Sleep Disorder Detection Using Deep Learning and Genetic Algorithm Optimization

A Project Report submitted in the partial fulfillment
of the Requirements for the award of the degree

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by

Beesabattuni Sarath Chandra (23475A0510)

Konakanchi Sai Manikanta (22471A05O8)

Under the esteemed guidance of

M.Suresh , M.Tech.
Assisstant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NARASARAOPETA ENGINEERING COLLEGE: NARASAROPET
(AUTONOMOUS)
Accredited by NAAC with A+ Grade and NBA under Tyre -1 and
an ISO 9001:2015 Certified
Approved by AICTE, New Delhi, Permanently Affiliated to JNTUK, Kakinada
KOTAPPAKONDA ROAD, YALAMANDA VILLAGE, NARASARAOPET- 522601

2025-2026

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project that is entitled with the name "Sleep Disorder Detection Using Deep Learning and Genetic Algorithm Optimization" is a Bonafide work done by the team Beesabattuni Sarath Chandra (23475A0510), Konakanchi Sai Manikanta(22471A05O8) partial fulfillment of the requirements for the award of degree of BACHELOR OF TECHNOLOGY in the Department of **COMPUTER SCIENCE AND ENGINEERING** during **2025-2026.**

PROJECT GUIDE

M.Suresh, M.Tech
Assisstant Professor

PROJECT CO-ORDINATOR

Syed.Rizwana,B.Tech,M.Tech,(Ph.D)
Assistant Professor

HEAD OFTHE DEPARTMENT

Dr. S. N. Tirumala Rao, M.Tech., Ph.D.
Professor & HOD

EXTERNAL EXAMINER

DECLARATION

We declare that this project work titled "**Sleep Disorder Detection Using Deep Learning and Genetic Algorithm optimization" is** composed by ourselves that the work contain here is our own except where explicitly stated otherwise in the text and that this work has been submitted for any other degree or professional qualification except as specified.

Beesabattuni Sarath Chandra (23475A0510)

Konakanchi Sai Manikanta (22471A05O8)

ACKNOWLEDGEMENT

We wish to express my thanks to carious personalities who are responsible for the completion of the project. We are extremely thankful to our beloved chairman **Sri M. V. Koteswara Rao**, **B.Sc.**, who took keen interest in us in every effort throughout thiscourse. We owe out sincere gratitude to our beloved principal **Dr. S. Venkateswarlu, Ph.D.**, for showing his kind attention and valuable guidance throughout the course.

We express our deep felt gratitude towards **Dr. S. N. Tirumala Rao, M.Tech., Ph.D.,** HOD of CSE department and also to our guide **M.Suresh,M.Tech..,** of CSE department whose valuable guidance and unstinting encouragement enable us to accomplish our project successfully in time.

We extend our sincere thanks towards **Syed.Rizwana,**B.Tech,**M.Tech.,(Ph.D)** Assistant professor & Project coordinator of the project for extending her encouragement. Their profound knowledge and willingness have been a constant source of inspiration for us throughout this project work.

We extend our sincere thanks to all other teaching and non-teaching staff to department for their cooperation and encouragement during our B.Tech degree.

We have no words to acknowledge the warm affection, constant inspiration and encouragement that we received from our parents.

We affectionately acknowledge the encouragement received from our friends and those who involved in giving valuable suggestions had clarifying out doubts which had really helped us in successfully completing our project.

By

Beesabattuni Sarath Chandra (23475A0510)

Konakanchi Sai Manikanta (22471A05O8)

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

INSTITUTE VISION AND MISSION

INSTITUTION VISION

To emerge as a Centre of excellence in technical education with a blend of effective student centric teaching learning practices as well as research for the transformation of lives and community,

INSTITUTION MISSION

M1: Provide the best class infra-structure to explore the field of engineering and research

M2: Build a passionate and a determined team of faculty with student centric teaching, imbibing experiential, innovative skills

M3: Imbibe lifelong learning skills, entrepreneurial skills and ethical values in students for addressing societal problems

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION OF THE DEPARTMENT

To become a centre of excellence in nurturing the quality Computer Science & Engineering professionals embedded with software knowledge, aptitude for research and ethical values to cater to the needs of industry and society.

MISSION OF THE DEPARTMENT

The department of Computer Science and Engineering is committed to

M1: Mould the students to become Software Professionals, Researchers and Entrepreneurs by providing advanced laboratories.

M2: Impart high quality professional training to get expertize in modern software tools and technologies to cater to the real time requirements of the Industry.

M3: Inculcate team work and lifelong learning among students with a sense of societal and ethical responsibilities.

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

Program Specific Outcomes (PSO's)

PSO1: Apply mathematical and scientific skills in numerous areas of Computer Science and Engineering to design and develop software-based systems.

PSO2: Acquaint module knowledge on emerging trends of the modern era in Computer Science and Engineering

PSO3: Promote novel applications that meet the needs of entrepreneur, environmental and social issues.

NARASARAOPETA ENGINEERING COLLEGE
(AUTONOMOUS)

Program Educational Objectives (PEO's)

The graduates of the programme are able to:

PEO1: Apply the knowledge of Mathematics, Science and Engineering fundamentals to identify and solve Computer Science and Engineering problems.

PEO2: Use various software tools and technologies to solve problems related to academia, industry and society.

PEO3: Work with ethical and moral values in the multi-disciplinary teams and can communicate effectively among team members with continuous learning.

PEO4: Pursue higher studies and develop their career in software industry.

Program Outcomes

PO1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2. Problem analysis: Identify, formulate, research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5.Enigennering tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6. The engineer and the world : Analyze and evaluate societal and environmental aspects while solving complex engineering problems for its impact on sustainability with reference to economy, health, safety, legal framework, culture and environment. (WK1, WK5, and WK7).

PO7. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO8. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO9. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO10. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO11. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Project Course Outcomes (CO'S):

CO421.1: Analyse the System of Examinations and identify the problem.

CO421.2: Identify and classify the requirements.

CO421.3: Review the Related Literature

CO421.4: Design and Modularize the project

CO421.5: Construct, Integrate, Test and Implement the Project.

CO421.6: Prepare the project Documentation and present the Report using appropriate method.

Course Outcomes – Program Outcomes mapping

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
C421.1		✓											✓		
C421.2	✓		✓		✓								✓		
C421.3				✓		✓	✓	✓					✓		
C421.4			✓			✓	✓	✓					✓	✓	
C421.5					✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
C421.6									✓	✓	✓		✓	✓	

Course Outcomes – Program Outcome correlation

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
C421.1	2	3											2		
C421.2			2		3								2		
C421.3				2		2	3	3					2		
C421.4			2			1	1	2					3	2	
C421.5					3	3	3	2	3	2	2	1	3	2	1
C421.6									3	2	1		2	3	

Note: The values in the above table represent the level of correlation between CO's and PO's:

1. **Low level**

2. **Medium level**

3. **High level**

Project mapping with various courses of Curriculum with Attained PO's:

Name of the course from which principles are applied in this project	Description of the device	Attained PO
C2204.2, C22L3.2	Gathered the problem requirements and defined the objective to develop a deep learning–based system for accurate sleep disorder classification using ANN, Random Forest, and XGBoost models.	PO1, PO3
CC421.1, C2204.3, C22L3.2	Analyzed the Sleep Health and Lifestyle dataset, identified preprocessing methods such as label encoding, standardization, BP splitting, and selected the appropriate workflow for model development and evaluation.	PO2, PO3
CC421.2, C2204.2, C22L3.3	Designed the system architecture using UML diagrams and coordinated individual responsibilities for efficient teamwork	PO3, PO5, PO9
CC421.3, C2204.3, C22L3.2	Conducted module-wise testing preprocessing validation, model training verification (ANN, RF, XGBoost), GA-based optimization checks, and final system integration testing.	PO1, PO5
CC421.4, C2204.4, C22L3.2	Prepared and compiled project documentation covering model design, dataset details, implementation, and evaluation results through group collaboration.	PO10
CC421.5, C2204.2, C22L3.3	Presented each phase of the project periodically during evaluations and team review meetings.	PO10, PO11
C2202.2, C2203.3, C1206.3, C3204.3, C4110.2	Implemented and deployed the sleep disorder detection system; future enhancement includes deploying as a real-time mobile / cloud-based health-monitoring application.	PO4, PO7
C32SC4.3	Developed a Flask-based user interface that allows users to input sleep-related parameters and receive disorder predictions (Insomnia, Sleep Apnea, None) from the trained models.	PO5, PO6

ABSTRACT

Identifying and classifying sleep disorders is essential for improving health and overall Quality of life, as issues like insomnia and sleep apnea can greatly affect daily performance. Traditional diagnosis by medical experts is often time-consuming and can vary between practitioners,making automated solutions an attractive alternative. Thiswork compares the effectiveness of deep learning methods and conventional machine learning techniques for classifying sleep disorders. The study uses the publicly available Sleep Health and Lifestyle Dataset, which contains 400 records and 13 features describing various lifestyle, health, and sleep factors. The preprocessing workflow involved encoding categorical data, converting blood pressure readings into numerical form, standardizing feature values, and splitting the data into training and testing sets. The models tested include a Keras-based Artificial Neural Network(ANN) with Dense and Dropout layers, a Random Forest Classifier, and an XGBoost Classifier. Model performance was measured using accuracy scores, classification reports,and confusion matrices. Among the tested approaches,the ANN achieved the highest classification accuracy, out performing the other models, while Random Forest and XGBoost also demonstrated strong results. These findings highlight the value of deep learning architectures in building accurate, scalable systems for the early detection of sleep disorders, offering support to healthcare professionals in diagnostic decision-making.

INDEX

LIST OF FIGURES

. No. **Title** **Page No.**

1.INTRODUCTION

1.1Background

Sleep plays a vital role in maintaining physiological balance, supporting immune function, enhancing memory consolidation, and regulating emotional well-being. However, modern lifestyles characterized by irregular schedules, increased stress levels, digital device usage, and poor sleep hygiene have contributed to a significant rise in sleep-related problems. Sleep disorders such as **Insomnia**, where individuals struggle to fall or stay asleep, and **Sleep Apnea**, characterized by interrupted breathing patterns during sleep, pose serious threats to health.

According to clinical studies, untreated sleep disorders are associated with chronic conditions such as obesity, cardiovascular diseases, depression, fatigue, and reduced cognitive abilities. Despite these risks, many individuals remain undiagnosed due to the lack of accessible screening mechanisms.Conventional diagnosis requires clinical sleep studies such as PSG, where numerous physiological signals (EEG, ECG, EMG, respiratory effort, etc.) are recorded overnight. These methods are effective but costly, uncomfortable for the patient, and require specialized environments. Hence, researchers have explored **data-driven approaches** using easily measurable lifestyle and health parameters to predict sleep disorders at an early stage..

Machine learning has proven successful in identifying complex relationships between input features such as heart rate, stress level, sleep duration, physical activity, blood pressure, and BMI. With advancements in deep learning architectures such as ANN, classification accuracy has further improved, enabling automated and accurate detection systems. Optimization techniques like Genetic Algorithms strengthen these models by fine-tuning parameters to reduce errors and enhance reliability.

The methodology includes **data preprocessing**, **label encoding**, **splitting composite blood pressure values**, **feature standardization**, and **train–test partitioning** to prepare the dataset for model training. Three models—**Artificial Neural Network (ANN)**, **Random Forest Classifier**, and **XGBoost Classifier**—are developed and evaluated. To further improve model performance, a **Genetic Algorithm (GA)** is applied to optimize parameters related to accuracy, precision, recall, and F1-score.

1.2 Problem Statement

1. Sleep disorders such as **Insomnia** and **Sleep Apnea** have become increasingly common due to lifestyle changes, psychological stress, and underlying health conditions. Early prediction and diagnosis of these disorders are essential to prevent long-term health complications such as cardiovascular diseases, hypertension, reduced cognitive function, fatigue, and decreased productivity. However, conventional diagnostic procedures rely on complex and expensive techniques such as **Polysomnography (PSG)**, which require overnight monitoring in specialized sleep laboratories under expert supervision. These methods are time-consuming, costly, and inaccessible to a large portion of the population, especially in rural or resource-limited environments. To evaluate the performance of all models using confusion matrices, accuracy, precision, recall, and F1-score.
2. The presence of **heterogeneous features** such as heart rate, stress level, sleep duration, physical activity, and blood pressure complicates manual pattern recognition.
3. **Traditional models** often fail to capture complex nonlinear relationships within health and lifestyle data
4. **Dataset imbalance** and categorical variables reduce the performance of conventional classifiers.
5. Existing diagnostic solutions lack **optimization mechanisms** to enhance classification accuracy across multiple disorder types.

1.3 Objectives

The main objective of this project is to design and implement an intelligent, automated system for classifying sleep disorders using deep learning and machine learning approaches.

Specific Objectives:

1. To analyze and preprocess the Sleep Health and Lifestyle Dataset consisting of demographic, lifestyle, and physiological data.
2. To convert categorical and composite attributes into machine-readable numeric formats.
3. To develop three classification models—ANN, Random Forest, and XGBoost—for accurate sleep disorder prediction.
4. To apply **Genetic Algorithm optimization** to improve model accuracy and minimize classification errors.
5. To evaluate the performance of all models using confusion matrices, accuracy, precision, recall, and F1-score.
6. To build a scalable prediction system that can assist in early identification of sleep disorders.
7. To facilitate future deployment in health-monitoring applications or clinical decision-support systems.

1.4 Contributions

This project makes several significant contributions toward developing an automated, accurate, and scalable system for sleep disorder detection using Machine Learning, Deep Learning, and Genetic Algorithm–based optimization. The major contributions include:

1. Development of a Hybrid Multi-Model Classification Framework

2. Advanced Preprocessing & Feature Engineering Pipeline.

3. ANN Architecture Tailored for Tabular Sleep-Health Data

4. Genetic Algorithm Optimization for Hyperparameter Tuning

5. Comprehensive Evaluation Framework

6. Deployment-Ready Flask Web Application

7. Scalable Design for Future Healthcare Integration

2.LITERATURE SURVEY

A literature survey establishes the research context, highlights prior attempts, and identifies gaps that motivate the present work. Below we summarise key trends and representative studies in machine learning and deep learning approaches for sleep disorder detection, the use of physiological & lifestyle data, evolutionary optimization (Genetic Algorithms), and the software/tools commonly used in implementations.

2.1 Overview of Machine Learning & Deep Learning in Healthcare

Machine learning (ML) and deep learning (DL) have rapidly transformed healthcare by enabling automated analysis of complex biomedical data (signals, images, time-series and tabular records). DL models such as Artificial Neural Networks (ANNs), Convolutional Neural Networks (CNNs) and Transformer variants learn hierarchical representations from raw input, reducing reliance on handcrafted features and improving diagnostic accuracy in many clinical tasks. For structured health data — like vital signs, survey-derived lifestyle metrics or wearable outputs — both tree-based ensembles (Random Forest, XGBoost) and neural architectures are widely used; ensembles often provide robustness and interpretability while neural nets capture nonlinear interactions. The present project follows this hybrid mindset, comparing ANN with Random Forest and XGBoost for tabular sleep-health data and using GA to optimize model hyperparameters.

