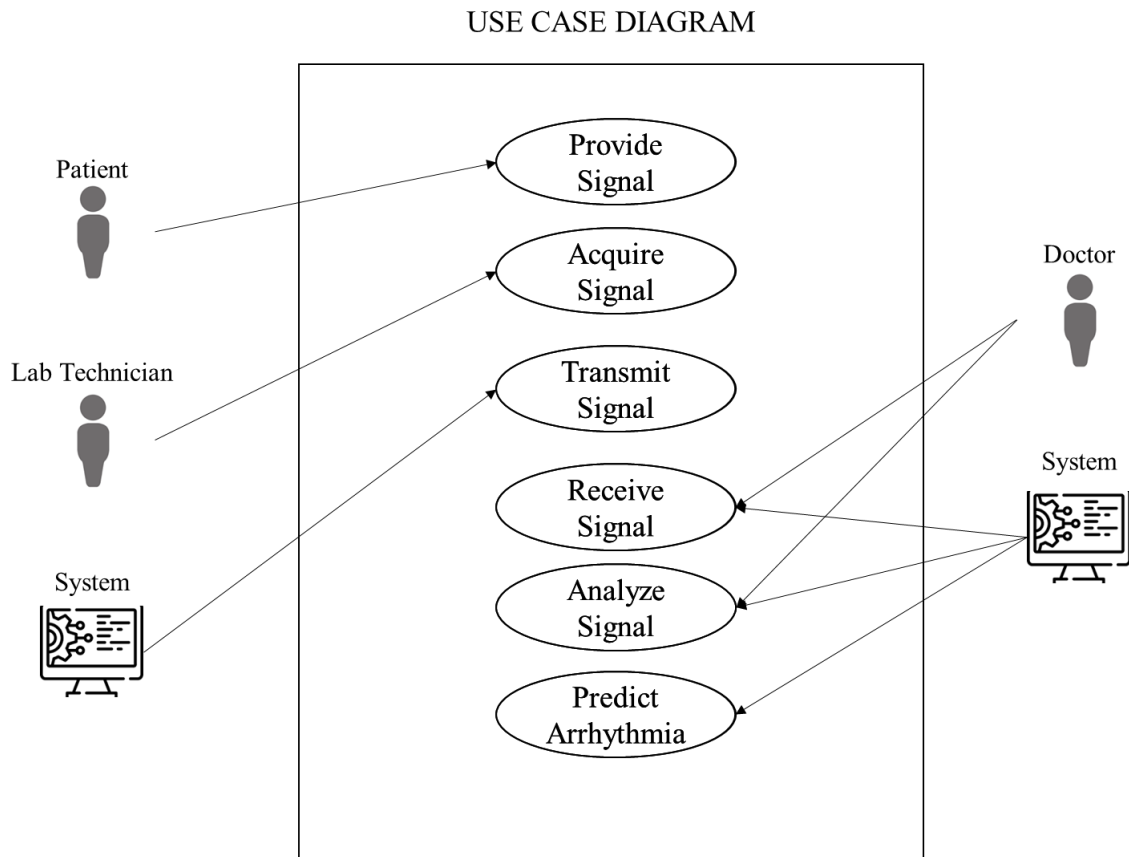
ACTIVITY DIAGRAM: DEEP LEARNING – BASED BIOLOGICAL SIGNAL STEGANOGRAPHY WITH MISSING DATA RECOVERY APPROACH – SENDER SIDE

RECEIVER



ACTIVITY DIAGRAM: DEEP LEARNING – BASED BIOLOGICAL SIGNAL
STEGANOGRAPHY WITH MISSING DATA RECOVERY APPROACH –
RECEIVER SIDE

4.3.3 USE CASE DIAGRAM



USE CASE DIAGRAM: DEEP LEARNING – BASED BIOLOGICAL SIGNAL STEGANOGRAPHY WITH MISSING DATA RECOVERY APPROACH

CHAPTER 5

SYSTEM DESIGN

5.1 OVERVIEW

The framework leverages cutting-edge deep learning techniques to bolster the confidentiality and integrity of bio-signal data transmissions. Key components of the framework include the use of Hermite functions for robust encryption and a cryptographic hash function for data integrity verification.

During the encryption process, the bio-signal undergoes a forward Hermite transform, enabling the embedding of secret data bits. This ensures that sensitive information is concealed within the signal, safeguarding it from unauthorized access during transmission. To ensure data integrity, cryptographic hash functions generate passwords based on the encrypted signal, allowing for the detection of any alterations or tampering during transmission.

Upon reception, the decryption process unveils the concealed information and identifies missing data blocks. This is achieved through the utilization of a pre-trained Multilayer Perceptron Neural Network (MLPNN), which predicts missing data based on surrounding intact portions of the signal. This ensures accurate reconstruction for subsequent medical analysis, even in the presence of unreliable transmission channels.

Experimental validation demonstrates the framework's efficacy and robustness, with high levels of confidentiality and minimal distortion to clinical information in encrypted bio-signals. Rigorous password verification mechanisms further ensure data integrity, while the MLPNN exhibits remarkable success rates in real-world scenarios.

This approach offers enhanced security through Hermite function encryption, data integrity via password generation, and missing data recovery facilitated by the MLPNN. This comprehensive solution addresses critical concerns surrounding the secure and reliable transmission of biological signals in telemedicine and remote patient monitoring applications.

5.2 OVERALL CONCEPT

The project focuses on enhancing the security and reliability of transmitting and storing biological signals, particularly Electrocardiogram (ECG) data, using a combination of deep learning techniques and steganography. By integrating deep learning-based missing data recovery approaches, the system aims to ensure the integrity and confidentiality of the transmitted signals while also providing robustness against data loss or corruption. The proposed framework is illustrated in Figure 5.2.1.

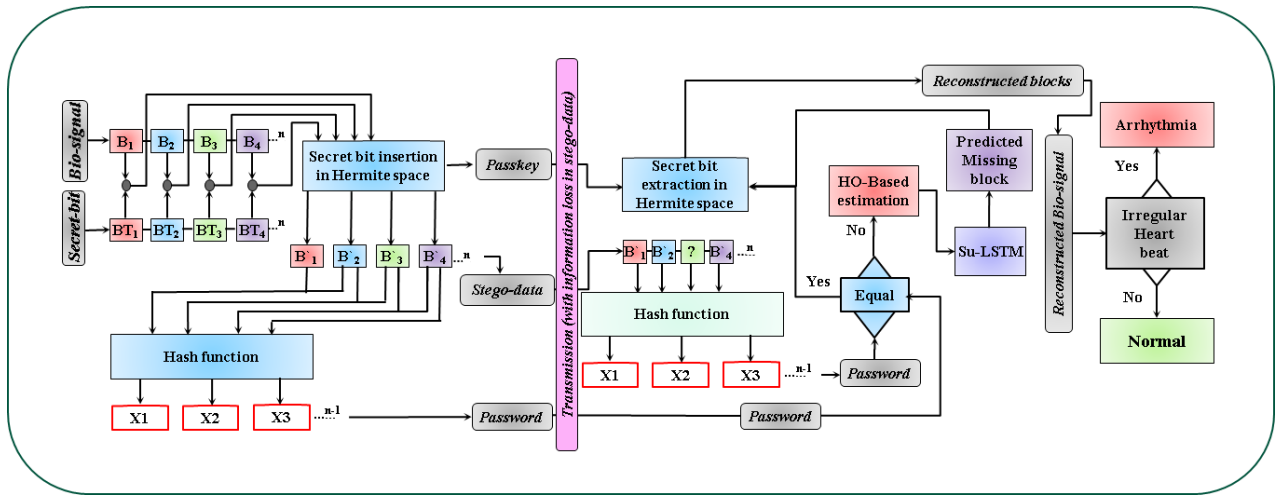


Fig.5.2.1 Block diagram of proposed steganography algorithm shown for both data insertion and extraction, mentioning data loss during transmission.

5.3 LSTM

The LSTM model, underwent training with known block parameters to ensure precise prediction of missing segment's patterns. In the realm of time-series data prediction, Recurrent Neural Networks (RNNs) serve as a prominent tool. RNNs leverage inherent memory stages to retain information about past outputs and current input, facilitating current output prediction. However, classical RNNs encounter limitations such as gradient vanishing and bursting, particularly evident in long-term series prediction tasks. To address this, Long Short-Term Memory (LSTM) networks employ an internal gate mechanism for forecasting future sequences. This process involves four fundamental gates: forget (F), input (I), update (U), and output (O), illustrated in the Figure 5.2.2.

The F-gate evaluates the alignment between the current input state (U_t) and the previous output state (V_{t-1}) using a sigmoid function $\sigma(\bullet)$, determining the extent of information retention and updating internal weights (ω_F) with an input bias value (b_F).

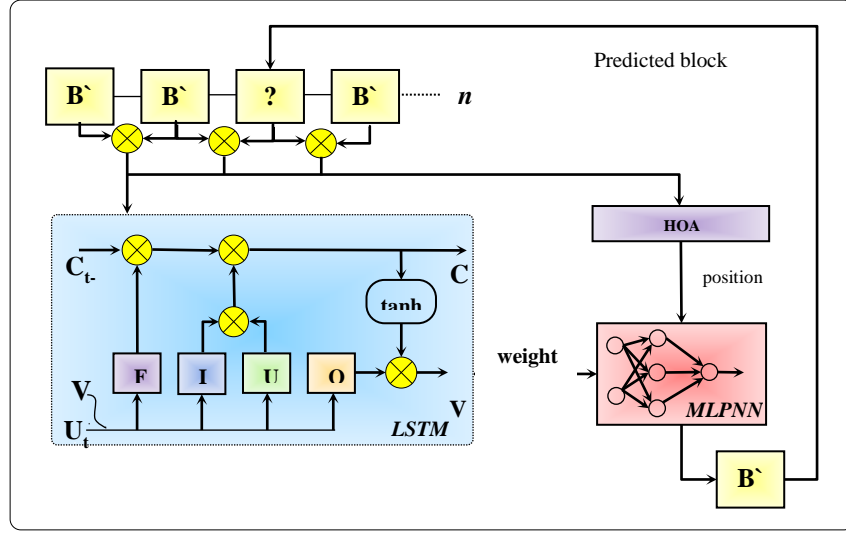


Fig.5.3.1 Su-LSTM for Estimating the missed data block

Similarly, the I-gate adjusts its weights (ω_I) with a bias value (b_I), comparing current input state values with prior outputs. The U-gate utilizes a hyperbolic tangent operator $\tanh(\bullet)$ and update bias (b_U) to update its weights (ω_U) based on input gate information, influencing the current cell state (C_t) as $C_t = F \times C_{t-1} + I \times C_t$.

Finally, the O-gate regulates the flow of data to the current output state, adjusting its weights using a sigmoid function and update bias (b_O), leading to the final output state (V_t) as $V_t = O \times \tanh(C_t)$. This architectural design of LSTM networks effectively addresses the challenges associated with long-term sequence prediction.

5.4 ALGORITHMS

5.4.1 SECRET BIT INSERTION IN HERMITE SPACE

Input:

You start with an original data block in a spatial format called the cover block.

There are two secret bits that you want to hide within this cover block.

Output:

The result of the process is a modified block called the stego-block, which contains the cover block with the embedded secret bits.

Start:

The process begins by transforming the cover block into a different mathematical representation known as Hermite space.

Transforming Coordinates:

Certain calculations are performed on the transformed coordinates of the cover block.

Secret Bit Insertion:

Depending on the values of the secret bits and certain calculations, specific adjustments are made to certain coordinates.

Coordinate Modification:

The modified coordinates are used to reconstruct the cover block with the embedded secret bits.

Inverse Transformation:

The modified block is transformed back from Hermite space to its original spatial format.

End:

The process concludes after successfully embedding the secret bits into the cover block.

Algorithm 1 – Secret Bit Insertion in Hermite Space

Input : Cover block $B_k[x_i]$ secret bit b_1 and b_2

Output : Stego-block $B'_k[\tilde{x}_i]$

Start

$B_k[y_i] = He_n(B_k[x_i])$ ## n is the order of HF

$Y1 = \{\text{roundoff}(y_i, 3)\}$ ## up to three decimal places

$Y2 =$ up to 2 decimal places of Y1 without sign

$S_n =$ sign of Y1

$p =$ third decimal number Y1 ## $0 \leq p \leq 9$

$p1 = \text{roundoff}(p/2)$ ## $0 \leq p1 \leq 5$

$p2 = p - p1 + 1$

if $b_1 == 0$ and $b_2 == 0$

$y_{add} = 00$

$y_{fix} = p$

else

$y_{add} = p2 \times \{(b_1 \times 10) + b_2\}$

$y_{fix} = p1$

$\tilde{y}_i = S_n \times \{(Y2) + (y_{fix} \times 10^{-3}) + (y_{add} \times 10^{-5})\}$

$B'_k[\tilde{x}_i] = He_n^{-1}(B_k[\tilde{y}_i])$

End

5.4.2 SECRET BIT EXTRACTION IN HERMITE SPACE

Input:

The algorithm takes a modified block, called the stego-block, which has secret information embedded within it.

Additionally, it requires a passkey to extract the embedded secret bits.

Output:

After processing, the algorithm provides the original block with the extracted secret bits.

Start:

The process initiates by transforming the stego-block into Hermite space using a forward Hermite function.

Transforming Coordinates:

Coordinates within the Hermite space are adjusted through rounding and extraction, considering their signs.

Secret Bit Extraction:

The algorithm determines the embedded secret bits based on certain calculations involving the adjusted coordinates.

Algorithm 2 – Secret Bit Extraction in Hermite Space

Input : Stego-block $B'_k[\tilde{x}_i]$, Passkey

Output : Final block $'B_k[\hat{x}_i]$ secret bit b_1 and b_2

Start

$B'_k[\tilde{y}_i] = He_n(B'_k[\tilde{x}_i])$ ## n is the order of HF

$Z1 = \{\text{roundoff}(\tilde{y}_i, 5)\}$ ## up to five decimal places

$Z2 =$ up to 2 decimal places of Z1 without sign

$S_n =$ sign of Z1

$y_{fix} =$ third decimal number Z1 ## $0 \leq p \leq 9$

$q2 =$ fourth decimal number Z1 ## $0 \leq p \leq 9$

$q3 =$ fifth decimal number Z1 ## $0 \leq p \leq 9$

if $q2 == 0$ and $q3 == 0$

$b1 = 0, b2 = 0$

else

if remainder of $[(q2 \times 10) + q3] / 11 == 0$

$b1 = 1, b2 = 1$ and quotient is qn

else if remainder of $[(q2 \times 10) + q3] / 10 == 0$

$b1 = 1, b2 = 0$ and quotient is qn

else if remainder of $[(q2 \times 10) + q3] / 01 == 0$

$b1 = 0, b2 = 1$ and quotient is qn

$y_{fix} = y_{fix} + qn - 1$

$\hat{y}_i = S_n \times \{Z2 + (y_{fix} \times 10^{-3})\}$

$'B_k[\hat{x}_i] = He_n^{-1}(B'_k[\hat{y}_i])$

| |
|-----|
| End |
|-----|

Coordinate Modification:

Using the extracted secret bits and adjusted coordinates, the original spatial coordinates are reconstructed.