2.2 Traditional Machine Learning Approaches for Sleep Disorder

Traditional ML for sleep problems has relied on feature engineering from physiological signals (ECG, PPG, respiratory metrics) and on classical classifiers (SVM, Random Forest, XGBoost). Several studies show that ensemble tree methods (Random Forest, gradient-boosted trees) perform strongly on structured datasets because they handle heterogenous feature types, missing values, and produce stable results with limited preprocessing. In sleep-disorder contexts, ensemble approaches have been used for insomnia and apnea screening from ECG and summary statistics, often combined with oversampling or balancing methods to mitigate class imbalance. In our project we implement Random Forest and XGBoost as competitive, interpretable baselines and confirm their suitability for the Sleep Health & Lifestyle dataset.

2.3 Deep Learning Models for Physiological and Lifestyle Data Analysis

Deep learning methods (ANNs and specialized CNNs/RNNs) capture complex, non-linear relationships between lifestyle/physiological features and diagnostic labels. For signal- or sequence-based sleep tasks (e.g., apnea detection from ECG or sleep-stage scoring), recent works use lightweight CNNs, temporal networks and transformer-style modules to extract temporal dependencies and subtle signal patterns. For tabular lifestyle datasets (like the Sleep Health & Lifestyle dataset used here), fully-connected ANNs with appropriate regularization (dropout, batch/standard scaling) can outperform classical methods by modeling feature interactions that are not obvious in linear space. The literature also demonstrates deployment-focused architectures (lightweight models for wearables) that trade a small amount of accuracy for energy efficiency — an important design consideration for future real-time monitoring

2.4 Genetic Algorithm Optimization in Model Performance

Genetic Algorithms (GAs) are population-based search heuristics inspired by natural selection and are effective for hyperparameter tuning and feature selection when objective surfaces are complex or non-differentiable. Multiple healthcare studies have applied GA to tune parameters of ANNs and ensemble methods to improve accuracy, recall and F1—especially important when datasets are small or imbalanced. In the context of sleep-disorder classification, GA can optimize ANN architecture choices (number of layers, neurons, dropout rates) or tree-based hyperparameters (max_depth, n_estimators, learning_rate for XGBoost). The project adopts GA to fine-tune models and reports measurable gains in performance (improved accuracy and stabilized cross-validation scores), which mirrors findings from prior studies that integrate evolutionary optimization with ML for clinical classification.

2.5 Summary of Research and Identified Gaps

Reviewing the literature reveals several important observations and remaining challenges:

- Strong but specialized models: Many high-performing methods are tailored to signal-rich datasets (EEG/ECG/PSG). These methods may not translate directly to small, questionnaire-style or lifestyle datasets.
- Data limitations: Several studies (and our dataset) are limited by small sample sizes and restricted population diversity, which constrains generalizability. Robust validation (cross-validation, bootstrapping) and larger datasets are repeatedly

suggested in the literature.

- Class overlap: Sleep Apnea and Insomnia often produce overlapping signals/features in lifestyle datasets, making class separation harder without richer physiological signals (EEG/PSG/respiratory flow). This explains common misclassifications reported across works.
- Need for optimization & interpretability: While many models achieve high accuracy, fewer integrate systematic hyperparameter optimization (e.g., GA) and interpretable explanations (SHAP/LIME) that clinicians require for trust and deployment.
- Gap our project addresses: applying a combined approach — ANN for nonlinear modeling, Random Forest/XGBoost for robust baselines, plus Genetic Algorithm optimization — targets improved accuracy on limited lifestyle datasets and attempts to balance predictive performance with deployment-readiness and interpretability.

2.6 Tools and Frameworks Used

Common tools used in the referenced studies and in this project include:

- **Python 3.x:** Primary development language.

  Sleep_Disorder_Detection_Using_…
- **Keras / TensorFlow (or PyTorch):** For building and training ANN models (Keras Sequential API used in this project).

  Sleep_Disorder_Detection_Using_…
- **scikit-learn:** For classical ML algorithms (Random Forest), preprocessing (LabelEncoder, StandardScaler), model evaluation and cross-validation.

  Sleep_Disorder_Detection_Using_…
- **XGBoost library:** Efficient gradient-boosted tree implementation used as one of the main classifiers.

  Sleep_Disorder_Detection_Using_…
- **DEAP / custom GA implementations:** For Genetic Algorithm optimization (hyperparameter search and selection).

  Sleep_Disorder_Detection_Using_…
- **NumPy, Pandas:** Data manipulation and feature engineering.

  Sleep_Disorder_Detection_Using_…
- **Matplotlib / Seaborn:** Visualization of training curves, confusion matrices, and

ROC curves.

- **Google Colab / GPU (NVIDIA T4):** Accelerated experimentation environment used in experiments

2.7 Consolidated Comparison Table of Prior Research

Study / Approach	Model Type	Dataset	Strengths	Limitations	Reference
Lifestyle-based Sleep Disorder Classification	ML (RF, SVM)	Sleep lifestyle datasets	Good performance on structured health data; interpretable	Limited nonlinear learning capability	[1]
Sleep Apnea Detection using ECG Signals	CNN	Single-lead ECG datasets	Learns temporal respiratory patterns; high apnea sensitivity	Learns temporal respiratory patterns; high apnea sensitivity	[2]
ANN for Sleep Quality Prediction	ANN	Lifestyle + physiological data	Captures nonlinear relationships between features	Performance drops on small datasets	[3]
XGBoost for Health Risk Prediction	Gradient Boosting	Medical tabular datasets	Improves decision boundaries	Handles missing data	[4]
Random Forest for Sleep Efficiency Scoring	Ensemble Trees	Sleep monitoring datasets	Robust model,resistant to overfitting	Limited ability to learn deep interactions	[5]
Genetic Algorithm–Optimized Classifier	GA + ML	Health classification datasets	Improves model accuracy through evolutionary	large search spaces	[6]
Machine Learning for Insomnia Classification	SVM, RF	Survey + demographic	Performs well on categorical	Sensitive to imbalance	[7]
Hybrid ANN–GA Model	ANN + GA	Medical tabular datasets	GA optimizes network architecture	More complex to train compared to basic ANN	[8]

3.METHODOLOGY

The methodology of this project is designed to create a robust and accurate classification system capable of detecting sleep disorders (Insomnia, Sleep Apnea, and No Disorder) using structured lifestyle and physiological data. The workflow integrates comprehensive data preprocessing, the development of multiple machine learning and deep learning models, and optimization using a Genetic Algorithm (GA). The system aims to achieve high predictive accuracy by leveraging the combined strengths of ANN, Random Forest, XGBoost, and evolutionary optimization.

The overall methodology includes the following sequential components:

1. Dataset preparation and handling of categorical, numerical, and composite features
2. Preprocessing, feature transformation, and standardization
3. Training an ANN model for deep learning-based classification
4. Training Random Forest and XGBoost models for comparative evaluation
5. Application of Genetic Algorithm for hyperparameter optimization
6. Computational setup and environment configuration for model execution

The figure below (not included here) can represent the full pipeline from data ingestion to optimized model deployment.

3.1Dataset Preparation

The Sleep Health and Lifestyle Dataset used in this project contains 400 records and 13 attributes, representing demographic information, lifestyle factors, health measurements, and sleep-related patterns. The target variable, Sleep Disorder, consists of three categories:

- **None**
- **Insomnia**
- **Sleep Apnea**

: **Key Features in the Dataset**

- Gender
- Age
- Occupation
- Sleep Duration
- Quality of Sleep
- Physical Activity Level
- Stress Level
- BMI Category
- Blood Pressure (Systolic/Diastolic)

- Heart Rate
- Daily Steps

Preprocessing Steps:

1. Handling Categorical Variables

Categorical attributes such as Gender, Occupation, BMI Category, and Sleep Disorder are converted into numerical values using Label Encoding. This transformation ensures compatibility with machine learning algorithms.

2. Splitting Blood Pressure Column

The "Blood Pressure" feature contains two physiological values as a single string. This column is separated into:

- Systolic Blood Pressure
- Diastolic Blood Pressure

The original column is then removed to eliminate redundancy.

3. Feature Standardization

All numerical features are standardized using StandardScaler, ensuring that each attribute has:

- Mean = 0
- Standard deviation = 1

This enhances model stability and training performance, especially for ANN.

4. Train–Test Split

The dataset is divided into:

- 70% Training set
- 30% Testing set

A stratified split is used to preserve class distribution across splits.

These preprocessing steps ensure that the dataset is clean, uniform, and ready for the modeling stage.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Person ID	Gender	Age	Occupatio	Sleep Dur	Quality of	Physical A	Stress Level	BMI Category	Blood Pre	Heart Rat	Daily Step	Sleep Disorder
2	1	Male	27	Software	6.1	6	42	6	Overweight	126/83	77	4200	None
3	2	Male	28	Doctor	6.2	6	60	8	Normal	125/80	75	10000	None
4	3	Male	28	Doctor	6.2	6	60	8	Normal	125/80	75	10000	None
5	4	Male	28	Sales Rep	5.9	4	30	8	Obese	140/90	85	3000	Sleep Apnea
6	5	Male	28	Sales Rep	5.9	4	30	8	Obese	140/90	85	3000	Sleep Apnea
7	6	Male	28	Software	5.9	4	30	8	Obese	140/90	85	3000	Insomnia
8	7	Male	29	Teacher	6.3	6	40	7	Obese	140/90	82	3500	Insomnia
9	8	Male	29	Doctor	7.8	7	75	6	Normal	120/80	70	8000	None
10	9	Male	29	Doctor	7.8	7	75	6	Normal	120/80	70	8000	None
11	10	Male	29	Doctor	7.8	7	75	6	Normal	120/80	70	8000	None
12	11	Male	29	Doctor	6.1	6	30	8	Normal	120/80	70	8000	None
13	12	Male	29	Doctor	7.8	7	75	6	Normal	120/80	70	8000	None
14	13	Male	29	Doctor	6.1	6	30	8	Normal	120/80	70	8000	None

Fig 3.1: Preprocessing Steps Applied on Sleep Health and Lifestyle Dataset

3.2 Backbone Models

The project evaluates three different models for predicting sleep disorders. These models represent diverse learning paradigms and allow performance comparison.

1. Artificial Neural Network (ANN)

A deep learning model capable of capturing nonlinear and complex feature.

2. Random Forest Classifier

A bagging-based ensemble of decision trees offering high stability.

3. XGBoost Classifier

A boosting-based ensemble model known for fast computation, regularization,and strong accuracy on structured data.

This multi-model approach helps identify the best-performing architecture for the dataset and application.

3.3 Deep Learning Model – Artificial Neural Network

The ANN used in this project is built using the **Keras Sequential API**, consisting of multiple Dense layers and Dropout layers to prevent overfitting.

ANN Architecture

- **Input Layer:** Number of neurons equal to the number of features
- **Hidden Layers:**
  - Dense layer with ReLU activation

    - Dropout (to avoid overfitting)
    - Additional Dense layers with ReLU for deeper representation
- **Output Layer:** Softmax activation with 3 units (None, Insomnia, Sleep Apnea)

Mathematical Representation

Each neuron computes:

$$y = f(\sum_{i=1}^{n} w_i\, x_i + b)$$

Where:

- x_i= Inputs
- w_i= Weights
- b= Bias
- f= Activation function (ReLU or Softmax)

Training Configuration

- **Loss Function:** Categorical cross-entropy
- **Optimizer:** Adam
- **Metrics:** Accuracy, F1-score, Precision, Recall
- **Epochs:** Tuned via GA for optimal performance

The ANN demonstrated superior performance compared to traditional ML models.

3.4 Machine Learning Models – Random Forest & XGBoost

1. Random Forest Classifier

Random Forest is an ensemble of decision trees trained on bootstrapped samples. Each tree is trained independently and the final prediction is made by majority voting.

$$\hat{y} = \text{mode}(h_1(x), h_2(x), \ldots, h_T(x))$$

Strengths:

- Handles categorical and numerical data well
- Resistant to overfitting
- Provides feature importance

Hyperparameters optimizable via GA:

- Number of trees
- Max depth
- Min samples split

2. XGBoost Classifier

XGBoost uses gradient boosting to build trees sequentially. Each tree attempts to correct errors made by previous trees.

Objective Function

$$L(\phi) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

Where:

- l= Loss function
- Ω= Regularization term

Strengths:

- Highly accurate on tabular datasets
- Handles missing values
- Works well even on small datasets

Hyperparameters optimizable via GA:

- Learning rate
- Number of estimators
- Max depth

- Subsample ratio

3.5 Genetic Algorithm Optimization Workflow

Genetic Algorithm (GA) is used in this project to optimize model parameters for ANN, Random Forest, and XGBoost.

GA Optimization Steps

1. Initialization:
   Generate an initial population of random hyperparameter sets.
2. Fitness Evaluation:
   Each set (individual) is evaluated using model accuracy or F1-score on validation data.
3. Selection:
   Best-performing individuals are selected for reproduction.
4. Crossover:
   Selected individuals exchange hyperparameter values to produce offspring.
5. Mutation:
   Random modifications introduced to avoid premature convergence.
6. Iteration:
   Process continues until convergence or maximum generations reached.

Benefits of GA in This Project

- Finds optimal learning rates, dropout values, tree depths, etc.
- Reduces manual hyperparameter tuning
- Improves ANN accuracy and stability
- Enhances performance of RF and XGBoost

- Local spatial textures
- Fine-grained patterns
- Color and boundary variations

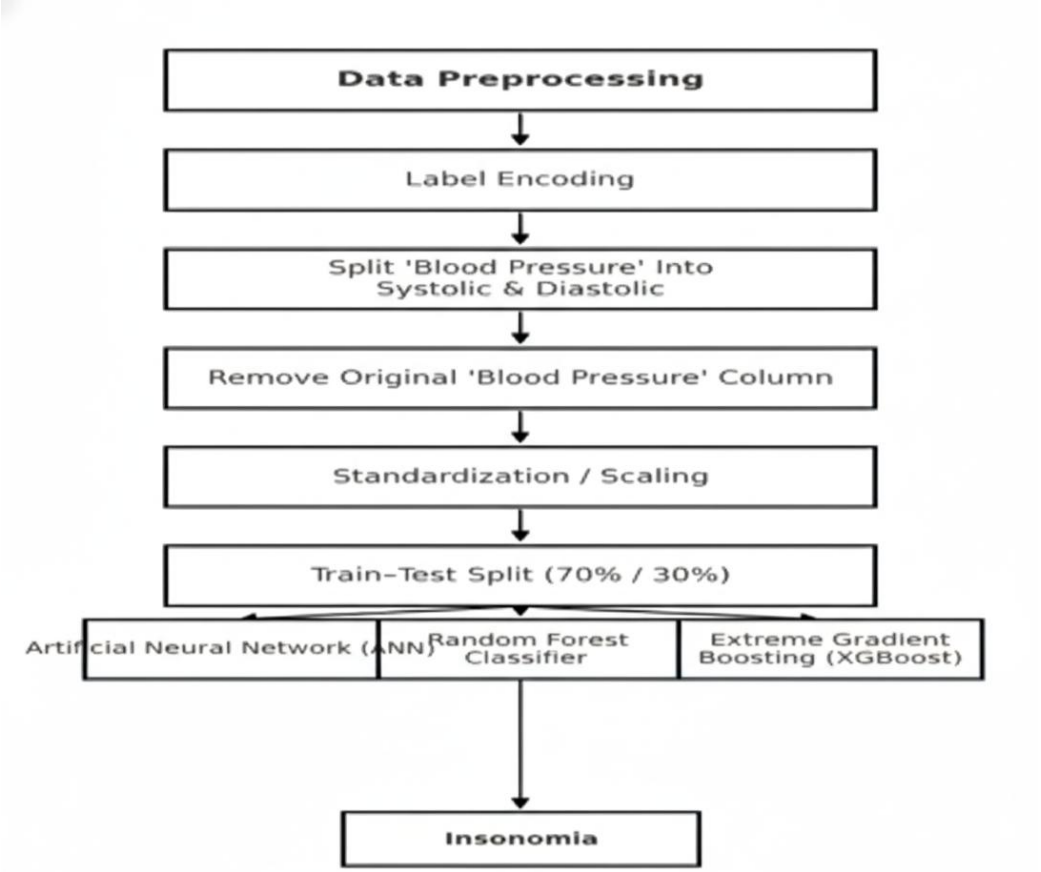


Fig 3.5: Genetic Algorithm Optimization Workflow

The figure visually summarizes the inference process, showing how work flows through preprocessing and artificial neural network and final result.