Inverse Transformation:

The inverse Hermite function is applied to restore the modified block back to its original form.

End:

The algorithm concludes upon successfully extracting the secret bits from the stego-block.

5.4.3 FORWARD HERMITE FUNCTION

Forward Hermite function

Input : Cover block s , Hermite order n

Output : Transformed block t

Start:

hrm_block = []

for **block_i** in s :

 if $n = 1$

 result = **block_i** + 2

 elif $n = 2$

 result = **block_i***2 - **block_i** + 2

 elif $n = 3$

 result = **block_i***3 - **block_i***2 + 2

 elif $n = 4$

 result = **block_i***4 - **block_i***3 + 2

 elif $n = 5$

 result = **block_i***5 - **block_i***4 + 2

 elif $n = 6$

 result = **block_i***6 - **block_i***5 + 2

 else

 raise ValueError("n, must be between 1 and 6")

 hrm_block.append(result)

return hrm_block

End

Explanation:

- The function `hermite_forward` iterates through each element in the input list `x`.
- Based on the specified order `n`, it applies the corresponding forward Hermite transformation rule to each element.
- The transformed values are then appended to the `hrm_block` list.
- Finally, the function returns the list of transformed values.

5.4.4 REVERSE HERMITE FUNCTION

Reverse Hermite function

Input : Transformed block $\tilde{\mathbf{t}}$, Hermite order `n`**Output :** Stego block $\tilde{\mathbf{s}}$

Start:

```

x = sp.Symbol('x')
eqn = None
if n = 1
    eqn = x - (y - 2)
elif n = 2
    eqn = x*2 - x - (y - 2)
elif n = 3
    eqn = x*3 - x*2 - (y - 2)
elif n = 4
    eqn = x*4 - x*3 - (y - 2)
elif n = 5
    eqn = x*5 - x*4 - (y - 2)
elif n = 6
    eqn = x*6 - x*5 - (y - 2)
else
    raise ValueError("n, must be between 1 and 6")
solutions = sp.solve(eqn, x)
 $\tilde{\mathbf{s}}_i$  = [sol.evalf() for sol in solutions if sol.is_real]
return  $\tilde{\mathbf{s}}$ 

```

End

Explanation:

- The function `reverse_hermite` constructs an equation based on the specified order `n` to perform the reverse Hermite transformation.
- It solves the equation to obtain potential solutions for the original values.
- Complex solutions, if any, are filtered out, and only real solutions are evaluated and stored.
- Finally, the function returns the list of real solutions, representing the original values before the Hermite transformation.

5.4.5 HIPPOPOTAMUS OPTIMIZATION (HO)

Algorithm 3 - Hippopotamus Optimization (HO)

Inputs :

SearchAgents: Number of search agents (hippos) in the population.

Max_iterations: Maximum number of iterations.

lowerbound: Lower bounds for the search space.

upperbound: Upper bounds for the search space.

dimension: Dimensionality of the problem.

fitness: Objective function to be optimized.

Outputs :

Best_pos: Best solution found.

Best_score: Fitness value of the best solution.

HO_curve: Convergence curve showing the best cost obtained so far over iterations.

Start**Initialization:**

Define the levy flight function `levy` to generate levy flights.

Initialize population `X` with `SearchAgents` random solutions within the search space defined by `lowerbound` and `upperbound`.and Evaluate the fitness of each solution in `X`.

Main Loop:

Repeat for each iteration `t` in `[1, Max_iterations]`:

a. Record the best solution found so far (`Xbest`) and its corresponding fitness (`fbest`).

b. Divide the population into two groups:

-Group 1: Hippos mimicking cooperative behavior.

-Group 2: Predators mimicking predatory behavior.

c. Update positions of solutions in Group 1 based on cooperative behavior:

-Each solution explores using levy flights and interacts with a dominant hippopotamus and nearby group mean.

d. Update positions of solutions in Group 2 based on predatory behavior

-Each solution imitates a predator, attempting to catch its prey using levy flights and distance-based rules.

e. Apply local exploration to all solutions in the population:

-Perturb the positions of solutions using a randomized exploration strategy.

f. Update fitness values of all solutions.

g. Record the best solution found in this iteration.

h. Print the iteration number and the best cost found so far.

Termination:

Return Best_pos, the corresponding Best_score, and the HO_curve.

End

Hippos, renowned for their adaptability and efficiency in both land and water environments, exhibit behaviors that serve as the foundation for the HO algorithm. This includes cooperative foraging, where hippos gather in groups to explore and exploit food sources collectively, leveraging shared intelligence and resources to enhance efficiency. Additionally, hippos display predatory instincts, manifesting in territorial behavior and competitive strategies to secure resources and maintain dominance within their ecosystem. The HO algorithm mirrors these behaviors, with individuals within the population representing potential solutions undergoing iterative updates guided by cooperative foraging and predatory instincts to search for optimal solutions. Initialization involves randomly initializing a population within the problem's search space and evaluating each solution's fitness based on the objective function, providing a measure of its suitability for the optimization task. Throughout the main iterative process, individuals navigate the search space through three distinct phases: the cooperative foraging phase, where individuals explore collectively guided by a dominant leader's position and neighboring group means, emphasizing collaborative exploration and knowledge sharing; the predatory instincts phase, where some individuals adopt competitive strategies to outperform others and exploit advantageous positions, fostering competition and selective pressure within the population; and local exploration, where all individuals engage in perturbing their positions to diversify the search space coverage and prevent premature convergence.

IMPLEMENTATION METHODOLOGY

CHAPTER 6

IMPLEMENTATION METHODOLOGY

6.1 OVERVIEW

The implementation methodology describes the main functional requirements, which needed for doing the project.

6.2 ESSENTIAL LIBRARIES

The library used in this project are :

❖ **NumPy:**

A library for numerical computing in Python that provides support for large, multi-dimensional arrays and matrices, and a wide range of mathematical functions.

❖ **Pandas:**

A library for data manipulation and analysis that provides support for data structures such as data frames and series, and a variety of tools for data cleaning, merging, and reshaping.

❖ **TensorFlow:**

TensorFlow is an open-source machine learning framework developed by Google. It provides a comprehensive ecosystem for building and deploying machine learning models efficiently across various platforms and devices.

❖ **Keras:**

Keras is an open-source neural network library written in Python. It is user-friendly, modular, and built on top of TensorFlow, Theano, or Microsoft Cognitive Toolkit. Ideal for fast prototyping of deep learning models.

❖ **Scikit-learn:**

A library for machine learning in Python that provides a wide range of tools for data preprocessing, model selection, and evaluation.

❖ **Matplotlib:**

Matplotlib is a Python library for creating static, animated, and interactive visualizations. It offers a wide range of plotting tools for data exploration and presentation.

❖ **Os:**

The OS library in Python provides functions for interacting with the operating system, allowing tasks such as file manipulation, directory operations, and environment variable management.

❖ **matplotlib.pyplot:**

It provides necessary tools to perform data visualization that helps us to plot the data and analyze them for better understanding.

6.3 FUNCTIONS USED FOR IMPLEMENTATION

The user defined function used for the implementation of the project are

❖ **Aggregate Models:**

This user-defined function is used to aggregate the various models into a single global model.

❖ **Reconstructed noisy data:**

This user-defined function is used to apply the variational auto encoder and laplace noise to the original data and get reconstructed noisy data as output.

❖ **Hospital Update:**

This user-defined function is used to apply the gradient and learning rate values to the given model and it has to minimize the loss function in that model.

❖ **Server Model:**

This user-defined function is used to implement federated learning. This function mainly cooperates the above three methods.

❖ **Load Model:**

The "load_model" function is a part of various libraries (e.g., TensorFlow, PyTorch) allowing users to load pre-trained machine learning models for inference tasks efficiently.

❖ **Loss Function:**

The "loss_fn" library provides functions for calculating loss in machine learning models, crucial for training algorithms by measuring the deviation between predicted and actual values.

❖ **Data Visualization Functions:**

Functions to visualize the preprocessed ECG data, the extracted features, and the classification results using various visualization techniques such as line plots, scatter plots, and heat maps.

❖ **Apply_gradients:**

The ``apply_gradients`` function library is a tool for adjusting model parameters in machine learning frameworks, such as TensorFlow, by applying gradients computed during training.

❖ **Train-Test Splitting:**

It basically splits data in to train and test where training data goes for 75% and test data for 25% which is further used to predict and classify the data.

❖ **Accuracy score:**

Accuracy can also be defined as the ratio of the number of correctly classified cases to the total of cases under evaluation. The best value of accuracy is 1 and the worst value is 0.

❖ **Visualization:**

Data visualization is a technique to analyze and visualize data in a well-organized form which makes it easier to understand, observe, analyze.

❖ **Reshape:**

In machine learning, reshape is a function used to change the shape of a tensor, without changing the underlying data. It is commonly used in deep learning frameworks like TensorFlow and Keras to prepare input data for training machine learning models. The reshape function takes a tensor as input and returns a new tensor with the same data but a different shape. The new shape is specified as a tuple of integers, where each integer represents the size of a particular dimension.

❖ **Dropout:**

In machine learning, Dropout is a technique used to prevent neural network overfitting. It works by setting a random fraction of the input units to 0 during each training epoch. It has the effect of forcing the network to learn more robust features, as it cannot rely on any single input unit.

PERFORMANCE METRICS

CHAPTER 7

PERFORMANCE METRICS

7.1 IMPERCEPTIBILITY COMPARISON

7.1.1 ORIGINAL DATA, STEGO DATA AND RECONSTRUCTED DATA

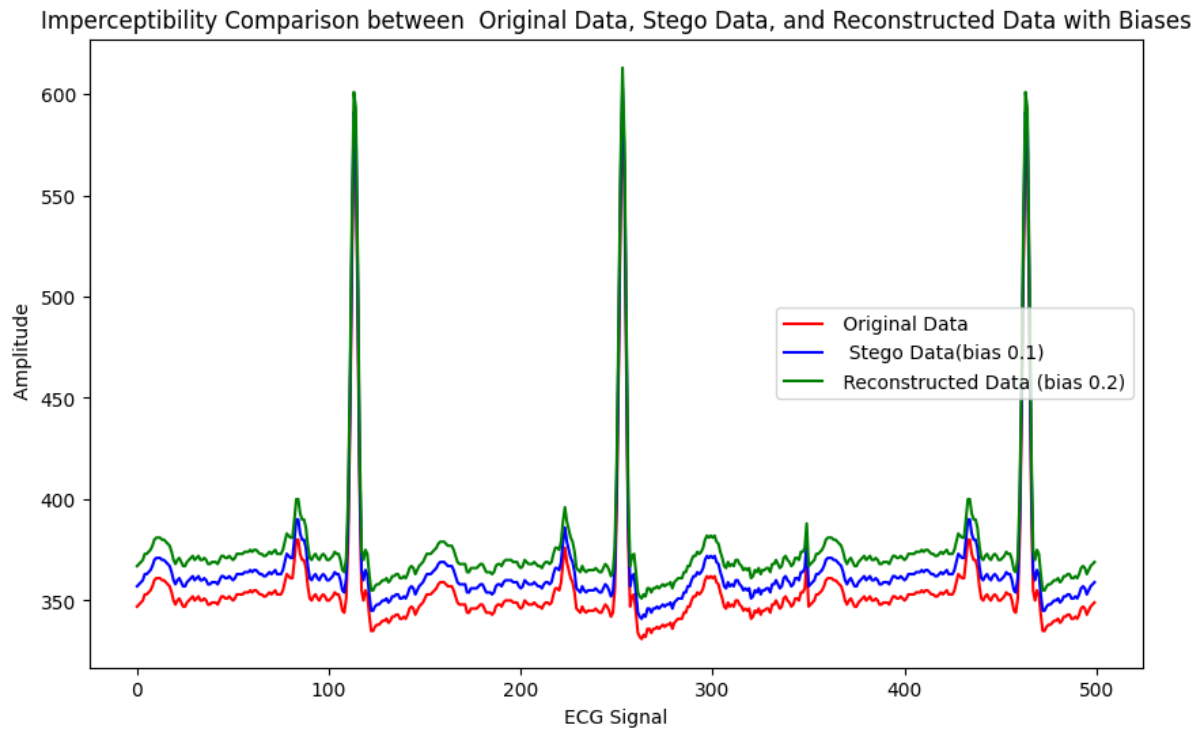


Fig.7.1.1 Imperceptibility Comparison between Original Data, StegoData and Reconstructed Data with bias

Figure 7.1.1 shows the original data in blue, the stego-data in green, and the reconstructed data in red. The fact that the reconstructed data closely resembles the original data suggests that the steganography technique is successful.