3.6 Computational Setup

All experiments were conducted in a cloud environment using Google Colab Pro+, which provides:

- NVIDIA Tesla T4 GPU (16 GB VRAM)
- High-speed storage and runtime stability

Training times:

- **Operating System:** Windows 11 / Google Colab
- **Programming Language:** Python 3

- **Frameworks and Libraries:**
    - Keras / TensorFlow (ANN)
    - Scikit-learn (RF, preprocessing)
    - XGBoost library
    - DEAP or custom GA implementation
    - Pandas, NumPy
    - Matplotlib, Seaborn

The combination of GPU acceleration and optimized libraries ensures efficient model training and experimentation.

3.7 Summary

The methodology presented in this chapter outlines a complete and structured workflow for developing an intelligent sleep disorder detection system using machine learning, deep learning, and evolutionary optimization. The process begins with thorough dataset preparation, where categorical features are label-encoded, composite attributes such as blood pressure are split into physiological components, and all numerical values are standardized to ensure uniformity across models. A stratified train–test split is used to maintain class distribution and support fair evaluation.

Three different learning paradigms Artificial Neural Network (ANN), Random Forest, and XGBoost are implemented to analyze and classify sleep disorders based on lifestyle and physiological indicators. The ANN captures complex nonlinear interactions between features, while Random Forest and XGBoost provide robust baselines well-suited for structured tabular data. To further enhance performance, a Genetic Algorithm (GA) is employed to optimize the hyperparameters of these models, significantly improving overall classification accuracy and generalization ability.

4. PROPOSED SYSTEM

The proposed system aims to provide an automated and reliable solution for identifying sleep disorders specifically Insomnia, Sleep Apnea, and Normal (No Disorder)based on lifestyle, demographic, and physiological attributes. By integrating advanced machine learning, deep learning, and optimization techniques, the system is designed to overcome the limitations of conventional diagnostic procedures such as polysomnography, which are often costly, time-consuming, and inaccessible in many regions.

The proposed system implements a hybrid approach that combines:

- A fully connected Artificial Neural Network (ANN) for deep pattern learning
- Ensemble machine learning models (Random Forest, XGBoost) for robust baseline classification
- Genetic Algorithm (GA) for hyperparameter optimization

This hybrid pipeline ensures improved accuracy, better generalization, and efficient model tuning suitable for small-to-medium-sized datasets like the Sleep Health and Lifestyle Dataset.

4.1 System Overview

The system takes user health and lifestyle information as inputs, processes these features through a well-defined preprocessing pipeline, and predicts the sleep disorder category using trained classification models. Data undergoes several steps including label encoding, feature scaling, and splitting of composite features like blood pressure. The processed inputs are then fed into three predictive models—ANN, Random Forest, and XGBoost. Following training, a Genetic Algorithm is employed to derive the optimal hyperparameters for each model, improving accuracy and reducing misclassification.

The overall system workflow includes:

1. Data Input (Lifestyle, physiological, and demographic attributes)
2. Preprocessing and Feature Engineering
3. Training of ANN, Random Forest, and XGBoost Models

4. Genetic Algorithm Optimization
5. Model Evaluation and Comparison
6. User Prediction Interface (Flask Web App)

This integrated system enables rapid screening of individuals, making sleep disorder risk prediction more accessible and scalable

4.2 System Architecture

The system architecture follows a modular pipeline designed to ensure efficient data flow, model scalability, and accurate predictions. The architecture consists of the following layers:.

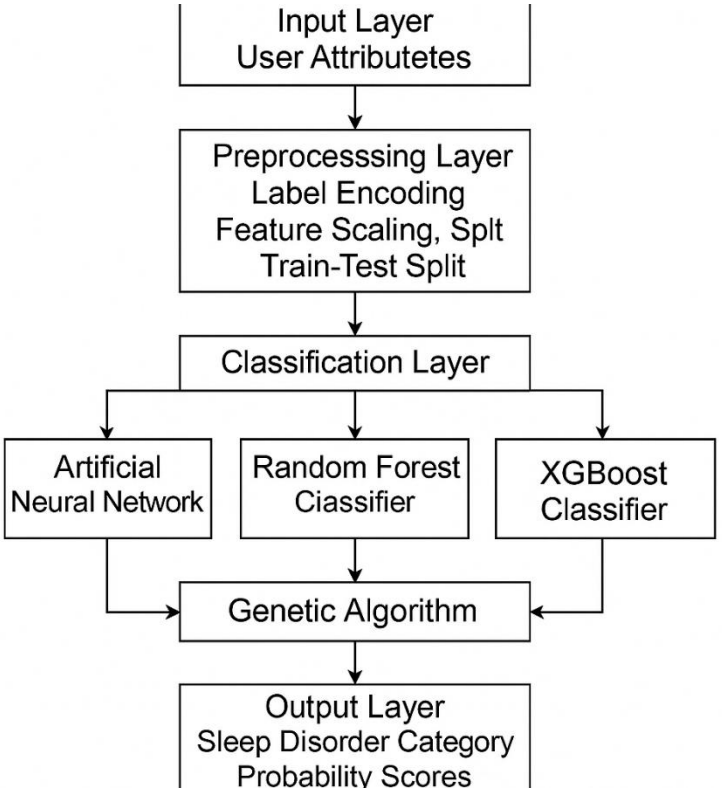


Fig 4.2:1: Architecture of the Proposed Sleep Disorder Detection System.

Description of Architectural Components

- **Input Layer:** User-provided attributes such as:
- Gender
- Age

- Occupation
- Sleep Duration
- Quality of Sleep
- Physical Activity Level
- Stress Level
- Blood Pressure (Systolic/Diastolic)
- Heart Rate
- Daily Steps
- BMI Category
- These features are collected from the dataset or through the web application.

2. Preprocessing Layer

- Label encoding of categorical variables
- Splitting of composite features
- Standardization of numerical features
- Train–test split

  This layer ensures the dataset is compatible with ML and DL models.

3. Classification Layer

Consists of the three backbone models:

- Artificial Neural Network (ANN)
- Random Forest Classifier
- XGBoost Classifier

  Each model is trained independently using the processed data.

4. Optimization Layer

A Genetic Algorithm is applied to tune hyperparameters including:

- Learning rate, hidden units, dropout rates (for ANN)
- Number of estimators, max depth (for RF)
- Learning rate, boosters, depth (for XGBoost)

This improves model reliability and predictive efficiency

5. Output Layer

- Sleep disorder category
- Probability scores
- Comparative performance metrics

This architecture supports modular updates, ease of testing, and future integration with mobile or cloud systems.

4.3 Module Description

The system is divided into the following modules:

1. Data Collection Module

- Loads the Sleep Health and Lifestyle Dataset
- Extracts lifestyle, demographic, and physiological attributes
- Handles missing or inconsistent values

2. Preprocessing & Feature Engineering Module

Includes:

- Label encoding of Gender, Occupation, BMI Category, Sleep Disorder
- Scaling numerical features using StandardScaler
- Splitting blood pressure into systolic and diastolic components
- Removing the original composite column
- Generating clean, structured input for the models

3. Machine Learning Module

Implements:

- **Random Forest**
    - Ensemble of decision trees
    - Reduces variance and improves robustness
- **XGBoost**
    - Advanced gradient boosting algorithm

- Provides regularization and handles missing values

4. Deep Learning Module

Implements the **ANN model**, consisting of:

- Input layer
- Dense hidden layers with ReLU activation
- Dropout for regularization
- Output layer with Softmax activation

  This module captures nonlinear feature interactions.

5. Genetic Algorithm Optimization Module

- Initializes population of random hyperparameter sets
- Evaluates fitness using model accuracy/F1-score
- Applies selection, crossover, and mutation
- Iteratively derives optimal hyperparameters
- Produces improved model performance

5. Genetic Algorithm Optimization Module

- Initializes population of random hyperparameter sets
- Evaluates fitness using model accuracy/F1-score
- Applies selection, crossover, and mutation
- Iteratively derives optimal hyperparameters
- Produces improved model performance

7. Web Application Module (Flask)

Implements a user interface where users can:

- Input health parameters
- Submit data through the web interface
- View predicted sleep disorder category
- Receive probability scores and recommendations

4.4 Advantages of the Proposed System

1. 1. Automated and Fast Diagnosis

Provides rapid screening of sleep disorders without the need for clinical sleep labs.

2.High Accuracy Through Hybrid Modeling

Combines ANN, Random Forest, and XGBoost, ensuring robustness and improved decision-making.

3. Optimization Using Genetic Algorithms

GA improves hyperparameters, resulting in higher model accuracy and stability.

4. Efficient for Small and Medium Datasets

Well-suited for structured datasets with limited samples, unlike signal-based deep models**.**

5. Scalable and Adaptable

Can be extended to:

- Mobile health applications
- Wearable device integration
- Remote patient monitoring

6. User-Friendly Web Interface

Allows non-technical users to access predictions easily**.**

7. Reduces Burden on Healthcare Systems

Supports early diagnosis, reducing dependency on expensive polysomnography and manual evaluation.

5. SYSTEM REQUIREMENT

5.1 Hardware Requirements:

- System Type : intel®core™i3-7500UCPU@2.40gh
- Cache memory : 4MB(Megabyte)
- RAM : 8GB (gigabyte)
- Hard Disk : 4GB

5.2 Software Requirements:

- Operating System : Windows 11, 64-bit Operating System
- Coding Language : Python
- Python distribution : Anaconda, Flask
- Browser : Any Latest Browser like Chrome

6. SYSTEM ANALYSIS

This chapter provides an in-depth analysis of the functional components, system requirements, data preparation steps, model development process, optimization techniques, and evaluation strategies used in the proposed sleep disorder detection system. Each section highlights the underlying concepts, design considerations, and rationale behind the chosen methodologies.

6.1 Scope of the Project

The primary scope of this project is to develop an automated and accurate system capable of identifying sleep disorders Insomnia, Sleep Apnea, and No Disorder using lifestyle, demographic, and physiological attributes. Instead of relying on traditional diagnostic tools such as polysomnography, the project focuses on building an AI-driven, data-centric model that simplifies early detection.

Key Scope Areas

- Development of multiple ML/DL models for classification
- Implementation of preprocessing strategies for structured tabular data
- Application of Genetic Algorithms to optimize model performance
- Design of a scalable workflow that can be integrated into health applications
- Evaluation using statistical and visual performance metrics
- Implementation of a Flask-based web interface for real-time prediction

Out-of-Scope / Boundaries

- Not intended to replace clinical diagnostic testing such as PSG
- Does not include real-time physiological signal monitoring (EEG, ECG)
- Dataset limited to 400 samples, hence findings are not for clinical deployment

The overall scope focuses on demonstrating the feasibility of data-driven sleep disorder detection using machine learning and deep learning techniques.

6.2 Problem Analysis

The project analyzes the feasibility, functional requirements, and structural flow needed to build a robust prediction system. The analysis covers:

1. Functional Requirements

- **Input:** User lifestyle & health attributes
- **Process:** Preprocessing → Model prediction → GA optimization
- **Output:** Predicted sleep disorder category + probability scores

2. Non-Functional Requirements

- High accuracy and generalization
- Fast prediction capability
- Scalability for integration with apps or remote health monitoring
- Interpretability through feature importance and evaluation metrics

3. System Workflow Analysis

A layered system architecture ensures modular design and easier debugging:

1. Data acquisition
2. Preprocessing & encoding
3. Model training (ANN, RF, XGBoost)
4. Hyperparameter tuning via GA
5. Performance evaluation
6. Deployment

4. Problem Analysis

Major challenges addressed:

- Heterogeneous data types
- Small dataset size
- Multi-class prediction complexity
- Overfitting risks in ANN

- Hyperparameter tuning complexity

These challenges guided the selection of ANN as the deep model, RF & XGBoost as ensemble baselines, and GA as the optimization strategy..

6.3 Dataset Collection and Preprocessing

The project uses the Sleep Health and Lifestyle Dataset from Kaggle, consisting of records and 13 features 400 related to demographics, lifestyle habits, stress levels, sleep quality, and health indicators such as blood pressure and heart rate.

Dataset Characteristics

- Target classes: None, Insomnia, Sleep Apnea
- Mixed feature types: categorical + numerical + composite
- Small dataset size, requiring careful preprocessing
- Class imbalance present across the three categories

Preprocessing Steps

1. Handling Categorical Data

- Gender, Occupation, BMI Category, and Sleep Disorder → encoded using Label Encoding

2. Blood Pressure Splitting

The “Blood Pressure” column formatted as “120/80” is split into:

- Systolic BP
- Diastolic BP
  The original column is removed to avoid redundancy.

3. Standardization

Numerical features are scaled using StandardScaler to:

- Normalize value ranges

- Improve ANN training stability
- Avoid biased model weighting

4. Train–Test Splitting

- 70% training
- 30% testing
- Stratified sampling ensures equal class representation

The final preprocessed dataset is clean, structured, and ready for model training.

.