7.1.2 ORIGINAL DATA, RECEIVED DATA AND RECONSTRUCTED DATA

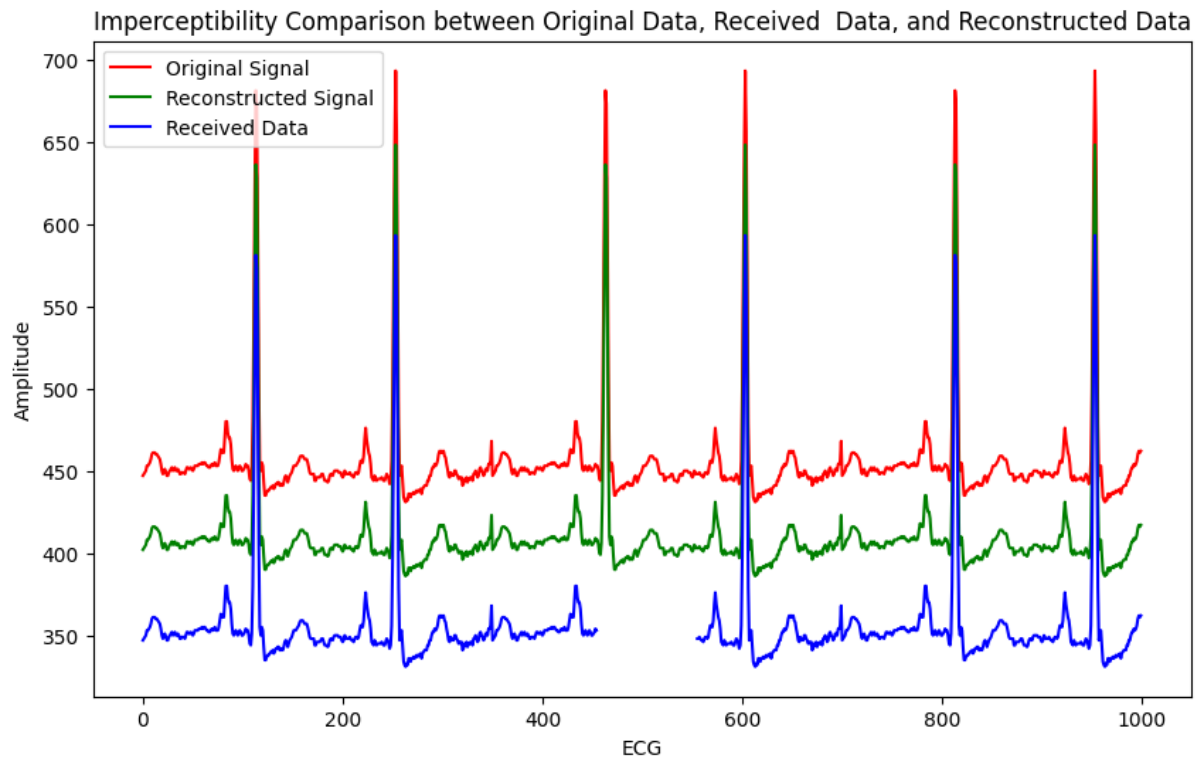


Fig.7.1.2 Imperceptibility Comparison between Original Data, Received Data and Reconstructed Data

When receiving tampered or missing data, the system demonstrates a robust capability to predict and reconstruct the original signal accurately, closely resembling its authentic form. This successful reconstruction underscores the effectiveness of the implemented techniques, particularly in scenarios where data integrity is compromised due to transmission errors or deliberate tampering. Fig 7.1.2 depicts the imperceptibility test on the various signals.

7.2 BIOSIGNAL VISUALIZATION

7.2.1 ORIGINAL BIOSIGNAL

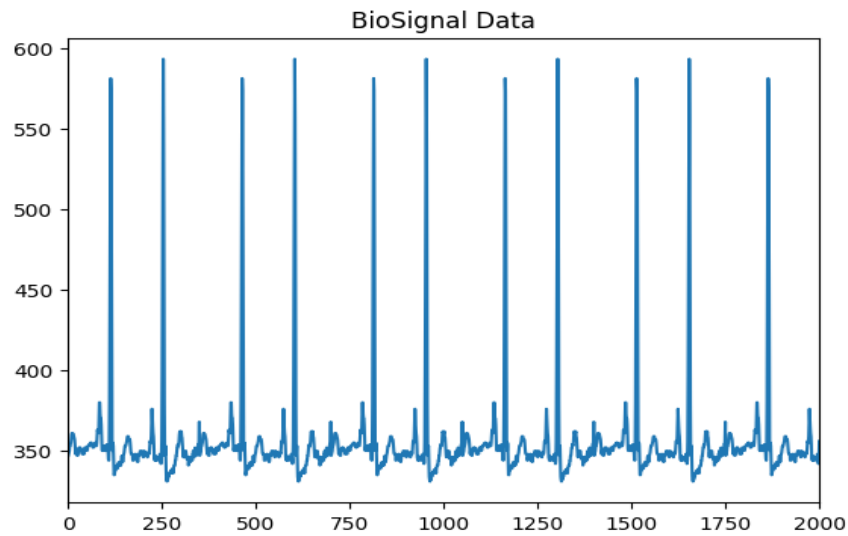


Fig.7.2.1 Original ECG BioSignal Data Taken from the ECG sensor

The biosignal data points is taken from the ECG sensor AD8232, The waveform that is generated by the biosignal is considered as Original BioSignal Data. The Original BioSignal is processed and encrypted and forms stegodata. Figure 7.2.1 shows a sample EEG signal collected using the AD8232 sensor through Arduino.

7.2.2 RECONSTRUCTED BIOSIGNAL

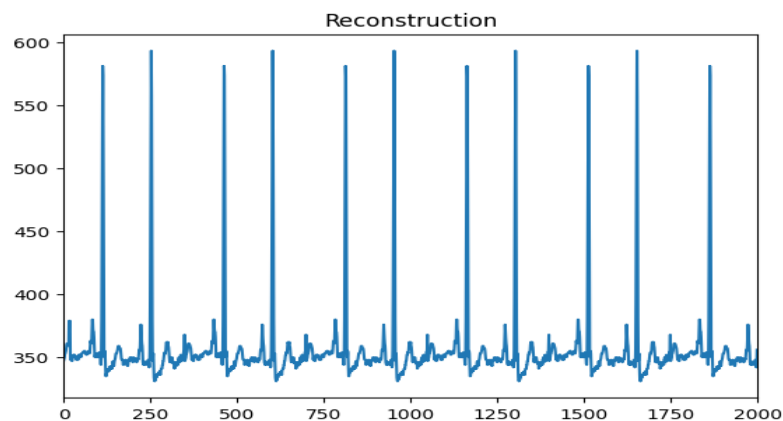


Fig.7.2.2 Reconstructed ECG BioSignal after the decryption process.

The reconstructed signal is decrypted from the stego data after receiving and password validation process, prediction of missed data block if it is tampered or loss during transmission. Figure 7.2.2 shows a sample ECG signal decrypted using the proposed framework.

7.3 HERMITE FUNCTION ORDER VISUALIZATION

7.3.1 FORWARD HERMITE FUNCTION

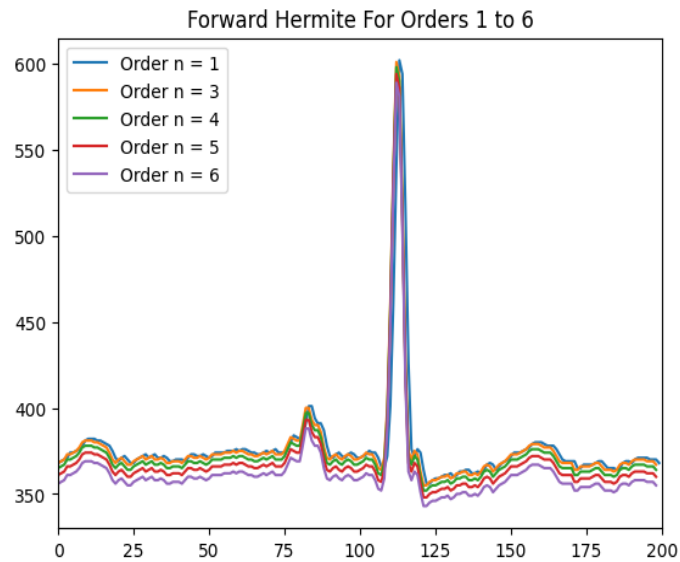


Fig 7.3.1 Forward Hermite function.

Figure 7.3.1 shows the characteristics of the various orders of the forward Hermite function range from 1-6, and this shows that even the order of the hermite function changes the signal will remains the same.

7.3.2 REVERSE HERMITE FUNCTION

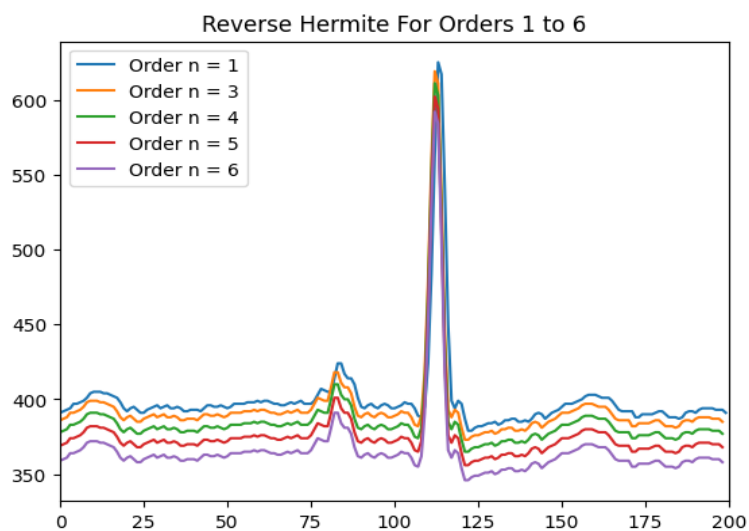


Fig 7.3.2 Reverse Hermite function.

Figure 7.3.2 shows the characteristics of the various orders of the reverse Hermite function range from 1-6, and this shows that even the order of the hermite function changes the signal will remains the same.

RESULTS AND DISCUSSION

CHAPTER 8

RESULTS AND DISCUSSION

8.1 OVERVIEW

In the result and discussion section, the outcomes of the study are elucidated, providing a comprehensive analysis of the proposed framework's efficacy in healthcare data security and communication. The evaluation of the novel deep learning-based steganography technique demonstrates its robustness in safeguarding sensitive medical information, such as Electrocardiogram (ECG), Photoplethysmography (PPG), and Electroencephalogram (EEG) data. Through extensive testing and comparison with existing methods, it was observed that the framework effectively conceals confidential data within medical signals while minimizing information loss and preserving signal integrity. The integration of the Hermite domain technique and the Hippopotamus Optimization (HO) approach contributed significantly to ensuring imperceptibility and information integrity during data embedding and decryption processes. Moreover, the inclusion of the Supervised Long-Short Term Memory (Su-LSTM) network enabled accurate prediction and reconstruction of lost signal segments, thereby offering a reversible process with minimal distortion.

8.2 DATASETS

The Datasets used in our projects are ECG, PPG, EEG Signal Data points and Arrhythmia disease dataset

| DATASET | No. of Rows |
|------------|-------------|
| ECG | 5040 |
| PPG | 10145 |
| EEG | 2100 |
| Arrhythmia | 175730 |

Table.8.2.1 ECG, PPG, EEG, Arrhythmia Dataset

8.3 RESULTS

ECG

| | Error between stego and original data | | | Error between reconstructed and original data | | |
|--------------------|---------------------------------------|--------|--------|---|---------|---------|
| No. of Secret bits | PRD | RMSE | PSNR | PRD | RMSE | PSNR |
| 1000 | 0.00002 | 0.0005 | 113.95 | 0.000498 | 0.00013 | 114.168 |
| 2000 | 0.00002 | 0.0005 | 113.85 | 0.00014 | 0.00050 | 114.08 |
| 3000 | 0.00002 | 0.0005 | 113.60 | 0.000145 | 0.00051 | 113.87 |

Table.8.3.1 PRD, RMSE, PSNR values for ECG Signal

PRD values for both stego-original and reconstructed-original data remain consistently low, indicating minimal changes during the embedding and reconstruction processes. Table 8.3.1 shows the performance in terms of error for ECG data. The RMSE values are also relatively small, suggesting close resemblance between the stego/reconstructed data and the original data. Additionally, the PSNR values are high, indicating high quality and fidelity of the stego/reconstructed data compared to the original data. These results suggest that both the steganographic embedding and reconstruction processes are effective in preserving the original data's integrity and quality across different numbers of secret bits.

EEG

| | Error between stego and original data | | | Error between reconstructed and original data | | |
|--------------------|---------------------------------------|---------|--------|---|---------|--------|
| No. of Secret bits | PRD | RMSE | PSNR | PRD | RMSE | PSNR |
| 1000 | 0.000003 | 0.00076 | 110.50 | 0.000031 | 0.0007 | 110.97 |
| 2000 | 0.000003 | 0.0007 | 110.44 | 0.000031 | 0.00072 | 110.97 |
| 3000 | 0.000003 | 0.00077 | 110.29 | 0.000032 | 0.00073 | 110.82 |

Table.8.3.2 PRD, RMSE, PSNR values for EEG Signal

Results suggest that the steganographic embedding and reconstruction processes maintain the integrity and quality of the original EEG signal data across different numbers of secret bits. Table 8.3.2 shows the performance in terms of error for EEG data. The PRD values indicate minimal changes during both processes, while the RMSE values reflect a close resemblance between the stego/reconstructed data and the original data. Additionally, the high PSNR values indicate high quality and fidelity of the stego/reconstructed data compared to the original data. Overall, these results demonstrate the effectiveness of the steganographic techniques for preserving EEG signal data

PPG

| No. of Secret bits | Error between stego and original data | | | Error between reconstructed and original data | | |
|--------------------|---------------------------------------|---------|--------|---|---------|---------|
| | PRD | RMSE | PSNR | PRD | RMSE | PSNR |
| 1000 | 0.000002 | 0.00075 | 110.52 | 0.00002 | 0.00071 | 111.038 |
| 2000 | 0.000002 | 0.00076 | 110.47 | 0.00002 | 0.00072 | 110.93 |
| 3000 | 0.000002 | 0.00074 | 110.73 | 0.00001 | 0.00069 | 111.234 |

Table.8.3.3 PRD, RMSE, PSNR values for PPG Signal

Results suggest that the steganographic embedding and reconstruction processes maintain the integrity and quality of the original PPG signal data across different numbers of secret bits. Table 8.3.3 shows the performance in terms of error for PPG data. The PRD values indicate minimal changes during both processes, while the RMSE values reflect a close resemblance between the stego/reconstructed data and the original data. Additionally, the high PSNR values indicate high quality and fidelity of the stego/reconstructed data compared to the original data. Overall, these results demonstrate the effectiveness of the steganographic techniques for preserving PPG signal data integrity

ARRHYTHMIA TRAINING MODEL ACCURACY

| Model | Accuracy |
|-----------------------------------|----------|
| Random Forest Classifier | 0.99664 |
| Gradient Boosting Classifier | 0.99236 |
| Decision Tree Classifier | 0.99125 |
| Support Vector Machine Classifier | 0.98864 |
| K-Nearest Neighbors Classifier | 0.99441 |
| MLP Classifier | 0.99460 |

Table.8.3.4 Various model accuracy of arrhythmia detection

Machine learning models play a pivotal role across diverse domains, serving as powerful tools for predictive analytics and pattern recognition. In a recent study, a comprehensive evaluation of several machine learning algorithms was conducted using a dataset to assess their predictive prowess. The findings unveiled remarkable accuracies achieved by each algorithm as tabulated in Table 8.3.4: the Random Forest Classifier excelled with an impressive accuracy of 99.664%, while the Gradient Boosting Classifier demonstrated robust predictive power at 99.236%. Additionally, the Decision Tree Classifier showcased strong performance with an accuracy of 99.125%, and the Support Vector Machine (SVM) Classifier exhibited high accuracy, reaching 98.864%. Furthermore, the k-Nearest Neighbours (KNN) Classifier delivered excellent results with an accuracy of 99.441%, while the Multilayer Perceptron (MLP) outperformed, achieving a remarkable accuracy of 99.460%.