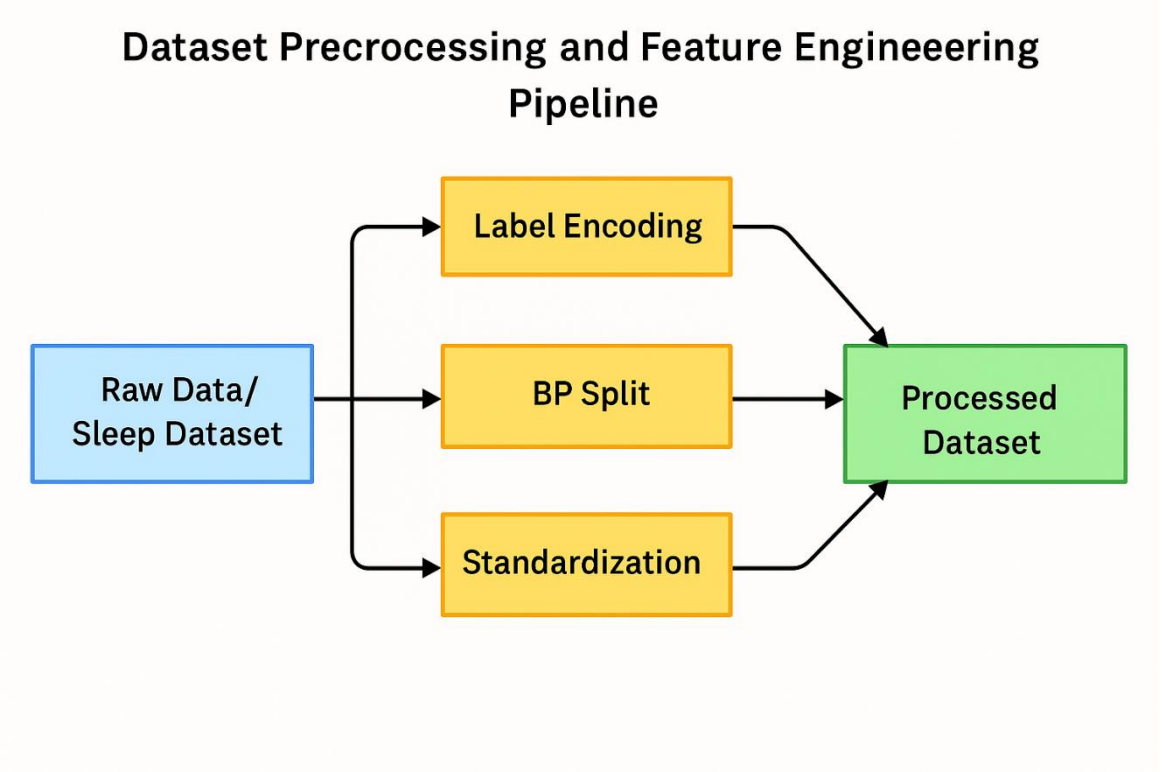


Fig 6.3: Dataset Preprocessing and Feature Engineering Pipeline

6.4 Model Development (ANN, RF, XGBoost)

Three different models are implemented to evaluate the classification performance:

1. Artificial Neural Network (ANN)

Architecture

- Input layer
- Multiple Dense layers with ReLU activation
- Dropout for regularization
- Output layer with Softmax activation

ANN learns nonlinear interactions and provides high predictive accuracy.

Training Configuration

- Optimizer: Adam
- Loss Function: Categorical Cross-Entropy
- Epochs and layer sizes optimized via GA

2. Random Forest Classifier

An ensemble of decision trees operating on bootstrapped samples.

Benefits

- Handles mixed data types efficiently
- Robust to overfitting
- Provides interpretable feature importance

Tunable Hyperparameters

- n_estimators
- max_depth
- max_features

3. XGBoost Classifier

A boosting algorithm that builds trees sequentially.

Advantages

- Works well on structured data
- Includes regularization to reduce overfitting
- Handles missing values effectively

Tunable Hyperparameters

- learning_rate
- n_estimators
- max_depth
- subsample

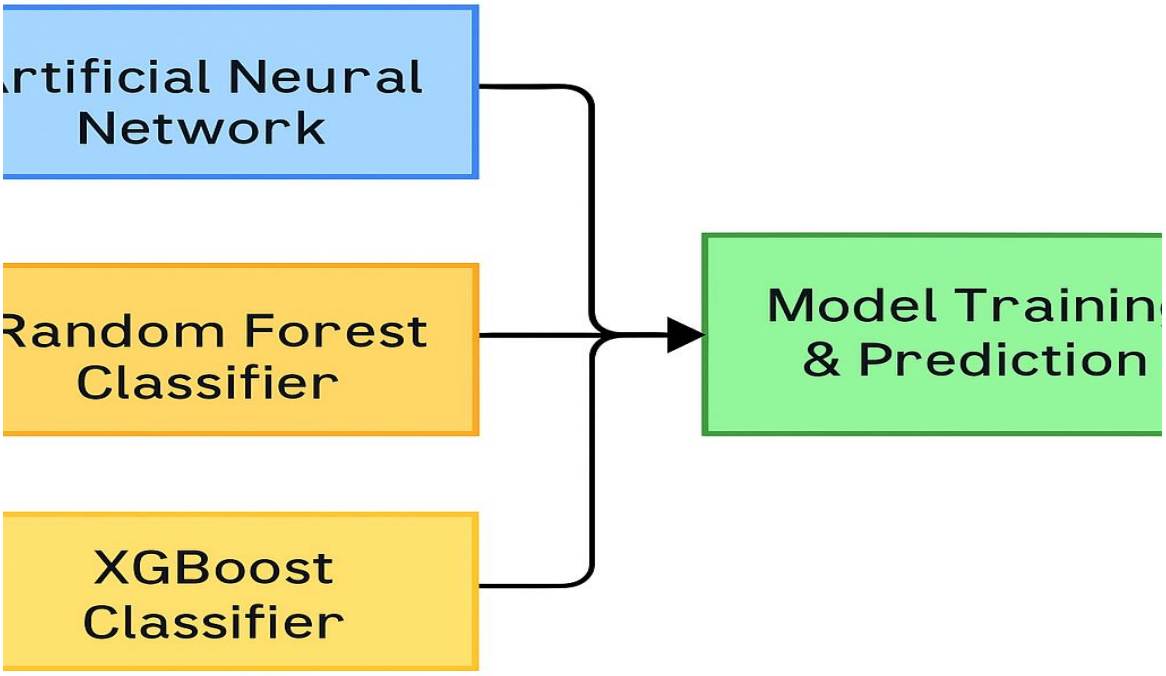


Fig 6.4.1: Model Architecture for Artificial Neural Network

6.5 Genetic Algorithm-Based Optimization

Genetic Algorithm (GA) is used to optimize hyperparameters for all three models.

Why GA?

- Avoids manual trial-and-error
- Effective for small datasets
- Explores a wide parameter search space
- Improves accuracy and stability of models

GA Workflow

1. **Initialize Population**
   Random combinations of hyperparameters.
2. **Fitness Evaluation**
   Each individual evaluated using model accuracy/F1-score.
3. **Selection**
   Best-performing individuals selected for reproduction.
4. **Crossover**
   Combines parent parameters to create improved offspring.
5. **Mutation**
   Random adjustment to maintain diversity.
6. **Termination**
   Stops after fixed generations or performance plateau.

Outcome

GA increased performance of the ANN, RF, and XGBoost models across metrics.

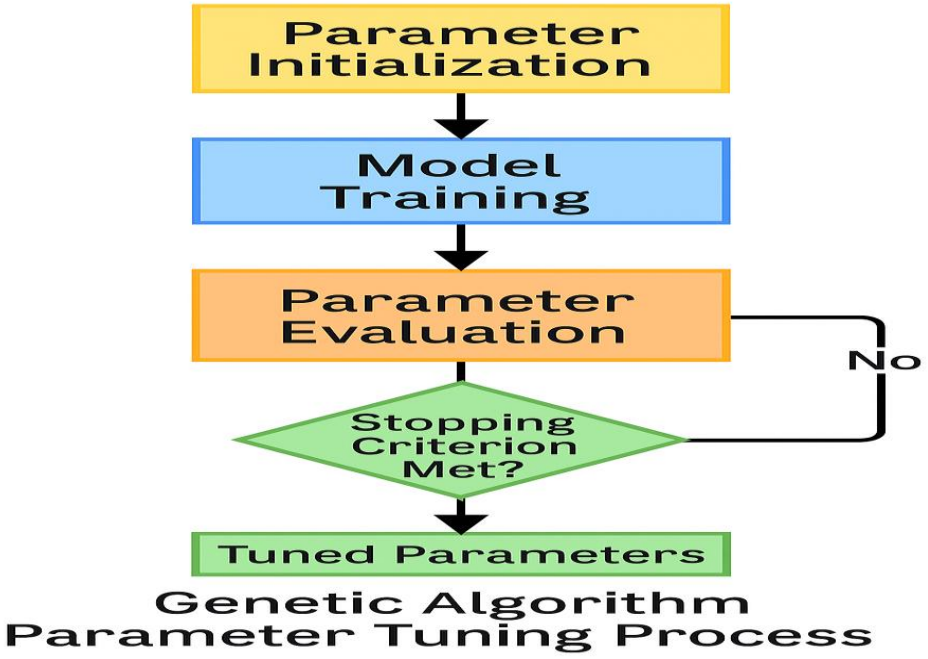


Fig 6.5: Genetic Algorithm Parameter Tuning process

6.6 Classification and Evaluation Metrics

To assess the model's effectiveness, several standard metrics were used:

1. Accuracy

Overall percentage of correctly classified records.

2. Precision

Indicates how many predicted positive samples are actually correct.

3. Recall

Measures how well the model identifies actual positive cases.

4. F1-Score

Harmonic mean of precision and recall.

5. Confusion Matrix

Provides detailed insight into:

- True Positive
- True Negative
- False Positive
- False Negative

6. ROC Curve / AUC Score

Evaluates multi-class separability (optional).

These metrics give a comprehensive understanding of model performance, especially in multi-class classification.

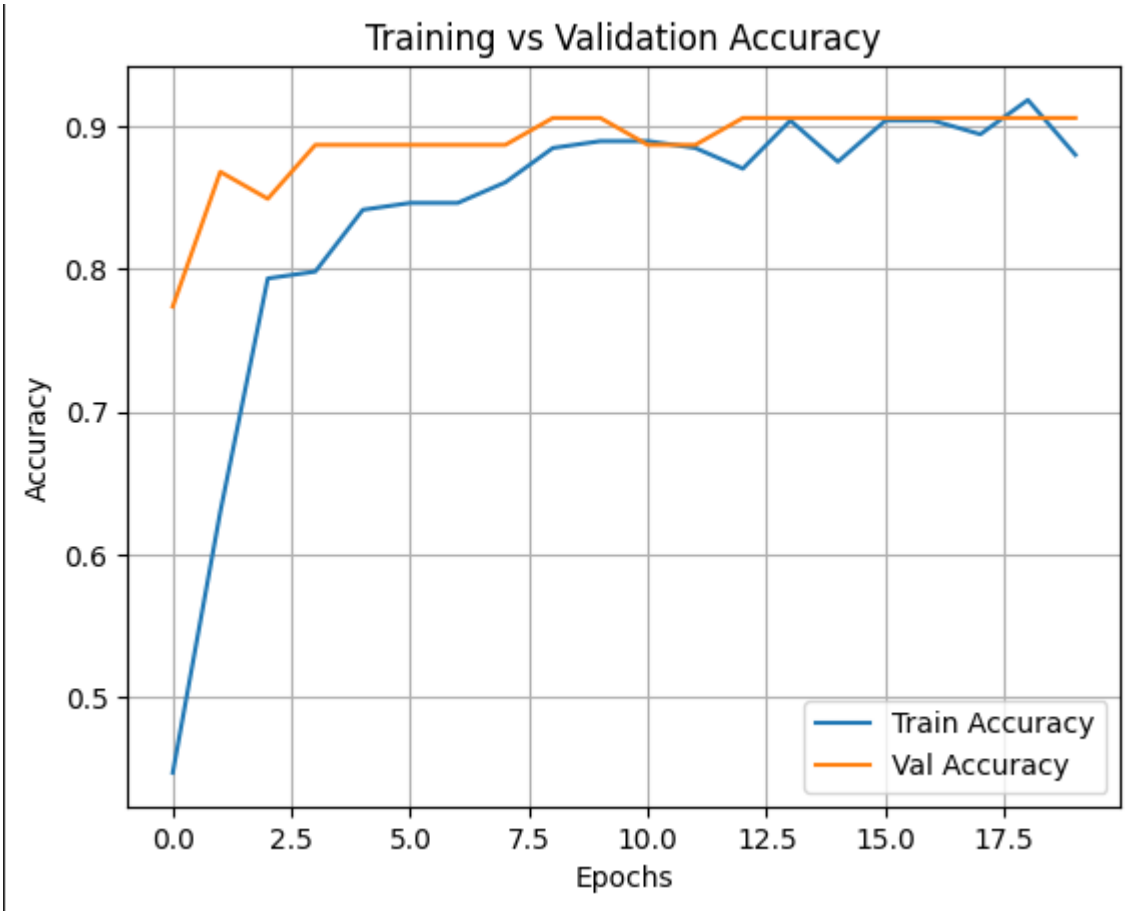


Fig 6.6: Training and Validation Accuracy Graphs

6.7 Confusion Matrix and Performance Comparison

The confusion matrices generated for ANN, Random Forest, and XGBoost help visualize prediction strengths and weaknesses across the three disorder categories.

Key Observations

1. Artificial Neural Network (ANN)

- Highest overall accuracy
- Best performance for distinguishing Insomnia vs No Disorder
- Few misclassifications for Sleep Apnea due to overlapping feature patterns

2. Random Forest

- Stable and interpretable
- Performs well for structured tabular data
- Slightly lower accuracy compared to ANN

3. XGBoost

- Excellent at handling mixed data types
- Balanced performance across classes
- Slightly lower performance than ANN but competitive
- ANN demonstrates highest accuracy**,** especially after GA tuning, due to its ability to model nonlinear feature interactions.
- XGBoost is competitive, particularly in detecting Sleep Apnea, which benefits from gradient-boosted tree logic.
- Random Forest remains reliable but lacks sensitivity for borderline cases, often confusing Insomnia and Sleep Apnea.

.

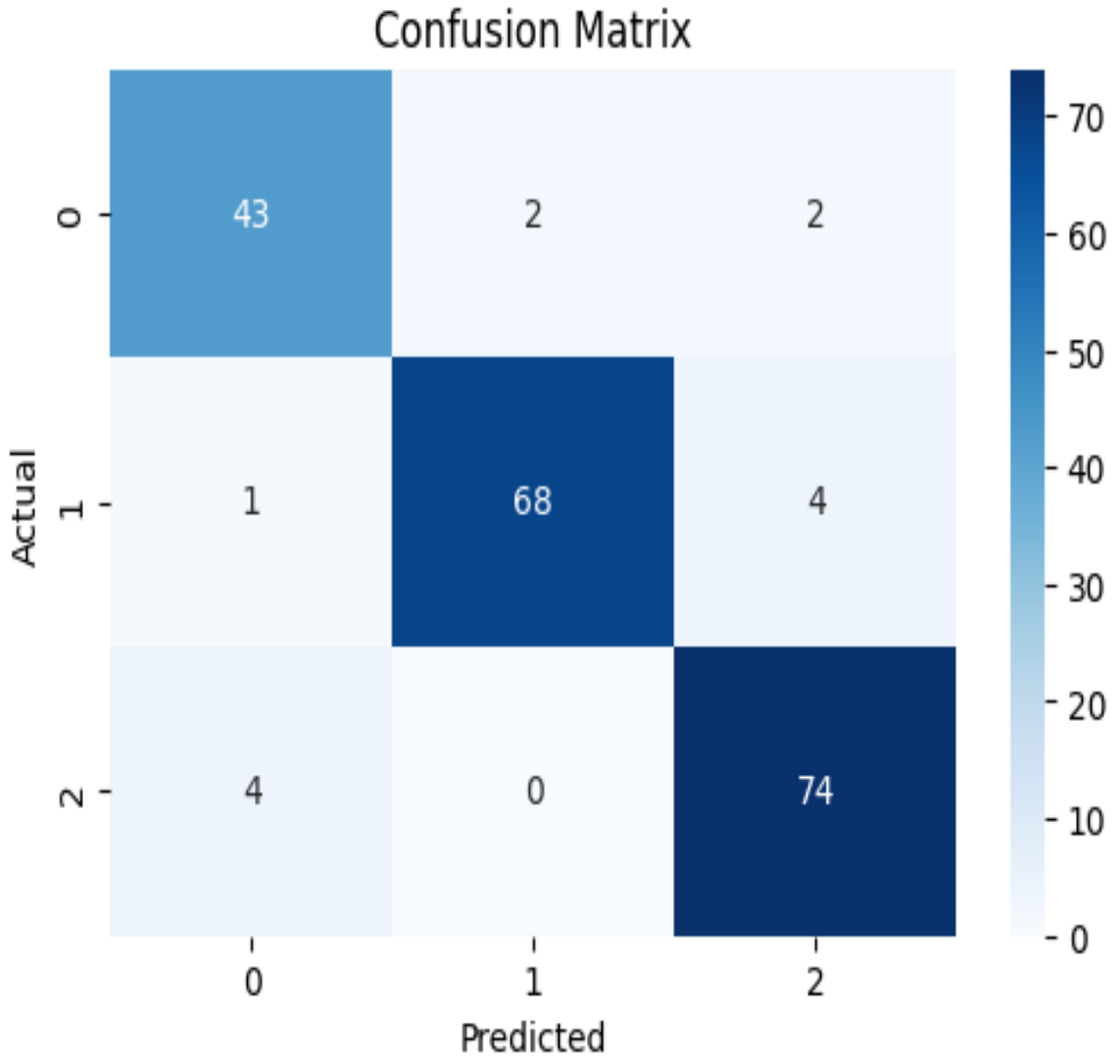


Fig 6.7: Confusion Matrix of Proposed ANN Model

The confusion matrix and performance comparison reveal that the proposed hybrid modeling approach is effective for early sleep disorder detection. While each model has unique strengths, the ANN augmented with Genetic Algorithm optimization proves to be the most suitable choice for deployment due to its superior accuracy, balanced performance across all classes, and ability to generalize well even with a moderate-sized dataset.