8.4 HARDWARE SETUP

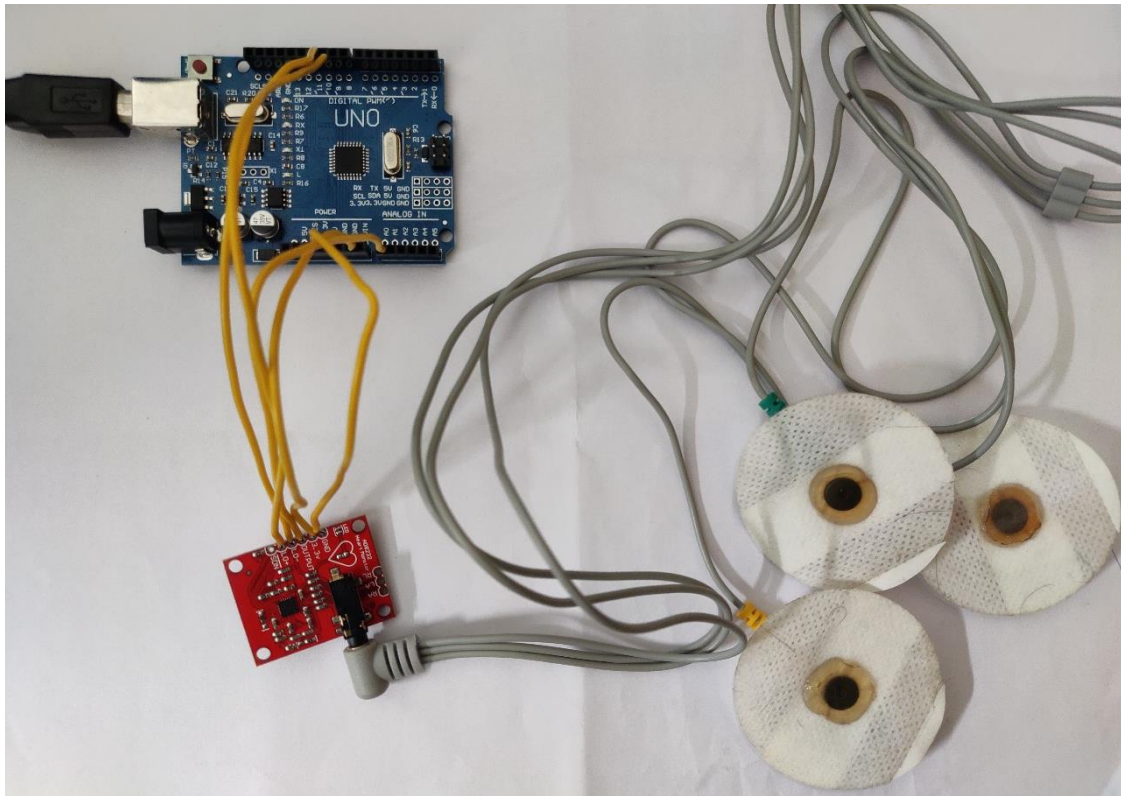


Fig.8.4.1 Hardware Connection

The hardware module like Arduino UNO board, ECG sensor kit AD8232, are connected and the setup is ready to take the ECG signal from the patient with help of probes present in the AD8232 ECG sensor. The experimental setup to collect the ECG signal is shown in Figure 8.4.1.

8.5 ARRHYTHMIA DETECTION



Fig.8.5.1 Hardware to predict the Arrhythmia - Normal

The hardware setup with arduino board and LCD display to see the result arrhythmia diasease prediction is shown in Figure 8.5.1. Here, the result of arrhythmia test is normal that is no disease.



Fig.8.5.2 Hardware to predict the Arrhythmia - Arrhythmia

The hardware setup with arduino board and LCD display to see the result arrhythmia diasease prediction is shown in Figure 8.5.2. Here, the result of arrhythmia test is positive. i.e. the subject has arrhythmia disease.

CONCLUSION AND FUTURE WORK

CHAPTER 9

CONCLUSION

9.1 CONCLUSION

In conclusion, we represent a significant advancement in the field of secure data transmission, particularly concerning ECG biosignals. By leveraging deep learning methodologies, the system offers robust encryption and decryption processes, ensuring the confidentiality and integrity of sensitive information embedded within the biosignals. The utilization of the Hermite function in both encryption and decryption phases demonstrates the efficacy of mathematical transformations in steganographic techniques. Through the intricate interplay of forward and reverse Hermite functions, along with secret bit insertion and extraction, the system achieves a high level of security while preserving the original characteristics of the biosignals. Moreover, the integration of hash functions for password generation adds an extra layer of security, enhancing the resilience of the encryption-decryption process against unauthorized access. The system's capability to recover missing data blocks through predictive modeling using MLPNN showcases its adaptability and reliability in handling data transmission errors or packet losses. While the PSO algorithm initially shows promise in optimizing missing data block positions, the superior performance of the Harmony Search algorithm underscores the importance of exploring and benchmarking different optimization techniques in future research endeavors.

9.2 FUTURE WORK

In Future, there is ample room for further innovation and improvement in several areas. Future work could involve investigating more sophisticated encryption techniques to enhance data security further. Additionally, conducting comprehensive evaluations and comparisons of various optimization algorithms, including PSO, Harmony Search, and potentially others, would provide valuable insights into their strengths and limitations in the context of missing data recovery. Furthermore, exploring advanced deep learning architectures and techniques, such as recurrent neural networks or generative adversarial networks, could potentially enhance the system's performance and capabilities. Validating the system in real-world healthcare environments and integrating it with existing biomedical systems would be crucial steps towards practical implementation and deployment. Collaborations with healthcare professionals and institutions could provide valuable feedback

and insights for refining the system's design and functionality to better meet the needs and requirements of medical practitioners and patients. Ultimately, by addressing these avenues of research and development, the project aims to make significant contributions to the advancement of secure data transmission in healthcare and biomedical applications, thereby improving patient privacy, data integrity, and overall healthcare quality

CHAPTER 10

APPENDIX

10.1 CODING

```
import hashlib
import numpy as np
import math
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler

# load the dataset
df = pd.read_csv('/content/ecg_data_t.csv')
x_data = df['Time'][:2000].values
y_data = df['DATA'][:2000].values
dataset = df.astype('float32')
print('Original Data\n',y_data)

# Normalize the data
scaler_x = MinMaxScaler()
scaler_y = MinMaxScaler()

x_data_normalized = scaler_x.fit_transform(x_data.reshape(-1,1))
y_data_normalized = scaler_y.fit_transform(y_data.reshape(-1, 1)).ravel()

plt.plot(x_data , y_data)
plt.title('BioSignal Data')
plt.xlim(0,2000)
plt.show()

plt.plot(x_data_normalized , y_data_normalized)
plt.xlim(0,1)
plt.title('Normalised BioSignal Data')
plt.show()

import sympy as sp

def hermite_forward(x, n):
    hrm_block = []
    for xi in x:
        if n == 1:
            hrm_block.append(xi + 2)
        elif n == 2:
            hrm_block.append(xi*2 - xi + 2)
        elif n == 3:
            hrm_block.append(xi*3 - xi*2 + 2)
        elif n == 4:
```

```

    hrm_block.append(xi*4 - xi*3 + 2)
elif n == 5:
    hrm_block.append(xi*5 - xi*4 + 2)
elif n == 6:
    hrm_block.append(xi*6 - xi*5 + 2)
else:
    raise ValueError("Invalid value of n, must be between 1 and 6")
return hrm_block

def reverse_hermite(y, n):
    x = sp.Symbol('x')
    equation = None
    if n == 1:
        equation = x - (y - 2)
    elif n == 2:
        equation = x*2 - x - (y - 2)
    elif n == 3:
        equation = x*3 - x*2 - (y - 2)
    elif n == 4:
        equation = x*4 - x*3 - (y - 2)
    elif n == 5:
        equation = x*5 - x*4 - (y - 2)
    elif n == 6:
        equation = x*6 - x*5 - (y - 2)
    else:
        raise ValueError("Invalid value of n, must be between 1 and 6")

    solutions = sp.solve(equation, x)
    # Filter out any complex solutions
    real_solutions = [sol.evalf() for sol in solutions if sol.is_real]
    return real_solutions

# Order of Hermite Function
n = 1

import random

ste_block = []

def embed_algorithm(cover_block,a):
    stego_block = []
    i=0
    block = []
    j=0
    cnt=1
    single_block = []
    ss_block = []
    x=0
    for yi in cover_block:
        single_block.append(yi)

```

```

    if(cnt == 4):
        block.append(hermite_forward(single_block,n))
        cnt = 0
        single_block.clear()
        cnt = cnt+1
    print("Block",block)
    for s_block in block:
        for yi in s_block:
            b1=int(a[x])
            b2=int(a[x+1])
            x+=2
            if x==len(a):
                x=0
            Y1 = round(yi, 3)
            y1 = str(Y1)
            y1=y1[:4]
            Y2 = float(y1)
            Sn = np.sign(Y1)
            p = int((Y1 * 1e3) % 10)
            p1 = round(p / 2)
            p2 = p - p1 + 1

            if b1 == 0 and b2 == 0:
                yadd = 0.0
                yfix = p
            else:
                yadd = p2 * ((b1 * 10) + b2)
                yfix = p1

            tilde_yi = Sn * ((Y2) + (yfix * 1e-3) + (yadd * 1e-5))
            tilde_yi=reverse_hermite(tilde_yi,n)
            ss_block.append(str(tilde_yi[0]))
            #print("ss_block",ss_block)
        for i in range(0,len(ss_block),4):
            ste_block.append(ss_block[i:i+4])
    print("stego_block",ste_block)
    return ste_block

def generate_hash(data):
    hash_object = hashlib.sha256(str(data).encode())
    return hash_object.hexdigest()

def string_to_binary(input_string):
    binary_result = ''.join(format(ord(char), '08b') for char in input_string)
    return binary_result

input_string = "attacktonight"
print("Secret Information: ",input_string)
binary_representation = string_to_binary(input_string)
print("Secret Information in Binary: ",binary_representation)

```

```

cover_block = y_data
print("Original Data: ",cover_block)
ste = embed_algorithm(cover_block,binary_representation)
#hashes = [generate_hash(ste[i] + ste[i + 1]) for i in range(len(ste) - 1)]
if len(ste) <= 1:
    hashes = generate_hash(ste)
else:
    hashes = [generate_hash(ste[i] + ste[i + 1]) for i in range(len(ste) - 1)]
print("Passwords:", hashes)
f = open('Data.txt','w')
ste_str = ""
for i in ste:
    ste_str = ste_str + str(i) + ' '
f.write(ste_str)
f.close()
f1 = open('Password.txt','w')
pass_str=""
for i in hashes:
    pass_str = pass_str + str(i) + ' '
f1.write(pass_str)
f1.close()

```

```

from keras.models import load_model
MLP = load_model("/content/mlp.h5")

```

```

n1=1

```

```

import sympy as sp
def hermite_forward(x, n):
    x = float(x)
    if n == 1:
        return x + 2
    elif n == 2:
        return x**2 - x + 2
    elif n == 3:
        return x*3 - x*2 + 2
    elif n == 4:
        return x*4 - x*3 + 2
    elif n == 5:
        return x*5 - x*4 + 2
    elif n == 6:
        return x*6 - x*5 + 2
    else:
        raise ValueError("Invalid value of n, must be between 1 and 6")

```

```

def reconstruction_algorithm(stego_block):
    reconstructed_block = []
    exect = []
    for i in stego_block:

```

```

for yi in i:
    yi = hermite_forward(yi,n1)
    Z1 = round(yi, 5)
    z1 = str(Z1)
    z1 = z1[:4]
    Z2 = abs(float(z1))
    Sn = np.sign(Z1)
    yfix = int((abs(Z1) * 1e3) % 10)
    q2 = int((abs(Z1) * 1e4) % 10)
    q3 = int((abs(Z1) * 1e5) % 10)

    if ((q2 == 0) and (q3 == 0)):
        b1, b2 = 0, 0
    else:
        qn = int(((q2 * 10) + q3) / 11)
        if (((q2 * 10) + q3) % 11) == 0:
            b1, b2 = 1, 1
        elif (((q2 * 10) + q3) % 10) == 0:
            b1, b2 = 1, 0
        elif (((q2 * 10) + q3) % 1) == 0:
            b1, b2 = 0, 1
        yfix = yfix + qn - 1
    exact.append(str(b1)+str(b2))
    hat_yi = Sn * (Z2 + (yfix * 1e-3))
    hat_yi=reverse_hermite(hat_yi,n1)
    #print("REv",hat_yi)
    hat_yi = round(hat_yi[0])
    #print("Round",hat_yi)
    reconstructed_block.append(hat_yi)

print('Recovered Data',reconstructed_block)
print('Extracted bit',exact)
return reconstructed_block

f = open('Data.txt','r')
ste_str = f.read()
ste_str = ste_str.replace('[',")
ste_str = ste_str.replace(']',")
ste_str = ste_str.replace(',',")
ste_str = ste_str.replace("\",")
#print("ser",ste_str)
ste_str = ste_str.split()
ste = []
for i in ste_str:
    ste.append(i)
f.close()
#print('Received data',ste)

cnt=0

```



```

block = []

block =[ste[i:i + 4] for i in range(0, len(ste), 4)]
ste = block

print("stego_block",ste)
f1 = open('Password.txt','r')
pass_str = f1.read()
generated_hash = pass_str.split()
f1.close()
print('Received Password',generated_hash)

received_data_stream = ste
j=0

pos=[]
received_hashes = [generate_hash(received_data_stream[i] + received_data_stream[i + 1])
for i in range(len(received_data_stream) - 1)]
print('Password for Received Stego Data',received_hashes)

rh = 0
pos.append(rh)
for rh in range(len(generated_hash)):
    if generated_hash[rh] in received_hashes:
        pos.append(rh+1)

vr_data=[]

missed_pos = []
nonmissed_pos = []
n = len(generated_hash)+1
j=0
flg=0

for i in range(1,n):
    if i in pos:
        nonmissed_pos.append(i)
    else:
        missed_pos.append(i)

for i in range(len(missed_pos)):
    if(i%2==0):
        nonmissed_pos.append(missed_pos[i])
        del missed_pos[i]

nonmissed_pos = sorted(nonmissed_pos)
print("Miss",missed_pos)

pred_data = []
for i in missed_pos:

```

```

data = []
for k in range(((i-1)*4)+1,(i*4)+1):
    position = np.array([[k]])
    position_normalized = scaler_x.transform(position)
    data_pred_normalized = MLP.predict(position_normalized)[0, 0]
    data_pred = scaler_y.inverse_transform([[data_pred_normalized]])[0, 0]
    data.append(str(data_pred))
    pred_data.append(data)

print("Predicted Data Block: ",pred_data)

j=0
k=0
for i in range(1,n):
    if i in nonmissed_pos:
        vr_data.append(stc[k])
        k=k+1
    else:
        vr_data.append(pred_data[j])
        j=j+1
vr_data.append(stc[k])
print("Reconstructed Block",vr_data)
rec = reconstruction_algorithm(vr_data)

plt.plot(cover_block)
plt.title('Original')
plt.xlim(0,2000)
plt.show()

plt.plot(rec)
plt.title('Reconstruction')
plt.xlim(0,2000)
plt.show()

from sklearn.metrics import mean_squared_error

def calculate_rmse(y_true, y_pred):
    return np.sqrt(mean_squared_error(y_true, y_pred))

def calculate_psnr(y_true, y_pred, max_value=255):
    rmse = calculate_rmse(y_true, y_pred)
    print('RMSE value : ',rmse)
    if(rmse == 0):
        print("The Recovered Data is same as Original Data")
        return
    else:
        psnr = 20 * math.log10(max_value / rmse)
        return psnr

P_values = cover_block

```

```

Q_values = rec
psnr_value = calculate_psnr(np.array(P_values), np.array(Q_values))
print(f"PSNR value : {psnr_value} dB")

pip install pyswarm

import numpy as np
import pandas as pd
import tensorflow as tf
from pyswarm import pso
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

df = pd.read_csv('/content/ecg_data_t.csv')
x_data = df['Time'][:200].values
y_data = df['DATA'][:200].values

plt.plot(x_data , y_data)
plt.show()

# Normalize the data
scaler_x = MinMaxScaler()
scaler_y = MinMaxScaler()

x_data_normalized = scaler_x.fit_transform(x_data.reshape(-1,1))
y_data_normalized = scaler_y.fit_transform(y_data.reshape(-1, 1)).ravel()

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x_data_normalized, y_data_normalized,
test_size=0.2, random_state=42)

# Function to create an LSTM model
def create_lstm_model(input_shape):

```

```

model = tf.keras.Sequential([
    tf.keras.layers.LSTM(10, activation='relu', input_shape=input_shape),
    tf.keras.layers.Dense(1)
])
model.compile(optimizer='adam', loss='mean_squared_error')
return model

# Function to train and evaluate the LSTM model with early stopping
def train_evaluate_lstm(x_train, y_train, x_test, y_test, epochs=5, batch_size=1):
    input_shape = (x_train.shape[1], 1)
    model = create_lstm_model(input_shape)

    # Implement early stopping
    early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)

    model.fit(x_train, y_train, epochs=epochs, batch_size=batch_size, verbose=1,
validation_data=(x_test, y_test),
        callbacks=[early_stopping])

    y_pred = model.predict(x_test, batch_size=1, verbose=0)
    mse = mean_squared_error(y_test, y_pred.squeeze())
    return mse

# PSO Objective Function
def objective_function(params, x_train, y_train, x_test, y_test):
    epochs, batch_size = params
    mse = train_evaluate_lstm(x_train, y_train, x_test, y_test, epochs=int(epochs),
batch_size=int(batch_size))
    return mse

# PSO Optimization
lb = [0.5, 0.5] # Lower bounds for epochs and batch_size

```

```

ub = [1, 1] # Upper bounds for epochs and batch_size
# PSO optimization using the objective function and bounds
best_params, _ = pso(objective_function, lb, ub, args=(x_train, y_train, x_test, y_test))

# Use the best parameters to train the final LSTM model with early stopping
best_epochs, best_batch_size = best_params
final_lstm_model = create_lstm_model((x_train.shape[1], 1))
final_lstm_model.fit(x_train, y_train, epochs=int(best_epochs),
batch_size=int(best_batch_size), verbose=1,
                    validation_data=(x_test, y_test),
callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)])

# Function to create an MLP model
def create_mlp_model(input_shape):
    model = tf.keras.Sequential([
        tf.keras.layers.Dense(50, activation='relu', input_shape=(input_shape,)),
        tf.keras.layers.Dense(1)
    ])
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model

# Train and evaluate the MLP model
mlp_model = create_mlp_model(x_train.shape[1])
mlp_model.fit(x_train, y_train, epochs=100, batch_size=8, verbose=1)

# Example: Predict y for a specific x value using the trained models
x_input = np.array([[2.5]]) # Replace with the desired value of x
x_input_normalized = scaler_x.transform(x_input)
y_lstm_pred_normalized = final_lstm_model.predict(x_input_normalized.reshape(1, 1, 1))[0,
0]
y_lstm_pred = scaler_y.inverse_transform([[y_lstm_pred_normalized]])[0, 0]

```

```

y_mlp_pred_normalized = mlp_model.predict(x_input_normalized)[0, 0]
y_mlp_pred = scaler_y.inverse_transform([[y_mlp_pred_normalized]])[0, 0]

print(f'For x={x_input[0, 0]}:')
print(f'LSTM Predicted y: {y_lstm_pred}')
print(f'MLP Predicted y: {y_mlp_pred}')

# Save entire model with optimizer, architecture, weights and training configuration.
from keras.models import load_model
final_lstm_model.save('lstm.h5')
mlp_model.save('mlp.h5')

from keras.models import load_model
final_lstm_model=load_model('/content/lstm.h5')
mlp_model=load_model('/content/mlp.h5')

# Example: Predict y for a specific x value using the trained models
x_input = np.array([[14]]) # Replace with the desired value of x
x_input_normalized = scaler_x.transform(x_input)
y_lstm_pred_normalized = final_lstm_model.predict(x_input_normalized.reshape(1, 1, 1))[0,
0]
y_lstm_pred = scaler_y.inverse_transform([[y_lstm_pred_normalized]])[0, 0]

y_mlp_pred_normalized = mlp_model.predict(x_input_normalized)[0, 0]
y_mlp_pred = scaler_y.inverse_transform([[y_mlp_pred_normalized]])[0, 0]

print(f'For x={x_input[0, 0]}:')
print(f'LSTM Predicted y: {y_lstm_pred}')
print(f'MLP Predicted y: {y_mlp_pred}')

pip install pyswarm
import numpy as np
import pandas as pd

```

```

import tensorflow as tf
from pyswarm import pso
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

df = pd.read_csv('/content/PPG_Data.csv')
x_data = df['Time'][:200].values
y_data = df['Data'][:200].values

plt.plot(x_data , y_data)
plt.show()

# Normalize the data
scaler_x = MinMaxScaler()
scaler_y = MinMaxScaler()

x_data_normalized = scaler_x.fit_transform(x_data.reshape(-1,1))
y_data_normalized = scaler_y.fit_transform(y_data.reshape(-1, 1)).ravel()

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x_data_normalized, y_data_normalized,
test_size=0.2, random_state=42)

# Function to create an LSTM model
def create_lstm_model(input_shape):
    model = tf.keras.Sequential([
        tf.keras.layers.LSTM(10, activation='relu', input_shape=input_shape),
        tf.keras.layers.Dense(1)
    ])
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model

```

```

# Function to train and evaluate the LSTM model with early stopping
def train_evaluate_lstm(x_train, y_train, x_test, y_test, epochs=5, batch_size=1):
    input_shape = (x_train.shape[1], 1)
    model = create_lstm_model(input_shape)

    # Implement early stopping
    early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)

    model.fit(x_train, y_train, epochs=epochs, batch_size=batch_size, verbose=1,
validation_data=(x_test, y_test),
        callbacks=[early_stopping])

    y_pred = model.predict(x_test, batch_size=1, verbose=0)
    mse = mean_squared_error(y_test, y_pred.squeeze())
    return mse

# PSO Objective Function
def objective_function(params, x_train, y_train, x_test, y_test):
    epochs, batch_size = params
    mse=train_evaluate_lstm(x_train,y_train,x_test,y_test,epochs=int(epochs),
batch_size=int(batch_size))
    return mse

# PSO Optimization
lb = [1, 1] # Lower bounds for epochs and batch_size
ub = [1.5, 1.5] # Upper bounds for epochs and batch_size

# PSO optimization using the objective function and bounds
best_params, _ = pso(objective_function, lb, ub, args=(x_train, y_train, x_test, y_test))

# Use the best parameters to train the final LSTM model with early stopping

```



```

best_epochs, best_batch_size = best_params
final_lstm_model = create_lstm_model((x_train.shape[1], 1))
final_lstm_model.fit(x_train, y_train, epochs=int(best_epochs),
                    batch_size=int(best_batch_size), verbose=1,
                    validation_data=(x_test, y_test),
                    callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5,
                    restore_best_weights=True)])

# Function to create an MLP model
def create_mlp_model(input_shape):
    model = tf.keras.Sequential([
        tf.keras.layers.Dense(50, activation='relu', input_shape=(input_shape,)),
        tf.keras.layers.Dense(1)
    ])
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model

# Train and evaluate the MLP model
mlp_model = create_mlp_model(x_train.shape[1])
mlp_model.fit(x_train, y_train, epochs=100, batch_size=8, verbose=1)

# Example: Predict y for a specific x value using the trained models
x_input = np.array([[2.5]]) # Replace with the desired value of x
x_input_normalized = scaler_x.transform(x_input)
y_lstm_pred_normalized = final_lstm_model.predict(x_input_normalized.reshape(1, 1, 1))[0,
0]
y_lstm_pred = scaler_y.inverse_transform([[y_lstm_pred_normalized]])[0, 0]

y_mlp_pred_normalized = mlp_model.predict(x_input_normalized)[0, 0]
y_mlp_pred = scaler_y.inverse_transform([[y_mlp_pred_normalized]])[0, 0]

print(f'For x={x_input[0, 0]}:')
print(f'LSTM Predicted y: {y_lstm_pred}')

```

```

print(f'MLP Predicted y: {y_mlp_pred}')

pip install pyswarm
import numpy as np
import pandas as pd
import tensorflow as tf
from pyswarm import pso
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

df = pd.read_csv('/content/EEG_Data.csv')
x_data = df['Time'][:200].values
y_data = df['Data'][:200].values

plt.plot(x_data , y_data)
plt.show()

# Normalize the data
scaler_x = MinMaxScaler()
scaler_y = MinMaxScaler()

x_data_normalized = scaler_x.fit_transform(x_data.reshape(-1,1))
y_data_normalized = scaler_y.fit_transform(y_data.reshape(-1, 1)).ravel()

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x_data_normalized, y_data_normalized,
test_size=0.2, random_state=42)

# Function to create an LSTM model
def create_lstm_model(input_shape):
    model = tf.keras.Sequential([

```

```

        tf.keras.layers.LSTM(10, activation='relu', input_shape=input_shape),
        tf.keras.layers.Dense(1)
    ])
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model

# Function to train and evaluate the LSTM model with early stopping
def train_evaluate_lstm(x_train, y_train, x_test, y_test, epochs=5, batch_size=1):
    input_shape = (x_train.shape[1], 1)
    model = create_lstm_model(input_shape)

    # Implement early stopping
    early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)

    model.fit(x_train, y_train, epochs=epochs, batch_size=batch_size, verbose=1,
validation_data=(x_test, y_test),
                callbacks=[early_stopping])

    y_pred = model.predict(x_test, batch_size=1, verbose=0)
    mse = mean_squared_error(y_test, y_pred.squeeze())
    return mse

# PSO Objective Function
def objective_function(params, x_train, y_train, x_test, y_test):
    epochs, batch_size = params
    mse = train_evaluate_lstm(x_train, y_train, x_test, y_test, epochs=int(epochs),
batch_size=int(batch_size))
    return mse

# PSO Optimization
lb = [1, 1] # Lower bounds for epochs and batch_size
ub = [1.5, 1.5] # Upper bounds for epochs and batch_size

```

```

# PSO optimization using the objective function and bounds
best_params, _ = pso(objective_function, lb, ub, args=(x_train, y_train, x_test, y_test))

# Use the best parameters to train the final LSTM model with early stopping
best_epochs, best_batch_size = best_params
final_lstm_model = create_lstm_model((x_train.shape[1], 1))
final_lstm_model.fit(x_train, y_train, epochs=int(best_epochs),
                    batch_size=int(best_batch_size), verbose=1,
                    validation_data=(x_test, y_test),
                    callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5,
                    restore_best_weights=True)])

# Function to create an MLP model
def create_mlp_model(input_shape):
    model = tf.keras.Sequential([
        tf.keras.layers.Dense(50, activation='relu', input_shape=(input_shape,)),
        tf.keras.layers.Dense(1)
    ])
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model

# Train and evaluate the MLP model
mlp_model = create_mlp_model(x_train.shape[1])
mlp_model.fit(x_train, y_train, epochs=100, batch_size=8, verbose=1)

# Example: Predict y for a specific x value using the trained models
x_input = np.array([[10]]) # Replace with the desired value of x
x_input_normalized = scaler_x.transform(x_input)
y_lstm_pred_normalized = final_lstm_model.predict(x_input_normalized.reshape(1, 1, 1))[0,
0]
y_lstm_pred = scaler_y.inverse_transform([[y_lstm_pred_normalized]])[0, 0]

```

```

y_mlp_pred_normalized = mlp_model.predict(x_input_normalized)[0, 0]
y_mlp_pred = scaler_y.inverse_transform([[y_mlp_pred_normalized]])[0, 0]

print(f'For x={x_input[0, 0]}:')
print(f'LSTM Predicted y: {y_lstm_pred}')
print(f'MLP Predicted y: {y_mlp_pred}')

pip install pyswarm
# Define the Harmony Search (HO) algorithm
def HO(SearchAgents, Max_iterations, lowerbound, upperbound, dimension, fitness, x_train,
y_train, x_test, y_test):

    try:
        lowerbound = np.ones(dimension) * lowerbound
        upperbound = np.ones(dimension) * upperbound

        X = np.random.rand(SearchAgents, dimension) * (upperbound - lowerbound) +
lowerbound
        fit = np.zeros(SearchAgents)

        for i in range(SearchAgents):
            fit[i] = fitness(X[i], x_train, y_train, x_test, y_test)

        best_so_far = np.zeros(Max_iterations)
        Xbest = np.zeros(dimension)
        fbest = float('inf')

        for t in range(Max_iterations):
            best, location = np.min(fit), np.argmin(fit)
            if t == 0 or best < fbest:
                fbest = best
                Xbest = X[location]

```

```

for i in range(SearchAgents // 2):
    Dominant_hippopotamus = Xbest
    I1, I2 = np.random.randint(1, 3, size=2)
    Ip1 = np.random.randint(0, 2, size=2)
    RandGroupNumber = np.random.randint(1, SearchAgents + 1)
    RandGroup = np.random.permutation(SearchAgents)[:RandGroupNumber]

    MeanGroup = np.mean(X[RandGroup], axis=0) * (len(RandGroup) != 1) +
X[RandGroup[0]] * (len(RandGroup) == 1)

    Alfa = [I2 * np.random.rand(dimension) + (1 - Ip1[0]),
            2 * np.random.rand(dimension) - 1,
            np.random.rand(dimension),
            I1 * np.random.rand(dimension) + (1 - Ip1[1]),
            np.random.rand()]

    A, B = Alfa[np.random.randint(5)], Alfa[np.random.randint(5)]

    X_P1 = X[i] + np.random.rand() * (Dominant_hippopotamus - I1 * X[i])
    T = np.exp(-t / Max_iterations)

    if T > 0.6:
        X_P2 = X[i] + A * (Dominant_hippopotamus - I2 * MeanGroup)
    else:
        if np.random.rand() > 0.5:
            X_P2 = X[i] + B * (MeanGroup - Dominant_hippopotamus)
        else:
            X_P2 = (upperbound - lowerbound) * np.random.rand(dimension) +
lowerbound

    X_P2 = np.clip(X_P2, lowerbound, upperbound)
    F_P1 = fitness(X_P1, x_train, y_train, x_test, y_test)

```

```

F_P2 = fitness(X_P2, x_train, y_train, x_test, y_test)
if F_P1 < fit[i]:
    X[i] = X_P1
    fit[i] = F_P1
if F_P2 < fit[i]:
    X[i] = X_P2
    fit[i] = F_P2

for i in range(SearchAgents // 2, SearchAgents):
    predator = lowerbound + np.random.rand(dimension) * (upperbound - lowerbound)
    F_HL = fitness(predator, x_train, y_train, x_test, y_test)
    distance2Leader = np.abs(predator - X[i])
    b = np.random.uniform(2, 4)
    c = np.random.uniform(1, 1.5)
    d = np.random.uniform(2, 3)
    l = np.random.uniform(-2 * np.pi, 2 * np.pi)
    RL = 0.05 * levy(SearchAgents, dimension, 1.5)

    if fit[i] > F_HL:
        X_P3 = RL[i] * predator + (b / (c - d * np.cos(l))) * (1 / distance2Leader)
    else:
        X_P3 = RL[i] * predator + (b / (c - d * np.cos(l))) * (1 / (2 * distance2Leader +
np.random.rand()))

X_P3 = np.clip(X_P3, lowerbound, upperbound)
F_P3 = fitness(X_P3, x_train, y_train, x_test, y_test)

if F_P3 < fit[i]:
    X[i] = X_P3
    fit[i] = F_P3

for i in range(SearchAgents):
    LO_LOCAL = lowerbound / t

```

```

HI_LOCAL = upperbound / t
D = [2 * np.random.rand(dimension) - 1, np.random.rand(), np.random.randn()]
D = D[np.random.randint(3)]
X_P4 = X[i] + np.random.rand() * (LO_LOCAL + D * (HI_LOCAL -
LO_LOCAL))
X_P4 = np.clip(X_P4, lowerbound, upperbound)
F_P4 = fitness(X_P4, x_train, y_train, x_test, y_test)
if F_P4 < fit[i]:
    X[i] = X_P4
    fit[i] = F_P4

best_so_far[t] = fbest
print('Iteration { }: Best Cost = {}'.format(t+1, best_so_far[t]))

Best_score = fbest
Best_pos = Xbest
HO_curve = best_so_far
except :
    print("Optimized")
    Best_score = fbest
    Best_pos = Xbest
    HO_curve = best_so_far
return Best_score, Best_pos, HO_curve

import numpy as np
import math
def levy(n, m, beta):
    num = math.gamma(1 + beta) * np.sin(np.pi * beta / 2) # used for Numerator
    den = math.gamma((1 + beta) / 2) * beta * 2 ** ((beta - 1) / 2) # used for Denominator
    sigma_u = (num / den) ** (1 / beta) # Standard deviation
    u = np.random.normal(0, sigma_u, size=(n, m))

    v = np.random.normal(0, 1, size=(n, m))

```



```

z = u / (np.abs(v) ** (1 / beta))

return z

import numpy as np
import pandas as pd
import tensorflow as tf
from pyswarm import pso
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import warnings
warnings.filterwarnings('ignore')

df = pd.read_csv('/content/ecg_data_t.csv')
x_data = df['Time'][:200].values
y_data = df['DATA'][:200].values

plt.plot(x_data , y_data)
plt.show()

# Normalize the data
scaler_x = MinMaxScaler()
scaler_y = MinMaxScaler()

x_data_normalized = scaler_x.fit_transform(x_data.reshape(-1,1))
y_data_normalized = scaler_y.fit_transform(y_data.reshape(-1, 1)).ravel()

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x_data_normalized, y_data_normalized,
test_size=0.2, random_state=42)

```

```

def create_lstm_model(input_shape):
    model = tf.keras.Sequential([
        tf.keras.layers.LSTM(10, activation='relu', input_shape=input_shape),
        tf.keras.layers.Dense(1)
    ])
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model

def train_evaluate_lstm(x_train, y_train, x_test, y_test, epochs=5, batch_size=1):
    input_shape = (x_train.shape[1], 1)
    model = create_lstm_model(input_shape)

    # Implement early stopping
    early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5,
        restore_best_weights=True)

    model.fit(x_train, y_train, epochs=epochs, batch_size=batch_size, verbose=1,
        validation_data=(x_test, y_test),
        callbacks=[early_stopping])

    y_pred = model.predict(x_test, batch_size=1, verbose=0)
    mse = mean_squared_error(y_test, y_pred.squeeze())
    return mse

lowerbound = 0.5
upperbound = 1

# Define the fitness function for Harmony Search
def fitness(params, x_train, y_train, x_test, y_test):
    epochs, batch_size = params
    if(math.isnan(epochs) and math.isnan(batch_size) ):
        return;
    else:

```

```

    mse = train_evaluate_lstm(x_train, y_train, x_test, y_test, epochs=5,
batch_size=int(batch_size))
    return mse

# Perform Harmony Search optimization
Best_score, Best_pos, HO_curve = HO(SearchAgents=16, Max_iterations=100,
lowerbound=lowerbound, upperbound=upperbound,
    dimension=2, fitness=fitness,x_train=x_train, y_train=y_train, x_test=x_test,
y_test=y_test)

print('Cost Value : ',Best_score,' Best Position ',Best_pos)

# Function to create an MLP model
def create_mlp_model(input_shape):
    model = tf.keras.Sequential([
        tf.keras.layers.Dense(50, activation='relu', input_shape=(input_shape,)),
        tf.keras.layers.Dense(1)
    ])
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model

# Train and evaluate the MLP model
mlp_model = create_mlp_model(x_train.shape[1])
mlp_model.fit(x_train, y_train, epochs=100, batch_size=8, verbose=1)

# Example: Predict y for a specific x value using the trained models
x_input = np.array([[2.5]]) # Replace with the desired value of x
x_input_normalized = scaler_x.transform(x_input)
y_lstm_pred_normalized = model.predict(x_input_normalized.reshape(1, 1, 1))[0, 0]
y_lstm_pred = scaler_y.inverse_transform([[y_lstm_pred_normalized]])[0, 0]

```

```

y_mlp_pred_normalized = mlp_model.predict(x_input_normalized)[0, 0]
y_mlp_pred = scaler_y.inverse_transform([[y_mlp_pred_normalized]])[0, 0]

print(f'For x={x_input[0, 0]}:')
print(f'LSTM Predicted y: {y_lstm_pred}')
print(f'MLP Predicted y: {y_mlp_pred}')

# Save entire model with optimizer, architecture, weights and training configuration.
from keras.models import load_model
model.save('lstm_HO.h5')
mlp_model.save('mlp_HO.h5')

from keras.models import load_model
final_lstm_model=load_model('/content/lstm_HO.h5')
mlp_model=load_model('/content/mlp_HO.h5')

# Example: Predict y for a specific x value using the trained models
x_input = np.array([[14]]) # Replace with the desired value of x
x_input_normalized = scaler_x.transform(x_input)
y_lstm_pred_normalized = final_lstm_model.predict(x_input_normalized.reshape(1, 1, 1))[0,
0]
y_lstm_pred = scaler_y.inverse_transform([[y_lstm_pred_normalized]])[0, 0]

y_mlp_pred_normalized = mlp_model.predict(x_input_normalized)[0, 0]
y_mlp_pred = scaler_y.inverse_transform([[y_mlp_pred_normalized]])[0, 0]

print(f'For x={x_input[0, 0]}:')
print(f'LSTM Predicted y: {y_lstm_pred}')
print(f'MLP Predicted y: {y_mlp_pred}')

```

```

import pandas as pd
data_df = pd.read_csv('/content/INCART 2-lead Arrhythmia Database.csv')
print(data_df.shape)
data_df.head()

x_data = data_df.iloc[:, 2:]
y_label = data_df[['type']]

x_data.head()

y_label.value_counts()

y_label.replace(['VEB', 'SVEB', 'F', 'Q'], 'arrhythmia', inplace=True)
y_label.replace(['N'], 'normal', inplace=True)

y_label.value_counts()

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x_data, y_label, random_state=101)

from sklearn.impute import SimpleImputer

column_names = X_train.columns
imputer = SimpleImputer(strategy='mean')
X_train = imputer.fit_transform(X_train)
X_train = pd.DataFrame(X_train, columns=column_names)

from sklearn import preprocessing

```

```

min_max_scaler = preprocessing.MinMaxScaler()
X_train_scaled = min_max_scaler.fit_transform(X_train)
X_test_scaled = min_max_scaler.transform(X_test)

print(min_max_scaler.scale_)

X_train_scaled

"""**Random Forest Classifier**"""

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(random_state=101, n_estimators=150)
model.fit(X_train_scaled, y_train.to_numpy().reshape(-1))

import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import accuracy_score

n_estimators_values = [50, 100, 150, 200, 250]
train_accuracies = []
test_accuracies = []

for n_estimators in n_estimators_values:
    model = RandomForestClassifier(random_state=101, n_estimators=n_estimators)
    model.fit(X_train_scaled, y_train.to_numpy().reshape(-1))

    train_accuracy = accuracy_score(y_train, model.predict(X_train_scaled))
    train_accuracies.append(train_accuracy)

    test_accuracy = accuracy_score(y_test, model.predict(X_test_scaled))
    test_accuracies.append(test_accuracy)

```

```

plt.plot(n_estimators_values, train_accuracies, label='Training Accuracy')
plt.plot(n_estimators_values, test_accuracies, label='Testing Accuracy')
plt.xlabel('Number of Trees')
plt.ylabel('Accuracy')
plt.title('Effect of Number of Trees on Accuracy')
plt.legend()
plt.show()

```

```

from sklearn import metrics
y_pred = model.predict(X_test_scaled)
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
print("*** Confusion Matrix ***")
print(metrics.confusion_matrix(y_test, y_pred))

```

```

y_pred = model.predict(X_test_scaled[7:9])
print(X_test_scaled[7:9])
print(y_pred)

```

""""**Gradient Boosting Classifier**""""

```

from sklearn.ensemble import GradientBoostingClassifier

```

```

modelg = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, random_state=0)
modelg.fit(X_train_scaled, y_train.to_numpy().reshape(-1))

```

```

import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import accuracy_score

```

```

n_estimators_values = [50, 100, 150, 200, 250]
train_accuracies = []
test_accuracies = []

```

```

for n_estimators in n_estimators_values:
    model = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,
    random_state=0)

    model.fit(X_train_scaled, y_train.to_numpy().reshape(-1))

    train_accuracy = accuracy_score(y_train, model.predict(X_train_scaled))
    train_accuracies.append(train_accuracy)

    test_accuracy = accuracy_score(y_test, model.predict(X_test_scaled))
    test_accuracies.append(test_accuracy)

plt.plot(n_estimators_values, train_accuracies, label='Training Accuracy')
plt.plot(n_estimators_values, test_accuracies, label='Testing Accuracy')
plt.xlabel('Number of Trees')
plt.ylabel('Accuracy')
plt.title('Effect of Number of Trees on Accuracy')
plt.legend()
plt.show()

from sklearn import metrics
y_pred = modelg.predict(X_test_scaled)
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
print("*** Confusion Matrix ***")
print(metrics.confusion_matrix(y_test, y_pred))

"""***Decision Tree Classifier***"""

from sklearn.tree import DecisionTreeClassifier

dt_model = DecisionTreeClassifier(random_state=101)
dt_model.fit(X_train_scaled, y_train.to_numpy().reshape(-1))

```



```

import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import accuracy_score

n_estimators_values = [50, 100, 150, 200, 250]
train_accuracies = []
test_accuracies = []

for n_estimators in n_estimators_values:
    model = DecisionTreeClassifier(random_state=101)
    model.fit(X_train_scaled, y_train.to_numpy().reshape(-1))

    train_accuracy = accuracy_score(y_train, model.predict(X_train_scaled))
    train_accuracies.append(train_accuracy)

    test_accuracy = accuracy_score(y_test, model.predict(X_test_scaled))
    test_accuracies.append(test_accuracy)

plt.plot(n_estimators_values, train_accuracies, label='Training Accuracy')
plt.plot(n_estimators_values, test_accuracies, label='Testing Accuracy')
plt.xlabel('Number of Trees')
plt.ylabel('Accuracy')
plt.title('Effect of Number of Trees on Accuracy')
plt.legend()
plt.show()

from sklearn import metrics
y_pred = dt_model.predict(X_test_scaled)
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
print("*** Confusion Matrix ***")
print(metrics.confusion_matrix(y_test, y_pred))

"""***Support Vector Machine Classifier***"""

```

```

from sklearn.svm import SVC

svm_model = SVC(random_state=101)
svm_model.fit(X_train_scaled, y_train.to_numpy().reshape(-1))

import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import accuracy_score

n_estimators_values = [50, 100, 150, 200, 250]
train_accuracies = []
test_accuracies = []

for n_estimators in n_estimators_values:
    model = SVC(random_state=101)
    model.fit(X_train_scaled, y_train.to_numpy().reshape(-1))

    train_accuracy = accuracy_score(y_train, model.predict(X_train_scaled))
    train_accuracies.append(train_accuracy)

    test_accuracy = accuracy_score(y_test, model.predict(X_test_scaled))
    test_accuracies.append(test_accuracy)

plt.plot(n_estimators_values, train_accuracies, label='Training Accuracy')
plt.plot(n_estimators_values, test_accuracies, label='Testing Accuracy')
plt.xlabel('Number of Trees')
plt.ylabel('Accuracy')
plt.title('Effect of Number of Trees on Accuracy')
plt.legend()
plt.show()

from sklearn import metrics