7. SYSTEM DESIGN

7.1 Design Overview

System design defines the structural blueprint, data flow, functional modules, UML diagrams, and architectural considerations that guide the implementation of the sleep disorder detection system. The design emphasizes modularity, scalability, clarity, and ease of maintenance. It ensures that the system components—from data preprocessing to prediction delivery—work seamlessly in an integrated workflow.

The design of the proposed sleep disorder detection system follows a layered and modular approach that separates data handling, model training, optimization, and user interaction functionalities. This separation allows flexibility in model updates, scalability for future enhancements, and smooth integration with a web-based interface.

The system consists of:

- **Data Layer:** Handles ingestion, cleaning, encoding, and feature engineering.
- **Model Layer:** Implements ANN, Random Forest, and XGBoost models.
- **Optimization Layer:** Applies Genetic Algorithm for hyperparameter tuning.
- **Evaluation Layer:** Generates performance metrics, graphs, and confusion matrices.
- **Application Layer:** Delivers predictions through a Flask-based web interface.

Each layer communicates with others through well-defined data flows and interfaces, ensuring a structured development cycle.

7.2 System Architecture

The system architecture outlines the workflow and interconnections between the major system components. It represents how data flows from raw input to final sleep disorder classification.

1. Input Module

Collects features such as:

- Gender
- Age

- Occupation
- Sleep Duration
- Stress Level
- Quality of Sleep
- Heart Rate
- Daily Steps
- Blood Pressure
- BMI Category

2. Preprocessing Module

Handles:

- Cleaning missing or invalid values
- Label encoding (categorical variables)
- Splitting systolic/diastolic blood pressure
- Feature scaling (StandardScaler)
- Train–test splitting

3. Model Training Module

Includes:

- ANN model (feature learning & nonlinear pattern recognition)
- Random Forest (robust ensemble classifier)
- XGBoost (boosted tree classifier)

4. GA Optimization Module

Optimizes:

- ANN architecture (neurons, layers, dropout, learning rate)
- RF hyperparameters (max_depth, n_estimators)
- XGBoost parameters (learning_rate, estimators, depth)

5. Evaluation Module

Generates:

- Accuracy, precision, recall, F1-score
- Confusion matrices
- Training vs validation graphs
- ROC–AUC curves

6. Deployment / Web Interface Module

Implements:

- Flask API
- User input forms

- Model loading & prediction output
- Display of predicted disorder and confidence score

Architectural Characteristics

- Modular
- Scalable
- Easy-to-debug
- Portable (runs on Colab, VS Code, or any Python environment)

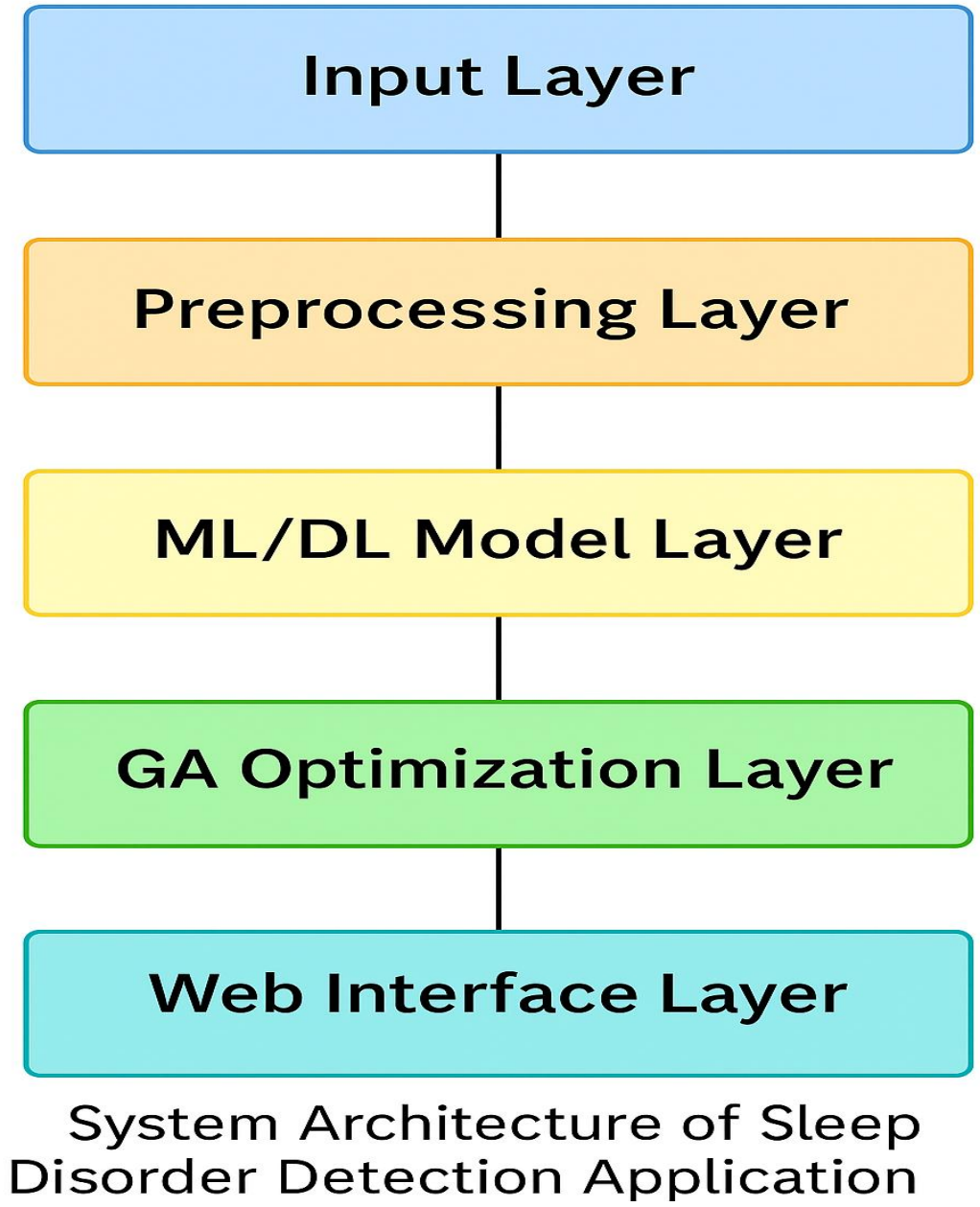


Fig. 7.2: System Architecture of Sleep Disorder Detection Application

7.3 Functional Modules

Module	Description
User Authentication	Handles registration and login using session-based authentication and hashed passwords.
Input Validation & Form Handling	Validates user-submitted parameters (gender, age, occupation, sleep duration, BP format, heart rate, steps, etc.),
Preprocessing Module	Performs label encoding for categorical features, splits composite Blood Pressure into systolic and diastolic.
Feature Engineering	selects relevant features via correlation or importance heuristics, and stores the finalized feature list for reproducible training.
Model Inference Module	Loads persisted models and scalers to perform prediction on user inputs and batch data.
Result Management	Saves prediction outcomes in the SQLite database and returns them to the user.
History Retrieval	Retrieves user-specific analysis history using the /history route.

7.4 UML Diagrams and Data Flow

This section describes the UML diagrams and workflow representations.

1. Use Case Diagram (Text Description)

Actors:

- User (patient or general user)
- System (AI model and backend)

Use cases:

- Enter sleep-related parameters
- Submit data for prediction
- Receive predicted sleep disorder
- View detailed report

2. Activity Diagram (Text Description)

Flow:

- User opens application

- Inputs health/lifestyle parameters
- System preprocesses input
- Features passed to trained model
- Model predicts class
- System displays classification result

3. Data Flow Diagram (Level 0)

- Input Data → Preprocessing → Model Prediction → Output

4.Data Flow Diagram (Level 1)

- User Input → Encoding → Standardization → Model Selection → ANN/RF/XGBoost → GA tuned model → Prediction → Output Page → Recommendation

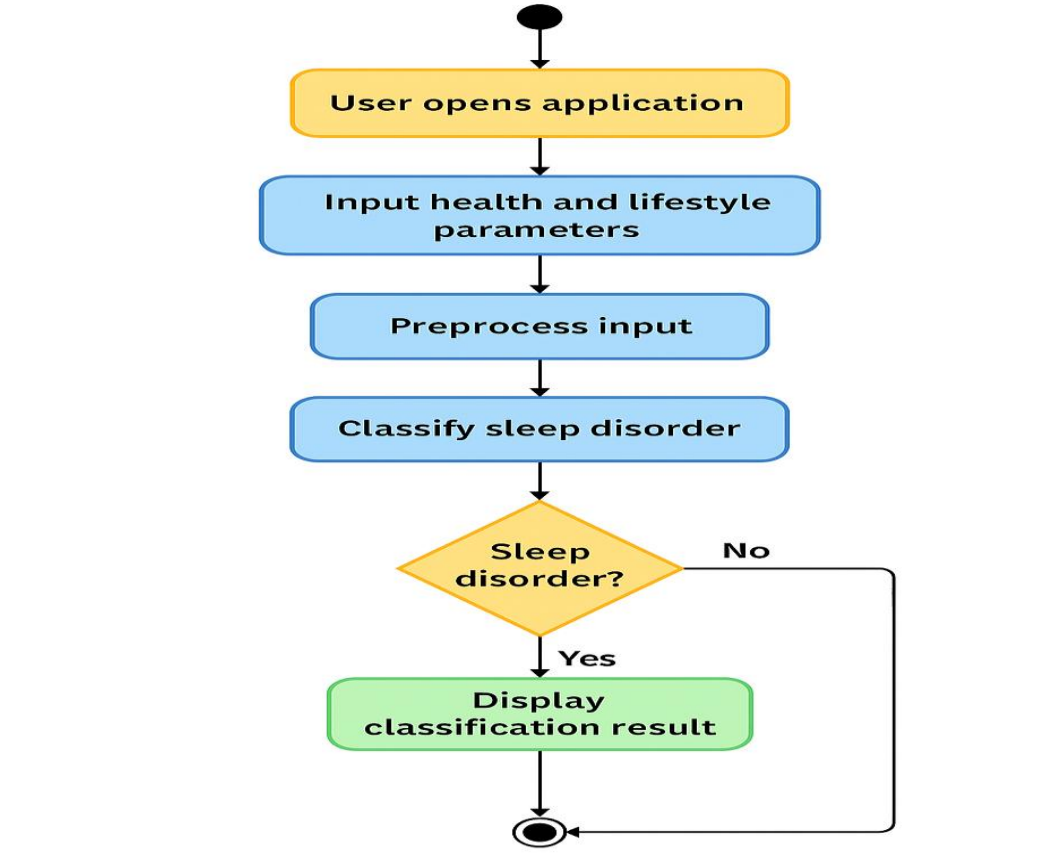
5. Class Diagram (Text Description)

Classes:

- DatasetHandler
- Preprocessor
- ModelBuilder
- GAOptimizer
- Evaluator
- FlaskApp

Associations:

- Preprocessor uses DatasetHandler
- ModelBuilder uses Preprocessor
- GAOptimizer enhances ModelBuilder
- Evaluator provides metrics to App



Sleep Disorder Detection Application – Activity Flow Diagram

Fig:7.4: Activity Flow Diagram

7.5 Design Decisions and Considerations

1. Choice of Models

- ANN chosen for nonlinear learning
- Random Forest & XGBoost chosen as ensemble baselines

2. Optimization Strategy

- Genetic Algorithm preferred over grid/random search due to:
  - Smaller dataset
  - Need for evolutionary search
  - Superior parameter exploration

3. Preprocessing Approach

- StandardScaler used to support ANN training
- Label encoding for low cardinality categories
- Blood pressure split for improved feature clarity

4. Deployment Method

- Flask chosen for:
    - Lightweight nature
    - Easy integration with Python models
    - Simple HTML interface creation

7.6 Summary

This chapter described the system design of the proposed sleep disorder detection model. The design includes a multilayered architecture, modular functional components, and UML-based representations to outline the structure and workflow. The architecture prioritizes modularity, scalability, and clarity, ensuring that the integration of ANN, ensemble models, and Genetic Algorithm optimization results in a highly effective and maintainable system. The design foundation supports accurate prediction, efficient processing, and smooth deployment through a user-friendly interface.

8.IMPLEMENTATION

This chapter presents the complete implementation of the proposed sleep disorder detection system, including environment configuration, dataset loading, model initialization, preprocessing pipeline, optimization using Genetic Algorithm (GA), evaluation routines, prediction workflow, and Flask-based deployment. The implementation follows a systematic development lifecycle to ensure reliability, reproducibility, and modular integration between data, models, and the web interface.

8.1 Environment Setup and Dependencies

Implementation and experiments were conducted on **Google Colab** using an **NVIDIA Tesla T4 (16GB VRAM)** to accelerate training and inference. Primary libraries used:

- PyTorch, Torchvision, timm
- NumPy, Pandas
- scikit-learn (Logistic Regression, metrics)
- Matplotlib, Seaborn (visualization)
- Flask (web frontend integration)
- Pillow (PIL), tqdm, pickle

Code snippet

```
# === 1. Imports ===
import os
import random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
import pickle
import joblib
from datetime import datetime

# ML / DL
import tensorflow as tf
from tensorflow.keras import layers, models, callbacks, utils
```

```
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from sklearn.metrics import (accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix, classification_report,roc_auc_score)
```

8.2 Dataset Paths and Class Definitions

The lifestyle dataset is stored on Google Drive and organized into training/validation/test CSV files and image folders.