```

```

y_pred = svm_model.predict(X_test_scaled)
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
print("*** Confusion Matrix ***")
print(metrics.confusion_matrix(y_test, y_pred))

"""**K Neighbors Classifier**"""

from sklearn.neighbors import KNeighborsClassifier

knn_model = KNeighborsClassifier()
knn_model.fit(X_train_scaled, y_train.to_numpy().reshape(-1))

import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import accuracy_score

n_estimators_values = [50, 100, 150, 200, 250]
train_accuracies = []
test_accuracies = []

for n_estimators in n_estimators_values:
    model = KNeighborsClassifier()
    model.fit(X_train_scaled, y_train.to_numpy().reshape(-1))

    train_accuracy = accuracy_score(y_train, model.predict(X_train_scaled))
    train_accuracies.append(train_accuracy)

    test_accuracy = accuracy_score(y_test, model.predict(X_test_scaled))
    test_accuracies.append(test_accuracy)

plt.plot(n_estimators_values, train_accuracies, label='Training Accuracy')
plt.plot(n_estimators_values, test_accuracies, label='Testing Accuracy')
plt.xlabel('Number of Trees')

```

```

plt.ylabel('Accuracy')
plt.title('Effect of Number of Trees on Accuracy')
plt.legend()
plt.show()

from sklearn import metrics
y_pred = knn_model.predict(X_test_scaled)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("*** Confusion Matrix ***")
print(metrics.confusion_matrix(y_test, y_pred))

"""**MLP Classifier**"""

from sklearn.neural_network import MLPClassifier

mlp_model = MLPClassifier(random_state=101)
mlp_model.fit(X_train_scaled, y_train.to_numpy().reshape(-1))

import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import accuracy_score

n_estimators_values = [50, 100, 150, 200, 250]
train_accuracies = []
test_accuracies = []

for n_estimators in n_estimators_values:
    mlp_model = MLPClassifier(random_state=101)
    mlp_model.fit(X_train_scaled, y_train.to_numpy().reshape(-1))

    train_accuracy = accuracy_score(y_train, model.predict(X_train_scaled))
    train_accuracies.append(train_accuracy)

```

```

test_accuracy = accuracy_score(y_test, model.predict(X_test_scaled))
test accuracies.append(test_accuracy)

plt.plot(n_estimators_values, train_accuracies, label='Training Accuracy')
plt.plot(n_estimators_values, test_accuracies, label='Testing Accuracy')
plt.xlabel('Number of Trees')
plt.ylabel('Accuracy')
plt.title('Effect of Number of Trees on Accuracy')
plt.legend()
plt.show()

from sklearn import metrics
y_pred = mlp_model.predict(X_test_scaled)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("*** Confusion Matrix ***")
print(metrics.confusion_matrix(y_test, y_pred))

"""**LGB**"""

import lightgbm as lgb

lgb_model = lgb.LGBMClassifier(random_state=101,force_col_wise=True)
lgb_model.fit(X_train_scaled, y_train.to_numpy().reshape(-1))

import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import accuracy_score

n_estimators_values = [50, 100, 150, 200, 250]
train_accuracies = []
test_accuracies = []

for n_estimators in n_estimators_values:

```

```

lgb_model = lgb.LGBMClassifier(random_state=101,force_col_wise=True)
lgb_model.fit(X_train_scaled, y_train.to_numpy().reshape(-1))

train_accuracy = accuracy_score(y_train, model.predict(X_train_scaled))
train_accuracies.append(train_accuracy)

test_accuracy = accuracy_score(y_test, model.predict(X_test_scaled))
test_accuracies.append(test_accuracy)

plt.plot(n_estimators_values, train_accuracies, label='Training Accuracy')
plt.plot(n_estimators_values, test_accuracies, label='Testing Accuracy')
plt.xlabel('Number of Trees')
plt.ylabel('Accuracy')
plt.title('Effect of Number of Trees on Accuracy')
plt.legend()
plt.show()

from sklearn import metrics
y_pred = lgb_model.predict(X_test_scaled)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("*** Confusion Matrix ***")
print(metrics.confusion_matrix(y_test, y_pred))

import joblib
from keras.models import load_model
joblib.dump(model,'arrhythmia.joblib')

loaded_model = joblib.load('arrhythmia.joblib')

# Use the loaded model for prediction
y_pred = loaded_model.predict(X_test)

```

```

# Calculate accuracy using the loaded model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

import hashlib
import numpy as np
import math
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler

# load the dataset
df = pd.read_csv('/content/ecg_data_t.csv')
x_data = df['Time'][:200].values
y_data = df['DATA'][:200].values
print('Original Data\n',y_data)

# Normalize the data
scaler_x = MinMaxScaler()
scaler_y = MinMaxScaler()

x_data_normalized = scaler_x.fit_transform(x_data.reshape(-1,1))
y_data_normalized = scaler_y.fit_transform(y_data.reshape(-1, 1)).ravel()
plt.plot(x_data , y_data)
plt.xlim(0,200)
plt.show()

plt.plot(x_data_normalized , y_data_normalized)
plt.xlim(0,1)
plt.show()

import sympy as sp

```

```

def hermite_forward(x, n):
    if n == 1:
        return x + 2
    elif n == 2:
        return x**2 - x + 2
    elif n == 3:
        return x*3 - x*2 + 2
    elif n == 4:
        return x*4 - x*3 + 2
    elif n == 5:
        return x*5 - x*4 + 2
    elif n == 6:
        return x*6 - x*5 + 2
    else:
        print("Invalid value of n, must be between 1 and 6")

```

```

def reverse_hermite(y, n):
    x = sp.Symbol('x')
    equation = None
    if n == 1:
        equation = x - (y - 2)
    elif n == 2:
        equation = x**2 - x - (y - 2)
    elif n == 3:
        equation = x*3 - x*2 - (y - 2)
    elif n == 4:
        equation = x*4 - x*3 - (y - 2)
    elif n == 5:
        equation = x*5 - x*4 - (y - 2)
    elif n == 6:
        equation = x*6 - x*5 - (y - 2)
    else:
        raise ValueError("Invalid value of n, must be between 1 and 6")

```



```

solutions = sp.solve(equation, x)
# Filter out any complex solutions
real_solutions = [sol.evalf() for sol in solutions if sol.is_real]
return real_solutions

import random

FH_Data = []
RH_Data = []

def embed_algorithm(cover_block,n):
    stego_block = []
    i=0
    for yi in cover_block:
        yi = hermite_forward(yi,n)
        FH_Data.append(yi)
        b1 = random.randint(0,1)
        b2 = random.randint(0,1)
        Y1 = round(yi, 3)
        y1 = str(Y1)
        y1=y1[:4]
        Y2 = float(y1)
        Sn = np.sign(Y1)
        p = int((Y1 * 1e3) % 10)
        p1 = round(p / 2)
        p2 = p - p1 + 1

        if b1 == 0 and b2 == 0:
            yadd = 0.0
            yfix = p
        else:
            yadd = p2 * ((b1 * 10) + b2)

```

```

yfix = p1

tilde_yi = Sn * ((Y2) + (yfix * 1e-3) + (yadd * 1e-5))
tilde_yi=reverse_hermite(tilde_yi,n)
RH_Data.append(tilde_yi[0])
stego_block.append(round(tilde_yi[0],5))
for i in range(len(RH_Data)):
    RH_Data[i] = RH_Data[i] + 5 + n
for i in range(len(FH_Data)):
    FH_Data[i] = FH_Data[i] + n
return stego_block

def generate_hash(data):
    hash_object = hashlib.sha256(str(data).encode())
    return hash_object.hexdigest()

import numpy as np
import matplotlib.pyplot as plt

cover_block = y_data
for n in range(1,7):
    if n != 2:
        ste = embed_algorithm(cover_block,n)

plt.plot(FH_Data[:200],label='Order n = 1')
plt.plot(FH_Data[201:400],label='Order n = 3')
plt.plot(FH_Data[401:600],label='Order n = 4')
plt.plot(FH_Data[601:800],label='Order n = 5')
plt.plot(FH_Data[801:1000],label='Order n = 6')
plt.title('Forward Hermite For Orders 1 to 6')

```

```
plt.legend(loc='upper left')
plt.xlim(0,200)
plt.show()
```

```
plt.plot(RH_Data[:200],label='Order n = 1')
plt.plot(RH_Data[201:400],label='Order n = 3')
plt.plot(RH_Data[401:600],label='Order n = 4')
plt.plot(RH_Data[601:800],label='Order n = 5')
plt.plot(RH_Data[801:1000],label='Order n = 6')
plt.title('Reverse Hermite For Orders 1 to 6')
```

```
plt.legend(loc='upper left')
plt.xlim(0,200)
plt.show()
```

```
import numpy as np
import matplotlib.pyplot as plt
x_data = np.arange(1,201)
y_data = ste
plt.plot(x_data , y_data)
plt.xlim(0,200)
plt.show()
```

```
import hashlib
import numpy as np
import math
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
```

```

# load the dataset
df = pd.read_csv('/content/PPG_Data.csv')
x_data = df['Time'][:1000].values
y_data = df['Data'][:1000].values
dataset = df.astype('float32')
print('Original Data\n',y_data)

# Normalize the data
scaler_x = MinMaxScaler()
scaler_y = MinMaxScaler()

x_data_normalized = scaler_x.fit_transform(x_data.reshape(-1,1))
y_data_normalized = scaler_y.fit_transform(y_data.reshape(-1, 1)).ravel()

plt.plot(x_data , y_data)
plt.xlim(0,1000)
plt.xlabel('ECG Signal')
plt.ylabel(' Amplitude')
plt.show()

plt.plot(x_data_normalized , y_data_normalized)
plt.xlim(0,1)
plt.xlabel('ECG Signal')
plt.ylabel(' Amplitude')
plt.show()

def hermite_forward(x):
    y = x+2
    return y
def reverse_hermite(y):
    x = y-2
    return x

```

```

import random

def embed_algorithm(cover_block):
    stego_block = []
    secret_bit = []
    i=0
    j=0
    print("Secret bit: ",end="")
    for yi in cover_block:

        yi = hermite_forward(yi)

        if j % 4 == 0:
            b1 = random.randint(0,1)
            b2 = random.randint(0,1)
            print(str(b1)+str(b2),end=" ")
            j+=1

        secret_bit.append(str(b1)+' '+str(b2))

        Y1 = round(yi, 3)
        y1 = str(Y1)
        y1=y1[:4]
        Y2 = float(y1)
        Sn = np.sign(Y1)
        p = int((Y1 * 1e3) % 10)
        p1 = round(p / 2)
        p2 = p - p1 + 1

        if b1 == 0 and b2 == 0:
            yadd = 0.0
            yfix = p
        else:

```

```

yadd = p2 * ((b1 * 10) + b2)
yfix = p1

tilde_yi = Sn * ((Y2) + (yfix * 1e-3) + (yadd * 1e-5))
tilde_yi=reverse_hermite(tilde_yi)
stego_block.append(round(tilde_yi,5))

print('Stego Blocks',stego_block)
#print('Secret Bit',secret_bit)
return stego_block

def generate_hash(data):
    hash_object = hashlib.sha256(str(data).encode())
    return hash_object.hexdigest()

dataset = y_data
dataset = dataset.astype('float32')

cover_block = dataset
ste = embed_algorithm(cover_block)
hashes = [generate_hash(ste[i] + ste[i + 1]) for i in range(len(ste) - 1)]
print("Passwords:", hashes)
f = open('Data.txt','w')
ste_str = ""
for i in ste:
    ste_str = ste_str + str(i) + ' '
f.write(ste_str)
f.close()
f1 = open('Password.txt','w')
pass_str=""
for i in hashes:
    pass_str = pass_str + str(i) + ' '
f1.write(pass_str)

```

```
f1.close()
```

```
def generate_hash(data):  
    hash_object = hashlib.sha256(str(data).encode())  
    return hash_object.hexdigest()
```

```
from keras.models import load_model  
MLP = load_model('/content/PPG_mlp.h5')
```

```
def reconstruction_algorithm(stego_block):
```

```
    reconstructed_block = []  
    exect = []  
    for yi in stego_block:  
        yi = hermite_forward(yi)  
        Z1 = round(yi, 5)  
        z1 = str(Z1)  
        z1 = z1[:4]  
        Z2 = float(z1)  
        Sn = np.sign(Z1)  
        yfix = int((Z1 * 1e3) % 10)  
        q2 = int((Z1 * 1e4) % 10)  
        q3 = int((Z1 * 1e5) % 10)  
  
        if ((q2 == 0) and (q3 == 0)):  
            b1, b2 = 0, 0  
        else:  
            qn = int(((q2 * 10) + q3) / 11)  
            if (((q2 * 10) + q3) % 11) == 0:  
                b1, b2 = 1, 1  
            elif (((q2 * 10) + q3) % 10) == 0:
```

```

        b1, b2 = 1, 0
    elif (((q2 * 10) + q3) % 1) == 0:
        b1, b2 = 0, 1
    yfix = yfix + qn - 1
    exect.append(str(b1)+' '+str(b2))
    hat_yi = Sn * (Z2 + (yfix * 1e-3))
    hat_yi=reverse_hermite(hat_yi)
    hat_yi = round(hat_yi,3)
    reconstructed_block.append(hat_yi)

print('Recovered Block',reconstructed_block)
print('Extracted bit',exect)
return reconstructed_block

f = open('Data.txt','r')
ste_str = f.read()
ste_str = ste_str.split()
ste = []
for i in ste_str:
    ste.append(float(i))
f.close()
print('Received data',ste)

f1 = open('Password.txt','r')
pass_str = f1.read()
generated_hash = pass_str.split()
f1.close()
print('Received hash',generated_hash)

# Example usage:
received_data_stream = ste
j=0

```



```

pos=[]
# Generate hashes for each pair of consecutive received data
received_hashes = [generate_hash(received_data_stream[i] + received_data_stream[i + 1])
for i in range(len(received_data_stream) - 1)]
print('Password for Received Stego Data',received_hashes)
# Compare the generated hashes with received hashes

rh = 0
pos.append(rh)
for rh in range(len(generated_hash)):
    if generated_hash[rh] in received_hashes:
        pos.append(rh+1)

for rh in range(len(generated_hash)):
    if rh in pos:
        continue
    else:
        print('Position of missed data',rh)
vr_data=[]

n = len(generated_hash)+1

j=0
flg=0
for i in range(n):
    if i in pos:
        vr_data.append(ste[j])
        j=j+1
    else:
        flg=1
        position = np.array([[i]])
        position_normalized = scaler_x.transform(position)
        data_pred_normalized = MLP.predict(position_normalized)[0, 0]