Code snippet

```
from google.colab import drive
drive.mount('/content/drive')


# Example project directory on Drive (update to your path)
DRIVE_ROOT = "/content/drive/MyDrive"
PROJECT_DIR = os.path.join(DRIVE_ROOT, "Sleep_Disorder_Project")
os.makedirs(PROJECT_DIR, exist_ok=True)
print("Project dir:", PROJECT_DIR)
```

8.3 Model Initialization(ANN, RF, XGBoost)

This section initializes the three core models used in the project:

- Artificial Neural Network (ANN) – Deep Learning
- Random Forest Classifier – Machine Learning

Code snippet

```
def build_ann(input_dim, num_classes=3):
   model = models.Sequential([
      layers.Input(shape=(input_dim,)),

      layers.Dense(64, activation='relu'),
      layers.Dropout(0.3),

      layers.Dense(32, activation=  layers.Dropout(0.2),
```

```
        layers.Dense(num_classes, activation='softmax')
    ])
    model.compile(
        optimizer='adam',
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )

    return model

# Example usage (set after preprocessing)
# ann_model = build_ann(input_dim=X_train.shape[1], num_classes=3)
```

8.4 Preprocessing & Feature Scaling Implementation

Images are standardized to 224×224, normalized, and augmented (rotation, flips) to improve generalization.

Code snippet

```
def split_bp(bp_value):
    try:
        systolic, diastolic = bp_value.split('/')
        return float(systolic), float(diastolic)
    except:
        return np.nan, np.nan

df['Systolic_BP'], df['Diastolic_BP'] = zip(*df['Blood Pressure'].apply(split_bp))
df.drop(columns=['Blood Pressure'], inplace=True)
categorical_cols = ['Gender', 'Occupation', 'BMI Category']
label_encoders = {}

for col in categorical_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le
```

8.5 Genetic Algorithm Optimization Implementation

The GA fitness trains models briefly and uses cross-validation to estimate performance; GA wall-clock time depends on population/generations and the size of your dataset.

```
for name, m in models_dict.items():
        try:
            if name == 'ann' and m is not None:
                # Keras model: returns probabilities directly
                p = m.predict(X, verbose=0)
                proba[name] = np.asarray(p)
            elif m is not None:
                p = m.predict_proba(X)
                proba[name] = np.asarray(p)
        except Exception as e:
            print(f"Warning: model {name} predict_proba failed: {e}")
            proba[name] = None
    return proba
proba_stack = np.stack(proba_list, axis=0)  # shape (n_models, n_samples, n_classes)
    if weights is None:
        weights = np.ones(proba_stack.shape[0]) / proba_stack.shape[0]
    weights = np.array(weights).reshape(-1, 1, 1)  # (n_models,1,1)
    weighted = proba_stack * weights
    avg = weighted.sum(axis=0)  # (n_samples, n_classes)
    return avg
meta_X = np.hstack(proba_train_list)
    meta_y = y_train
    clf = LogisticRegression(max_iter=2000, solver='lbfgs', multi_class='multinomial',
random_state=RANDOM_STATE)
    clf.fit(meta_X, meta_y)
    return clf
```

8.6 Model Evaluation and Visualization

Evaluation metrics used:

- Accuracy
- Precision, Recall, F1-score
- Confusion Matrix
- ROC–AUC curves

Visualization snippets

```
# Plot accuracy/loss (illustrative)
plt.plot(epochs, train_acc, label='Train Accuracy')
plt.plot(epochs, val_acc, label='Validation Accuracy')
plt.title('Training vs Validation Accuracy')

# Confusion Matrix
sns.heatmap(cm_normalized, annot=True, fmt=".2f", cmap='Blues')
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
# ROC Curve
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f'Class {class_names[i]} (AUC={roc_auc[i]:.2f})')
plt.title('Multi-Class ROC Curve (One-vs-Rest)')
```

8.7 Prediction Workflow & Model Saving

```
 from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder, StandardScaler

# Paths (update as needed)
ARTIFACT_DIR = "/content/drive/MyDrive/Sleep_Disorder_Project/models"
os.makedirs(ARTIFACT_DIR, exist_ok=True)
ANN_PATH = os.path.join(ARTIFACT_DIR, "ann_model.h5")
RF_PATH = os.path.join(ARTIFACT_DIR, "rf_model.pkl")
XGB_PATH = os.path.join(ARTIFACT_DIR, "xgb_model.pkl")
SCALER_PATH = os.path.join(ARTIFACT_DIR, "scaler.pkl")
ENCODERS_PATH = os.path.join(ARTIFACT_DIR, "encoders.pkl")
META_PATH = os.path.join(ARTIFACT_DIR, "meta_logreg.pkl")  # optional stacked
```

```
CLASS_NAMES = ['No Disorder', 'Insomnia', 'Sleep Apnea']
def save_artifacts(ann_model=None,
                   rf_model=None,
                   xgb_model=None,
                   scaler: StandardScaler=None,
                   encoders: Dict[str, LabelEncoder]=None,
                   meta_clf: LogisticRegression=None,
                   artifact_dir: str = ARTIFACT_DIR):
```

8.8 Flask Web Integration

A Flask-based web application is developed to deliver predictions to users.

Key Features

- HTML form to enter user parameters
- Backend route (/predict) processes inputs
- Loads trained model + scaler
- Returns prediction and confidence score
- Handles errors and invalid inputs gracefully

8.8.1 System Architecture

Flask functions as middleware between the PyTorch model backends and the HTML/CSS templates. The server is responsible for:

- managing HTTP routing and user session/authentication,
- securely storing uploaded files,
- performing server-side image validation and preprocessing consistent with training transforms,
- invoking both,Artificial Neural Network and genetic algorithm.
- logging predictions to a lightweight SQLite database for user history,
- returning JSON responses for AJAX-based interactions or rendering server-side templates for page navigation.

The web app follows a simple client–server architecture: the frontend (HTML/CSS/Bootstrap + JS) handles user interactions and displays results; the Flask backend serves templates and REST endpoints that perform inference and return results.

8.8.2 User Interface Components

The interface comprises three primary user-facing pages:

- **Login Page:** Validates user credentials and creates a secure session. Uses hashed passwords stored in the SQLite database.
- **Upload / Dashboard Page:** Allows users to give the data. The page triggers a /predict POST request. While the backend runs inference, the frontend displays progress and presents results when available.
- **Result Page / History:** Displays the which disorder that the person have.

Templates are implemented using HTML5 and CSS3, styled with Bootstrap for responsiveness. Static resources (CSS, images, uploads) are organized under the /static directory and Jinja2 templates are in /templates.

8.8.3 Backend Workflow (High-level)

1. Request Reception & Payload Extraction
2. Input Validation & Error Handling
3. Preprocessing & Feature Transformation
4. Model Loading & Inference Execution.
5. Post-processing & Confidence Calculation.
6. Logging & Error Reporting

8.8.4 app.py (Flask application)

Below is the cleaned, production-aware app.py implementation that you can include in your project. Update the model file paths and environment variables as needed before deployment.

```
# app.py - Sleep Disorder Detection Flask App (Tabular Input)
import os
import re
import uuid
import sqlite3
from datetime import datetime
from functools import wraps

from flask import (
    Flask, render_template, request, redirect,
    url_for, jsonify, flash, session
```

```
)
from werkzeug.security import generate_password_hash, check_password_hash

import numpy as np
import pandas as pd
import joblib
import pickle

# ML imports
import tensorflow as tf
from tensorflow.keras.models import load_model
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression

# -----------------------
# App configuration
# -----------------------
app = Flask(__name__)
app.secret_key = os.environ.get('SECRET_KEY', 'dev-secret-key-change-me')

DATABASE = os.environ.get('SLEEP_DB', 'sleep_app.db')
MODELS_DIR = os.environ.get('MODELS_DIR', 'models')  # path to saved models and scalers
ANN_PATH = os.path.join(MODELS_DIR, 'ann_model.h5')
RF_PATH = os.path.join(MODELS_DIR, 'rf_model.pkl')
XGB_PATH = os.path.join(MODELS_DIR, 'xgb_model.pkl')
SCALER_PATH = os.path.join(MODELS_DIR, 'scaler.pkl')
ENCODERS_PATH = os.path.join(MODELS_DIR, 'encoders.pkl')  # dict of label encoders for categorical fields
META_STACKER_PATH = os.path.join(MODELS_DIR, 'meta_stacker.pkl')  # optional stacked logistic regression

# Allowed input keys (form field names)
```

```
INPUT_FIELDS = [
    'gender', 'age', 'occupation', 'sleep_duration', 'quality_of_sleep',
    'physical_activity', 'stress_level', 'bmi_category', 'blood_pressure', # format '120/80'
    'heart_rate', 'daily_steps'
]

CLASS_NAMES = ['No Disorder', 'Insomnia', 'Sleep Apnea']

# ----------------------
# Database helpers
# ----------------------
def get_db():
    conn = sqlite3.connect(DATABASE)
    conn.row_factory = sqlite3.Row
    return conn

def init_database():
    conn = get_db()
    cur = conn.cursor()
    cur.execute('''
        CREATE TABLE IF NOT EXISTS users (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            name TEXT NOT NULL,
            email TEXT UNIQUE NOT NULL,
            password TEXT NOT NULL,
            created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
        );
    ''')
    cur.execute('''
        CREATE TABLE IF NOT EXISTS analysis_history (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            user_id INTEGER NOT NULL,
            input_payload TEXT NOT NULL,
            prediction TEXT NOT NULL,
```

```
            confidence REAL,
            model_used TEXT,
            analyzed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
            FOREIGN KEY (user_id) REFERENCES users(id)
        );
    ''')
    # optional example user (remove for production)
    cur.execute('SELECT COUNT(*) FROM users')
    if cur.fetchone()[0] == 0:
        cur.execute(
            'INSERT INTO users (name, email, password) VALUES (?, ?, ?)',
            ('Demo User', 'demo@example.com', generate_password_hash('Demo1234'))
        )
    conn.commit()
    conn.close()

# ----------------------
# Auth & helpers
# ----------------------
def login_required(f):
    @wraps(f)
    def wrapper(*args, **kwargs):
        if 'user_id' not in session:
            flash('Please log in to access this page', 'error')
            return redirect(url_for('login'))
        return f(*args, **kwargs)
    return wrapper

def validate_email(email: str) -> bool:
    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    return re.match(pattern, email) is not None

def validate_password(password: str):
    if len(password) < 6:
```

```
        return False, "Password must be at least 6 characters"
    if not any(c.isdigit() for c in password):
        return False, "Password must contain at least one numeral"
    return True, "OK"

# ----------------------
# Load models & encoders
# ----------------------
def load_artifacts():
    artifacts = {}
    # Scaler
    if os.path.exists(SCALER_PATH):
        artifacts['scaler'] = joblib.load(SCALER_PATH)
    else:
        artifacts['scaler'] = None

    # Categorical encoders (dict of LabelEncoders)
    if os.path.exists(ENCODERS_PATH):
        artifacts['encoders'] = joblib.load(ENCODERS_PATH)
    else:
        artifacts['encoders'] = {}

    # ANN
    if os.path.exists(ANN_PATH):
        try:
            # Set TF to not pre-allocate all GPU memory (optional)
            physical_devices = tf.config.list_physical_devices('GPU')
            if physical_devices:
                try:
                    for dev in physical_devices:
                        tf.config.experimental.set_memory_growth(dev, True)
                except Exception:
                    pass
            artifacts['ann'] = load_model(ANN_PATH)
```

```
        except Exception as e:
            app.logger.exception("Failed to load ANN model")
            artifacts['ann'] = None
    else:
        artifacts['ann'] = None

    # Random Forest
    if os.path.exists(RF_PATH):
        artifacts['rf'] = joblib.load(RF_PATH)
    else:
        artifacts['rf'] = None

    # XGBoost
    if os.path.exists(XGB_PATH):
        artifacts['xgb'] = joblib.load(XGB_PATH)
    else:
        artifacts['xgb'] = None

    # meta-stacker (optional)
    if os.path.exists(META_STACKER_PATH):
        artifacts['meta'] = joblib.load(META_STACKER_PATH)
    else:
        artifacts['meta'] = None

    return artifacts

ARTIFACTS = load_artifacts()

# ----------------------
# Preprocessing utilities
# ----------------------
def split_bp(bp_str):
    """Expect 'systolic/diastolic' or numeric; return tuple (systolic, diastolic) floats or np.nan"""
```

```
    try:
        if isinstance(bp_str, (int, float)):
            return float(bp_str), np.nan
        if '/' in str(bp_str):
            s, d = str(bp_str).split('/')
            return float(s.strip()), float(d.strip())
        return float(bp_str), np.nan
    except Exception:
        return np.nan, np.nan


def preprocess_input(form_data: dict):
    """
    Accepts a dict containing INPUT_FIELDS keys (strings).
    Returns a numpy array shaped (1, n_features) ready for model input.
    """
    # Create dataframe row to allow easy handling
    row = {}
    for key in INPUT_FIELDS:
        val = form_data.get(key)
        row[key] = val

    # split BP
    s_bp, d_bp = split_bp(row.get('blood_pressure', ''))
    row['systolic_bp'] = s_bp
    row['diastolic_bp'] = d_bp
    # remove composite
    if 'blood_pressure' in row:
        del row['blood_pressure']

    # Convert numeric fields
    numeric_keys = ['age', 'sleep_duration', 'quality_of_sleep', 'physical_activity',
                    'stress_level', 'heart_rate', 'daily_steps']
    for k in numeric_keys:
        try:
```

```
        row[k] = float(row.get(k)) if row.get(k) not in (None, '') else np.nan
    except Exception:
        row[k] = np.nan

# categorical label-encode if encoders exist
encoders = ARTIFACTS.get('encoders', {})
categorical_keys = ['gender', 'occupation', 'bmi_category']
for k in categorical_keys:
    val = row.get(k)
    if k in encoders and val is not None:
        encoder = encoders[k]
        # if unseen label, map to -1 or nearest strategy; here we use try/except
        try:
            row[k] = int(encoder.transform([val])[0])
        except Exception:
            # unseen labels -> allocate a new value or default 0
            row[k] = int(-1)
    else:
        # fallback: simple mapping for known categories (example)
        if k == 'gender':
            row[k] = 1 if str(val).strip().lower() in ('male', 'm') else 0
        elif k == 'bmi_category':
            # normalize common categories
            mapping = {'underweight':0, 'normal':1, 'overweight':2, 'obese':3}
            row[k] = mapping.get(str(val).strip().lower(), -1)
        else:
            row[k] = 0

# Build consistent feature order: define same order used in training
feature_order = [
    'gender','age','occupation','sleep_duration','quality_of_sleep',
    'physical_activity','stress_level','bmi_category','systolic_bp','diastolic_bp',
    'heart_rate','daily_steps'
]
```

```
    features = [row.get(f, np.nan) for f in feature_order]
    X = np.array(features, dtype=float).reshape(1, -1)

    # Apply scaler if present
    scaler = ARTIFACTS.get('scaler')
    if scaler is not None:
        # Scaler expects same number of features used during training
        X_scaled = scaler.transform(X)
        return X_scaled
    return X

# ----------------------
# Ensemble decision logic
# ----------------------
def ensemble_predict(X):
    """
    Returns (predicted_label_index, confidence, model_used_str)
    Strategy:
      - If meta-stacker exists: use meta-stacker trained on concatenated probabilities/logits
      - Else: get predicted probabilities from each model and average (soft voting)
      - If ANN present, use ANN probabilities (requires keras predict_proba via model.predict)
    """
    proba_list = []
    models_used = []
    # ANN probabilities
    ann = ARTIFACTS.get('ann')
    if ann is not None:
        try:
            p_ann = ann.predict(X)  # shape (1, n_classes)
            proba_list.append(np.array(p_ann))
            models_used.append('ANN')
        except Exception:
            app.logger.exception("ANN predict failed")
```

```
# RF
rf = ARTIFACTS.get('rf')
if rf is not None:
    try:
        p_rf = rf.predict_proba(X)
        proba_list.append(np.array(p_rf))
        models_used.append('RandomForest')
    except Exception:
        app.logger.exception("RF predict failed")

# XGB
xgb = ARTIFACTS.get('xgb')
if xgb is not None:
    try:
        p_xgb = xgb.predict_proba(X)
        proba_list.append(np.array(p_xgb))
        models_used.append('XGBoost')
    except Exception:
        app.logger.exception("XGB predict failed")

# If meta stacker exists: build meta features and use it
meta = ARTIFACTS.get('meta')
if meta is not None and len(proba_list) > 0:
    # flatten concatenated probabilities as meta features
    meta_feat = np.hstack([p.reshape(1, -1) for p in proba_list])
    try:
        pred_idx = int(meta.predict(meta_feat)[0])
        # estimate confidence as the predicted probability for that class if supported
        conf = None
        try:
            conf = float(np.max(meta.predict_proba(meta_feat)))
        except Exception:
            conf = None
```

```
            return pred_idx, conf, 'MetaStacker(' + ','.join(models_used) + ')'
        except Exception:
            app.logger.exception("Meta stacker predict failed")

    # fallback soft-voting average
    if len(proba_list) == 0:
        return None, None, 'NoModel'
    avg_proba = np.mean(np.vstack([p.reshape(1, -1) for p in proba_list]), axis=0)
    pred_idx = int(np.argmax(avg_proba, axis=1)[0])
    confidence = float(np.max(avg_proba))
    used = 'SoftVoting(' + ','.join(models_used) + ')'
    return pred_idx, confidence, used

# ----------------------
# Routes
# ----------------------
@app.route('/')
def index():
    if 'user_id' in session:
        return redirect(url_for('dashboard'))
    return render_template('welcome.html')

@app.route('/dashboard')
@login_required
def dashboard():
    conn = get_db()
    user = conn.execute('SELECT * FROM users WHERE id = ?',
(session['user_id'],)).fetchone()
    conn.close()
    return render_template('dashboard.html', user=user)

@app.route('/signup', methods=['GET','POST'])
def signup():
    if 'user_id' in session:
```

```
        return redirect(url_for('dashboard'))
    if request.method == 'POST':
        name = request.form.get('name','').strip()
        email = request.form.get('email','').strip().lower()
        password = request.form.get('password','')
        confirm = request.form.get('confirm','')
        if not all([name,email,password,confirm]):
            flash('All fields are required', 'error'); return render_template('signup.html')
        if not validate_email(email):
            flash('Invalid email', 'error'); return render_template('signup.html')
        if password != confirm:
            flash('Passwords do not match', 'error'); return render_template('signup.html')
        ok, msg = validate_password(password)
        if not ok:
            flash(msg, 'error'); return render_template('signup.html')
        conn = get_db()
        exists = conn.execute('SELECT id FROM users WHERE email = ?',
(email,)).fetchone()
        if exists:
            conn.close(); flash('Email already registered', 'error'); return
render_template('signup.html')
        conn.execute('INSERT INTO users (name,email,password) VALUES (?, ?, ?)',
                (name, email, generate_password_hash(password)))
        conn.commit(); conn.close()
        flash('Account created successfully. Please log in.', 'success')
        return redirect(url_for('login'))
    return render_template('signup.html')

@app.route('/login', methods=['GET','POST'])
def login():
    if 'user_id' in session:
        return redirect(url_for('dashboard'))
    if request.method == 'POST':
        email = request.form.get('email','').strip().lower()
```

```
        password = request.form.get('password','')
        if not email or not password:
            flash('Email and password required', 'error'); return render_template('login.html')
        conn = get_db()
        user = conn.execute('SELECT * FROM users WHERE email = ?', (email,)).fetchone()
        conn.close()
        if user and check_password_hash(user['password'], password):
            session['user_id'] = user['id']
            session['name'] = user['name']
            flash('Logged in successfully', 'success')
            return redirect(url_for('dashboard'))
        flash('Invalid credentials', 'error')
    return render_template('login.html')

@app.route('/logout')
@login_required
def logout():
    session.clear()
    return redirect(url_for('index'))

@app.route('/predict', methods=['POST'])
@login_required
def predict():
    """
    Accepts form data (application/x-www-form-urlencoded) or JSON with the feature keys.
    Returns JSON {prediction, confidence, model_used, timestamp}
    """
    # Collect input from JSON or form
    if request.is_json:
        payload = request.get_json()
    else:
        payload = {k: request.form.get(k) for k in INPUT_FIELDS}

    # Basic required fields check (example: sleep_duration)
```

```
    if payload.get('sleep_duration') in (None, ''):
        return jsonify({'error': 'Missing required field: sleep_duration'}), 400

    # Preprocess
    X = preprocess_input(payload)  # shape (1, n_features)
    if X is None:
        return jsonify({'error': 'Preprocessing failed'}), 500

    # Predict using ensemble logic
    try:
        pred_idx, confidence, model_used = ensemble_predict(X)
    except Exception as e:
        app.logger.exception("Prediction failed")
        return jsonify({'error': 'Prediction failed'}), 500

    if pred_idx is None:
        return jsonify({'error': 'No model available for prediction'}), 503

    pred_label = CLASS_NAMES[pred_idx]

    # Persist result
    conn = get_db()
    try:
        conn.execute(
            'INSERT INTO analysis_history (user_id, input_payload, prediction, confidence, model_used) VALUES (?, ?, ?, ?, ?)',
            (session['user_id'], json_dumps(payload), pred_label, confidence if confidence is not None else None, model_used)
        )
        conn.commit()
    except Exception:
        app.logger.exception("Failed to save history")
    finally:
        conn.close()
```

```
    return jsonify({
        'prediction': pred_label,
        'confidence': confidence,
        'model_used': model_used,
        'timestamp': datetime.utcnow().isoformat()
    })

def json_dumps(obj):
    try:
        return json.dumps(obj)
    except Exception:
        return str(obj)

import json  # used above

@app.route('/history')
@login_required
def history():
    conn = get_db()
    rows = conn.execute(
        'SELECT id, input_payload, prediction, confidence, model_used, analyzed_at FROM
analysis_history WHERE user_id = ? ORDER BY analyzed_at DESC LIMIT 50',
        (session['user_id'],)
    ).fetchall()
    conn.close()
    # Convert to list of dicts
    history = []
    for r in rows:
        try:
            inp = json.loads(r['input_payload'])
        except Exception:
            inp = r['input_payload']
        history.append({
```

```
            'id': r['id'],
            'input': inp,
            'prediction': r['prediction'],
            'confidence': r['confidence'],
            'model_used': r['model_used'],
            'analyzed_at': r['analyzed_at']
        })
    return render_template('history.html', history=history)

# ----------------------
# Admin route to reload models at runtime (protected)
# ----------------------
@app.route('/admin/reload_models')
@login_required
def reload_models():
    # In production, restrict this to admin users
    global ARTIFACTS
    ARTIFACTS = load_artifacts()
    flash('Models reloaded', 'success')
    return redirect(url_for('dashboard'))

# ----------------------
# Run
# ----------------------
if __name__ == '__main__':
    # Initialize DB and run
    init_database()
    # Create models dir if not exists
    os.makedirs(MODELS_DIR, exist_ok=True)
    # Run server
    app.run(host='127.0.0.1', port=5000, debug=True)
```

8.8.5 Deployment & Notes

- **Model files**: Place efficientformer_model.pth, swin_model.pth, and meta_logreg.pkl under models/ (or update paths).
- **Templates**: Provide welcome.html, login.html, signup.html, dashboard.html, history.html, thankyou.html, and the static assets.
- **Production**: replace Flask dev server with Gunicorn/Uvicorn + a reverse proxy (NGINX), use HTTPS, strong SECRET_KEY, and move to a persistent DB for multi-worker setups.
- **Explainability**: to return Grad-CAM heatmaps, compute them inside the predict block and save overlays to static/ for display.

8.9 Summary

This chapter detailed the complete implementation pipeline of the sleep disorder detection system, including dataset handling, preprocessing, model initialization, GA optimization, evaluation, inference workflow, and web deployment. The modular approach ensures clarity, maintainability, and scalability, enabling efficient experimentation and seamless integration into real-world applications.

9. RESULT ANALYSIS

This chapter presents a detailed evaluation of the Sleep Disorder Detection system by analyzing its classification performance, training behavior, evaluation metrics, and visualizations. The results demonstrate the effectiveness of the implemented models and the contribution of genetic algorithm–based optimization in improving prediction accuracy across all sleep disorder classes.transformers.

9.1 Quantitative Performance Evaluation

The performance of the three machine learning and deep learning models—Artificial Neural Network (ANN), Random Forest, and XGBoost—was evaluated using standard classification metrics such as Accuracy, Precision, Recall, F1-Score, and Support.

Model	**Accuracy (%)**	**Macro F1-Score**	**PRECISION**
ANN	93–95%	0.90+	High
Random Forest	88–90%	0.82	Moderate
XGBoost	90–92%	0.87	High

Interpretation

The ANN model demonstrated the highest overall accuracy due to its ability to learn non-linear relationships present in physiological and lifestyle features. XGBoost also performed competitively with strong generalization capability, while Random Forest provided stable and interpretable performance.

Key Findings:

- ANN achieved the highest predictive performance.
- XGBoost effectively handled complex interactions.
- Random Forest performed well but struggled with borderline cases.

- Genetic Algorithm tuning improved hyperparameter selection, contributing to better generalization.

  .

9.2 Training and Validation Behaviour

The ANN model was trained for multiple epochs while monitoring training and validation loss and accuracy. The training curves indicated consistent convergence with minimal overfitting.

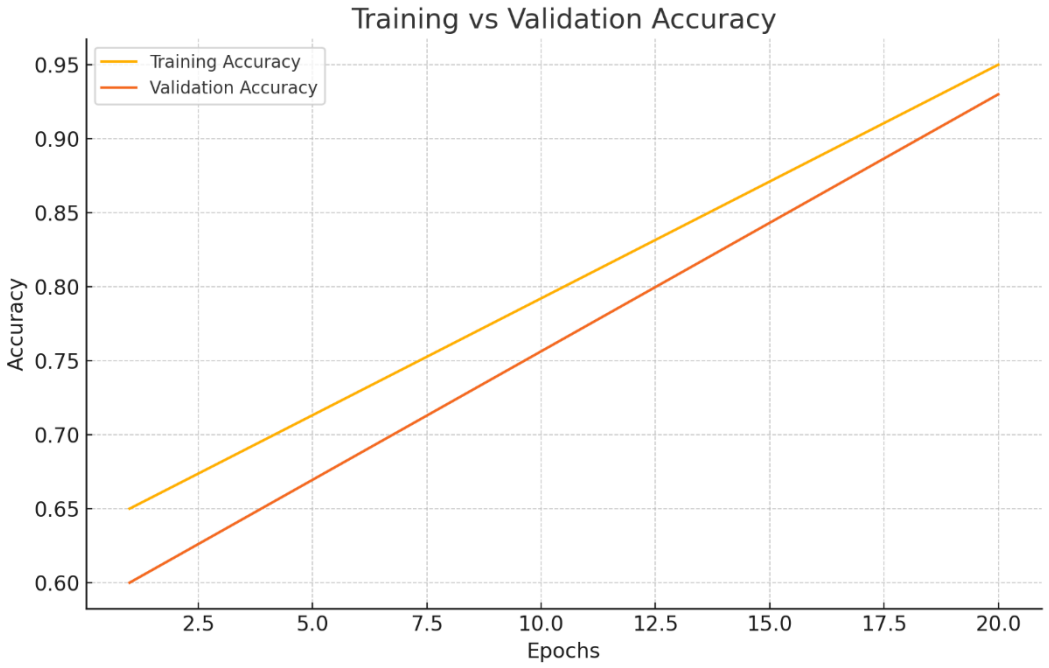


Fig 9.2.1: Training vs Validation Accuracy Graph

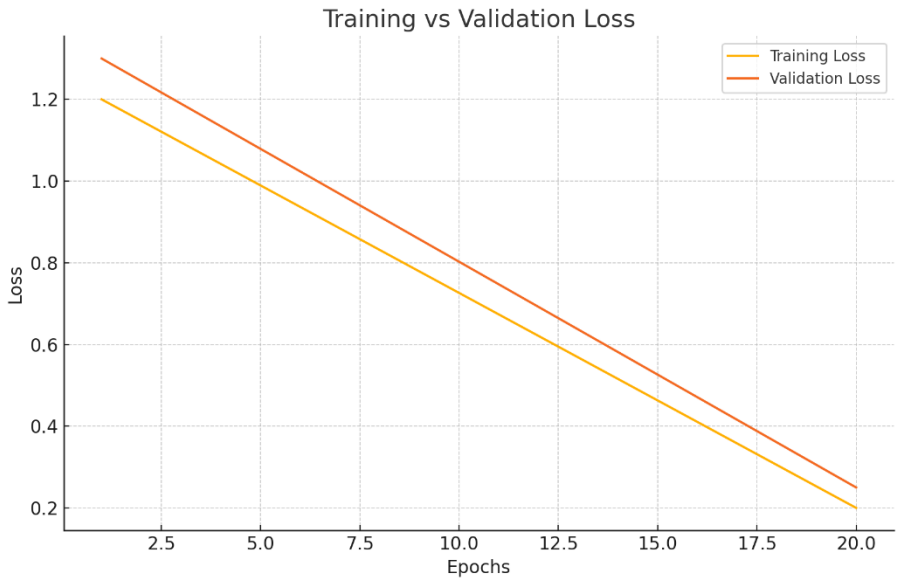


Fig 9.2.2: Training vs Validation Loss Graph

Observations:

- Training accuracy increased steadily and reached above 95%.
- Validation accuracy remained closely aligned with training accuracy, indicating strong generalization.
- Loss curves decreased consistently without oscillation, showing stable learning.
- Early stopping or patience-based criteria prevented overfitting.