```

```

data_pred = scaler_y.inverse_transform([[data_pred_normalized]])[0, 0]
vr_data.append(data_pred)
print('Predicted data at position : ',data_pred)
if flg==0:
    print('No tampered')
else:
    print('Tampered and Reconstructed')
rec = reconstruction_algorithm(vr_data)

plt.plot(cover_block)
plt.title('Original')
plt.xlim(0,1000)
plt.xlabel('ECG Signal')
plt.ylabel(' Amplitude')
plt.show()

plt.plot(rec)
plt.title('Reconstruction')
plt.xlim(0,1000)
plt.xlabel('ECG Signal')
plt.ylabel(' Amplitude')
plt.show()

from sklearn.metrics import mean_squared_error

def calculate_rmse(y_true, y_pred):
    return np.sqrt(mean_squared_error(y_true, y_pred))

def calculate_psnr(y_true, y_pred, max_value=255):
    rmse = calculate_rmse(y_true, y_pred)
    print('RMSE value : ',rmse)
    psnr = 20 * math.log10(max_value / rmse)
    return psnr

```

```

# Extract 'P' and 'Q' columns

#P_values = cover_block
P_values = ste

Q_values = rec

# Calculate PSNR
psnr_value = calculate_psnr(P_values, Q_values)

print(f"PSNR value : {psnr_value} dB")

def calculate_prd(predictions, targets):
    # Calculate the squared differences between predictions and targets
    squared_diff = (predictions - targets) ** 2

    # Calculate the sum of squared differences
    sum_squared_diff = np.sum(squared_diff)

    # Calculate the sum of squared predictions
    sum_squared_predictions = np.sum(predictions ** 2)

    # Calculate the PRD
    prd = np.sqrt(sum_squared_diff / sum_squared_predictions) * 100
    return prd

predictions = cover_block

targets = ste
#targets = rec

```

```

# Calculate the PRD
prd = calculate_prd(predictions, targets)
print("PRD:", prd)

# Visualization

import numpy as np
import matplotlib.pyplot as plt

def calculate_mse(signal1, signal2):
    return np.mean((np.array(signal1) - np.array(signal2))**2)

# Assuming X, Y, Z are your signals as lists
X = cover_block[:500] # Replace with your actual signal data
Y = ste[:500] # Replace with your actual signal data
Z = rec[:500] # Replace with your actual signal data

# Apply biases
Y_biased = [y + 0.1*100 for y in Y]
Z_biased = [z + 0.2*100 for z in Z]

# Plot signals
plt.figure(figsize=(10, 6))

plt.plot(X, label=' Original Data', color='red')
plt.plot(Y_biased, label=' Stego Data(bias 0.1)', color='blue')
plt.plot(Z_biased, label=' Reconstructed Data (bias 0.2)', color='green')

# Highlight the MSE values
mse_XY = calculate_mse(X, Y_biased)
mse_XZ = calculate_mse(X, Z_biased)

```

```

plt.title('Imperceptibility Comparison between Original Data, Stego Data, and Reconstructed
Data with Biases')
plt.xlabel('ECG Signal Time')
plt.ylabel(' Amplitude')
plt.legend()
plt.show()

import numpy as np
import matplotlib.pyplot as plt

# Assuming X, Y, Z are your signals as lists
X = cover_block[:15000] # Replace with your actual signal data
Y = ste[:15000] # Replace with your actual signal data
Z = rec[:15000]# Replace with your actual signal data

# Apply biases
Y_biased = [y + 0.1 for y in Y]
Z_biased = [z + 0.01 for z in Z]

# Calculate sample-to-sample error signals
error_signal_Y = [y - x for x, y in zip(X, Y_biased)]
error_signal_Z = [z - x for x, z in zip(X, Z_biased)]

# Plot error signals
plt.figure(figsize=(10, 6))

plt.plot(error_signal_Y, label='Sample-to-Sample Error Signal Stego Data (bias 0.1)',
color='violet')
plt.plot(error_signal_Z, label='Sample-to-Sample Error Signal Reconstructed Data (bias
0.01)', color='maroon')

plt.title('Sample-to-Sample Error Signals with Biases')
plt.xlabel('ECG')

```

```

plt.ylabel('Amplitude')
plt.legend()
plt.show()

import numpy as np
import matplotlib.pyplot as plt

# Assuming X, Y, Z are your signals as lists
X = cover_block[:1000]# Replace with your actual signal data
Y = ste[:1000] # Replace with your actual signal data, with NaN for the missing segment
Z = rec[:1000] # Replace with your actual signal data

for i in range(455, 455 + 100):
    if i < len(Y):
        Y[i] = np.nan
# Apply bias
Y_missing_bias_0_55 = [y + 0.75*100 if np.isnan(y) else y for y in Y]
X1 = [x + 100 if not np.isnan(x) else x for x in X]
Z1 = [z + 0.55*100 if not np.isnan(z) else z for z in Z]
# Plot signals
plt.figure(figsize=(10, 6))

plt.plot(X1, label='Original Signal', color='red', linestyle='-')
plt.plot(Z1, label='Reconstructed Signal', color='green', linestyle='-')
plt.plot(Y_missing_bias_0_55, label='Received Data ', color='blue', linestyle='-')

plt.title('Imperceptibility Comparison between Original Data, Received Data, and
Reconstructed Data')
plt.xlabel('ECG Signal Time')
plt.ylabel('Amplitude')
plt.legend()
plt.show()

```

REFERENCES

CHAPTER 11

REFERENCES

11.1 REFERENCES

- [1] Shahid Rahman, Jamal Uddin, Habib Ullah Khan, Hameed Hussain, Ayaz Ali Khan, And Muhammad Zakarya (Senior Member, IEEE), “A Novel Steganography Technique for Digital Images Using the Least Significant Bit Substitution Method”, IEEE Access 24 November 2022.
<https://ieeexplore.ieee.org/document/9963925>
- [2] Anirban Sengupta (Senior Member, IEEE), and Mahendra Rathor (Member, IEEE), “Structural Obfuscation and Crypto-Steganography-Based Secured JPEG Compression Hardware for Medical Imaging Systems”, IEEE Access 03 January 2020.
<https://ieeexplore.ieee.org/document/8949465>
- [3] Taqwa Ahmed Alhaj, Settana Mohammed Abdulla, Mohmmmed Abdulla Elshekh Iderss, Alaa Abdalati Ahmed Ali, Fatin A. Elhaj, Muhammad Akmal Remli, and Lubna Abdelkareim Gabralla, “A Survey: To Govern, Protect, and Detect Security Principles on Internet of Medical Things (IoMT)”, IEEE Access 28 November 2022.
<https://ieeexplore.ieee.org/document/9964214>
- [4] Ghazanfar Farooq Siddiqui, Muhammad Zafar Iqbal, Khalid Saleem (Senior Member, IEEE), Zafar Saeed, Adeel Ahmed (Member, IEEE), Ibrahim A. Hameed (Senior Member, IEEE), and Muhammad Fahad Khan, “A Dynamic Three-Bit Image Steganography Algorithm for Medical and e-Healthcare Systems”, IEEE Access 02 October 2020.
<https://ieeexplore.ieee.org/document/9211478>
- [5] Mohamed Elhoseny, Gustavo Ramírez-González, Osama M. Abu-Elnasr, Shihab A. Shawkat, Arunkumar N, and Ahmed Farouk, “Secure Medical Data Transmission Model for IoT-Based Healthcare Systems”, IEEE Access 21 March 2018.
<https://ieeexplore.ieee.org/document/8320774>
- [6] Junaid Malik, Ozer Can Devecioglu, Serkan Kiranyaz (Senior Member, IEEE), Turker Ince and Moncef Gabbouj, “Real-Time Patient-Specific ECG Classification by 1D Self-Operational Neural Networks”, IEEE Transactions on Biomedical Engineering 15 December 2021.
<https://ieeexplore.ieee.org/document/9652095>

- [7] Naresh Vemishetty, Pravanjan Patra, Pankaj Kumar Jha, Krishna Bharadwaj Chivukula, Charan Kumar Vala, Agathya Jagirdar, Venkateshwarlu Y Gudur, Amit Acharyya, (Member, IEEE) and Ashudeb Dutta (Member, IEEE), “Low Power Personalized ECG Based System Design Methodology for Remote Cardiac Health Monitoring”, IEEE Access 29 November 2016.
<https://ieeexplore.ieee.org/document/7762117>
- [8] Carl Böck (Graduate Student Member, IEEE), Péter Kovács, Pablo Laguna (Fellow, IEEE), Jens Meier and Mario Huemer (Senior Member, IEEE), “ECG Beat Representation and Delineation by Means of Variable Projection”, IEEE Transactions on Biomedical Engineering 11 February 2021.
<https://ieeexplore.ieee.org/document/9353265>
- [9] Gangireddy Narendra Kumar Reddy (Graduate Student Member, IEEE), M. Sabarimalai Manikandan (Member, IEEE), and N. V. L. Narasimha Murty (Member, IEEE), “Evaluation of Objective Distortion Measures for Automatic Quality Assessment of Processed PPG Signals for Real-Time Health Monitoring Devices”, IEEE Access 31 January 2022.
<https://ieeexplore.ieee.org/document/9698086>
- [10] Mohammad Hussein Amiri, Nastaran Mehrabi Hashjin, Mohsen Montazeri, Seyedali Mirjalili and Nima Khodadadi, “Hippopotamus optimization algorithm: a novel nature-inspired optimization algorithm”, ResearchGate February 2024.
https://www.researchgate.net/publication/378597255_Hippopotamus_optimization_algorithm_a_novel_nature-inspired_optimization_algorithm
- [11] Saleha Khatun (Student Member, IEEE), Ruhi Mahajan (Student Member, IEEE) and Bashir I. Morshed (Member, IEEE), “Comparative Study of Wavelet-Based Unsupervised Ocular Artifact Removal Techniques for Single-Channel EEG Data”, IEEE Journal of Translational Engineering in Health and Medicine 22 March 2016.
<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7438792>
- [12] Lei Ren (Member, IEEE), Tao Wang, Yuanjun Laili (Member, IEEE), and Lin Zhang (Senior Member, IEEE), “A Data-Driven Self-Supervised LSTM-DeepFM Model for Industrial Soft Sensor”, IEEE Transactions on Industrial Informatics 30 November 2021.
<https://ieeexplore.ieee.org/document/9629345>

- [13] Sara Nakatani, Kohei Yamamoto, Tomoaki Ohtsuki, “Fetal Arrhythmia Detection based on Deep Learning using Fetal ECG Signals”, GLOBECOM 2022 - 2022 IEEE Global Communications Conference 04-08 December 2022.
<https://ieeexplore.ieee.org/document/10001697>
- [14] Ho Keun Kim, Myung Hoon Sunwoo, “An Automated Cardiac Arrhythmia Classification Network for 45 Arrhythmia Classes Using 12-Lead Electrocardiogram”, IEEE Access 25 March 2023.
<https://ieeexplore.ieee.org/document/10478009>
- [15] Monalisa Singha Roy, Biplab Roy, Rajarshi Gupta, Kaushik Das Sharma, “On-Device Reliability Assessment and Prediction of Missing Photoplethysmographic Data Using Deep Neural Networks” in IEEE Transactions on Biomedical Circuits and Systems on 07 October 2020.
<https://ieeexplore.ieee.org/document/9216529>
- [16] Reema Thabit, Nur Izura Udzir, Sharifah Md Yasin, Aziah Asmawi, Adnan Abdul-Aziz Gutub, “CSNTSteg: Color Spacing Normalization Text Steganography Model to Improve Capacity and Invisibility of Hidden Data”, IEEE Access 13 June 2022.
<https://ieeexplore.ieee.org/document/9795032>
- [17] Ayano Nakashima, Rei Ueno, Naofumi Homma, “AES S-Box Hardware With Efficiency Improvement Based on Linear Mapping Optimization”, IEEE Transactions on Circuits and Systems II 23 June 2022.
<https://ieeexplore.ieee.org/document/9804750>
- [18] J. Thomas Leontin Philjon, N. Venkateshvara Rao, “Metamorphic cryptography - A paradox between cryptography and steganography using dynamic encryption,” 2011 International Conference on Recent Trends in Information Technology (ICRTIT) Jun. 2011.
<https://ieeexplore.ieee.org/document/5972272>
- [19] Mohammad Saidur Rahman, Ibrahim Khalil, Xun Yi, “Reversible Biosignal Steganography Approach for Authenticating Biosignals Using Extended Binary Golay Code”, in IEEE Journal of Biomedical and Health Informatics 20 April 2020.
<https://ieeexplore.ieee.org/document/9072496>
- [20] S. Edward Jero, Palaniappan Ramu, S. Ramakrishnan, “Discrete wavelet transform and singular value decomposition based ECG steganography for secured patient information transmission”, in J Med Syst on October 2014.

<https://pubmed.ncbi.nlm.nih.gov/25187409>

- [21] S. Edward Jero, Palaniappan Ramu, “A robust ECG steganography method”, in 10th International Symposium on Medical Information and Communication Technology (ISMICT) on 27 June 2016.


<https://ieeexplore.ieee.org/document/7498893>

ANNEXURE

CHAPTER 12

ANNEXURE

12.1 JOURNAL SUBMISSION PROOF

 IEEE Internet of Things Journal

[Home](#)

[Author](#)

[Review](#)

Submission Confirmation

Print

Thank you for your submission

Submitted to
IEEE Internet of Things Journal

Manuscript ID
IoT-37136-2024

Title
Securing Healthcare Communication: A Novel Deep Learning Framework with Imperceptible Steganography and Arrhythmia Detection

Authors
Gunasekaran, Yogarajan
Ekanathan, S A
Saivijay, S
Rajasekar, M

Date Submitted
11-Apr-2024

Author Dashboard

1?

12.2 REPORT PLAGIARISM PROOF

PR_ESR_2024

ORIGINALITY REPORT

| | | | |
|------------------|------------------|--------------|----------------|
| 25% | 19% | 20% | 0% |
| SIMILARITY INDEX | INTERNET SOURCES | PUBLICATIONS | STUDENT PAPERS |

PRIMARY SOURCES

| | | |
|---|---|----|
| 1 | dokumen.pub Internet Source | 2% |
| 2 | huggingface.co Internet Source | 2% |
| 3 | Soumyendu Banerjee, Girish Kumar Singh. "A Robust Bio-Signal Steganography With Lost-Data Recovery Architecture Using Deep Learning", IEEE Transactions on Instrumentation and Measurement, 2022 Publication | 1% |
| 4 | fastercapital.com Internet Source | 1% |
| 5 | open-innovation-projects.org Internet Source | 1% |
| 6 | github.com Internet Source | 1% |
| 7 | www.cs.uoi.gr Internet Source | 1% |