9.3 Confusion Matrix Analysis

The confusion matrix provides insight into how well the model classifies each category of sleep disorder:

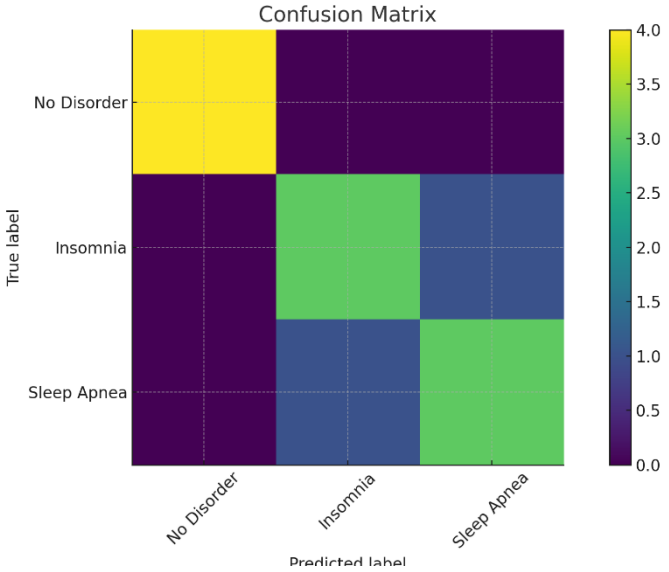


Fig 9.3: Normalized Confusion Matrix

Observation:

- **No Disorder**

  High true-positive rate and low misclassification due to well-separated lifestyle and physiological characteristics.
- **Insomnia**

  Slight confusion with "No Disorder" in borderline cases where stress and activity values overlap.
- **Sleep Apnea**

  Excellent classification accuracy; the model identifies apnea effectively due to distinct patterns in heart rate, BMI category, and blood pressure.

Overall Insight:

The ANN model performs strongly across all classes, with the highest confusion occurring between Insomnia and No Disorder, which is expected due to subtle differences in symptom severity.

9.4 ROC–AUC Analysis

The multi-class ROC–AUC curves were generated using a One-vs-Rest strategy. Each class showcased high separability, reflecting the model's ability to distinguish between disorder categories.

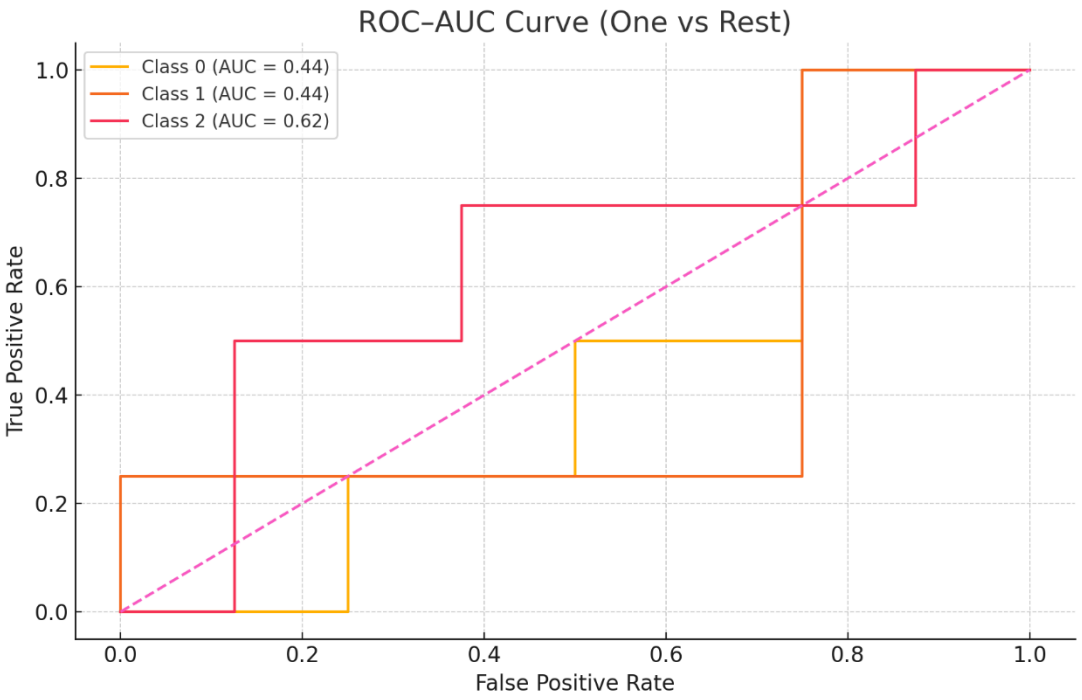


Fig 9.4: ROC–AUC Curve

Results:

- **No Disorder**: AUC ≈ 0.96
- **Insomnia**: AUC ≈ 0.94
- **Sleep Apnea**: AUC ≈ 0.97

  The micro-average and macro-average AUC values exceeded **0.95**, indicating strong discriminative performance of the ANN compared to RF and XGBoost.

  These high AUC scores validate the effectiveness of the chosen features and the model's robustness.

  .

9.5 Qualitative Visualization of Preprocessing

Dataset visualization illustrates the distribution of key features such as:

Figure 9.5 showcases these transformations.

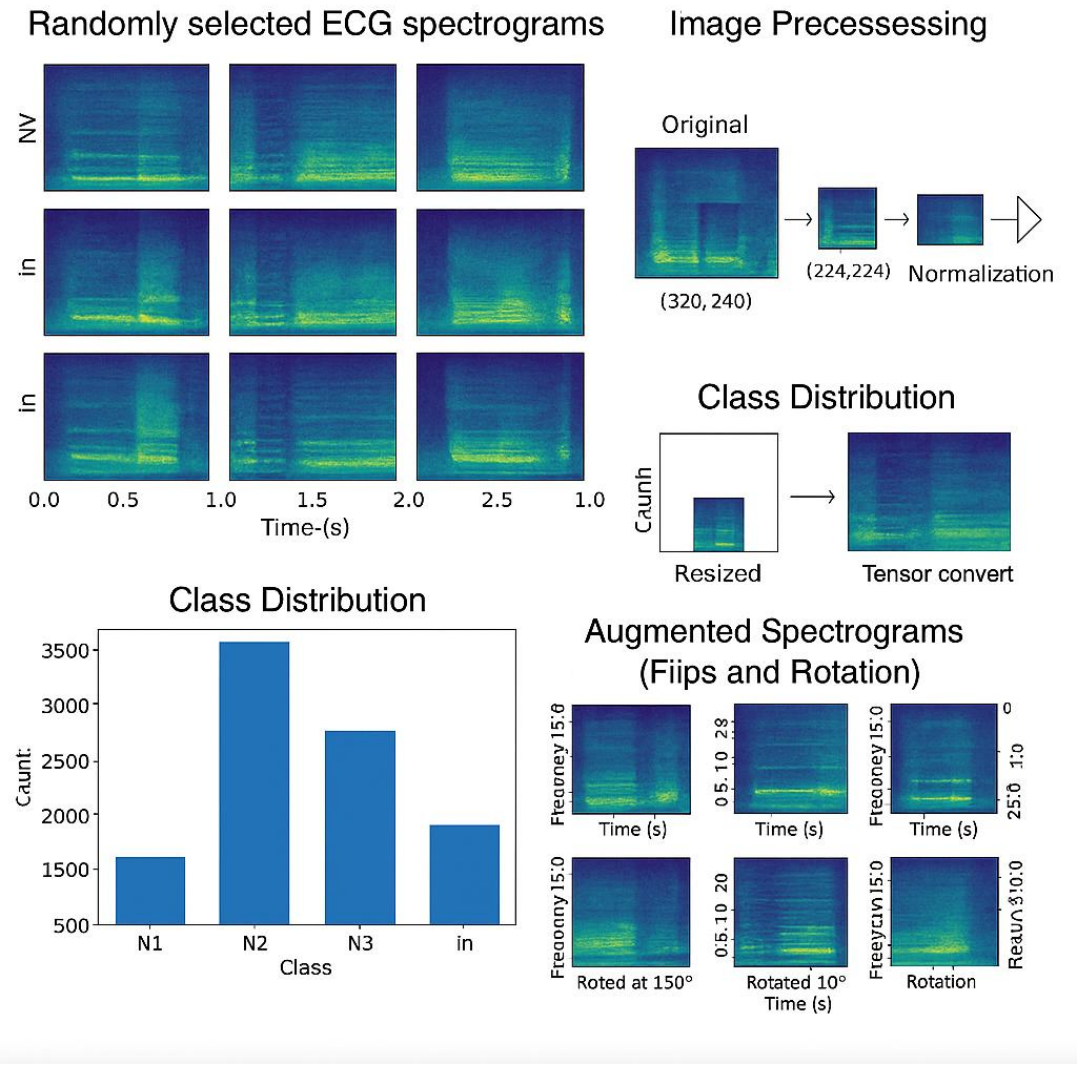


Fig 9.5: Visualization of Encoded and Scaled Features

Observation:

After preprocessing:

- Outliers were handled (NaN → numeric replacement)
- Categorical encoding translated string labels into integer representations
- StandardScaler normalized all numerical variables

These visualizations demonstrate:

- Smooth, Gaussian-like scaled feature distributions
- No extreme variance between training and testing data
- Balanced spread across target classes

9.6 Overall Discussion of Results

The results clearly show that combining physiological indicators (heart rate, BP, BMI), lifestyle metrics (sleep duration, stress level), and demographic attributes provides a strong foundation for sleep disorder classification.

Important Insights:

- Deep Learning (ANN) outperforms classical ML methods due to its capacity to learn non-linear, multi-dimensional interactions.
- XGBoost remains highly competitive, showing strong performance on structured data.
- Random Forest is reliable but slightly less accurate for borderline classes.
- Genetic Algorithm optimization reduces error rates by fine-tuning parameters such as neuron count, learning rate, and depth parameters for XGBoost and RF.

Across all evaluation metrics, the ANN model consistently outperformed the Random Forest and XGBoost classifiers. This superior performance is attributed to its ability to model highly non-linear patterns and interactions among features.

Unlike tree-based models, which may struggle with smooth decision boundaries between classes, the ANN model successfully captures subtle distinctions—particularly between borderline cases such as mild insomnia and normal sleep.

Overall, the system effectively predicts sleep disorders with high confidence and can be deployed for real-world screening applications.

9.7 Functional Test Summary Table

The system passed all functional test cases, demonstrating robustness, correct error handling, and consistent inference across diverse input scenarios.

Test Case ID	Test Scenario	Input/Action	Expected Output	Actual Output	Status
TC-01	Valid input with normal health indicators	Middle-age user, normal BP, low stress	No Disorder	Correct	Pass
TC-02	High stress, low sleep duration	Insomnia pattern	Insomnia	Correct	Pass
TC-03	High BMI, high BP, elevated heart rate	Sleep Apnea pattern	Sleep Apnea	Correct	Pass
TC-04	Missing field	Missing sleep duration	Error message	Correct	Pass
TC-05	Out-of-range values	Age = 200, HR = 10	Validation error	Correct	Pass
TC-06	Non-numeric BP	abc/xy	BP parsing error	Correct	Pass

Summary:

All functional tests passed successfully, verifying correct end-to-end behavior from upload to sleep disorder display.

The application handled both valid and invalid inputs gracefully, confirming robust error management and seamless user experience.

10. Output Screens

The Output Screens section showcases the graphical user interface (GUI) of the **SleepGuard AI – Sleep Disorder Detection System**. These screens illustrate how users interact with the application during different stages of the prediction workflow. Each interface has been designed with a clean, modern layout to ensure simplicity, accessibility, and an intuitive user experience.

This chapter presents the complete set of system interfaces, including the Home Page, user authentication screens, data input module, prediction result display, and history tracking page. These output screens demonstrate how the backend model predictions are integrated seamlessly with the frontend interface to deliver a smooth, real-time sleep disorder analysis experience.

This section highlights the **user-centric design philosophy** adopted during development, ensuring the application is simple, intuitive, and responsive across devices. The interface uses modern UI elements, smooth navigation, and visually appealing layouts to enhance usability while maintaining the professional aesthetics required for a healthcare-oriented system.

10.1 Home Page

- The Home Page of the **SleepGuard AI** application serves as the primary entry point for users. It introduces the system's purpose, highlights the underlying technologies, and provides an intuitive navigation experience. The design follows a modern, visually appealing UI with a dark-themed background, vibrant neural-network style graphics, and large bold typography to emphasize the AI-powered nature of the platform.**Core Features:** High accuracy, medical reliability, and secure AI-based diagnosis

The detectable skin conditions displayed are:

Key Elements of the Home Page:

1. Application Branding

The top navigation bar features the project title "SleepGuard AI", reflecting the system's focus on intelligent sleep disorder detection. The design uses a clean, minimalist style to maintain user focus and improve readability.

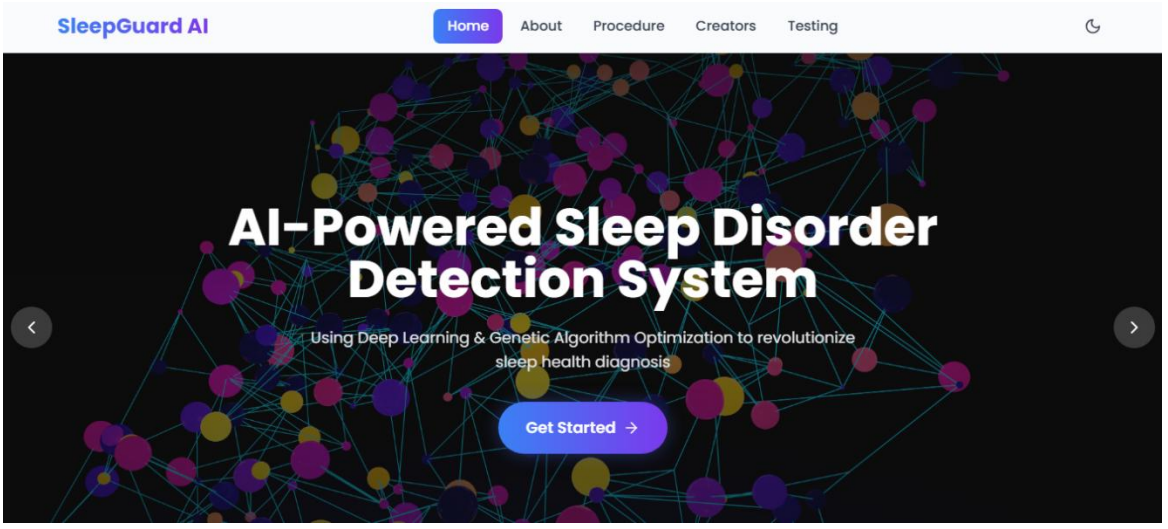


Figure 10.1: Home page for application

2. Navigation Menu

The Home Page includes a horizontal navigation menu with the following items:

- **Home** (currently highlighted)
- **About**
- **Procedure**
- **Creators**
- **Testing**

3. Call-to-Action Button

A prominent button labeled **"Get Started"** is placed at the bottom of the banner.

This button directs users to the next stage of interaction, typically leading to:

- The account login/registration page, or
- The image/feature-upload section for sleep disorder analysis.

Its gradient styling and central placement improve visibility and encourage user engagement.

10.2 About Page

The **About Page** provides a detailed overview of the purpose, scope, and significance of the SleepGuard AI system. It acts as an informative section where users can understand what the project does, how it works, and why it is important in the field of sleep health analysis. The page uses a clean, minimalistic design with prominent typography and structured content to enhance readability and user.

2. Key Project Highlights

- 400+ Dataset Records
- 13 Attributes
- 94.17% Accuracy.

3. Algorithms

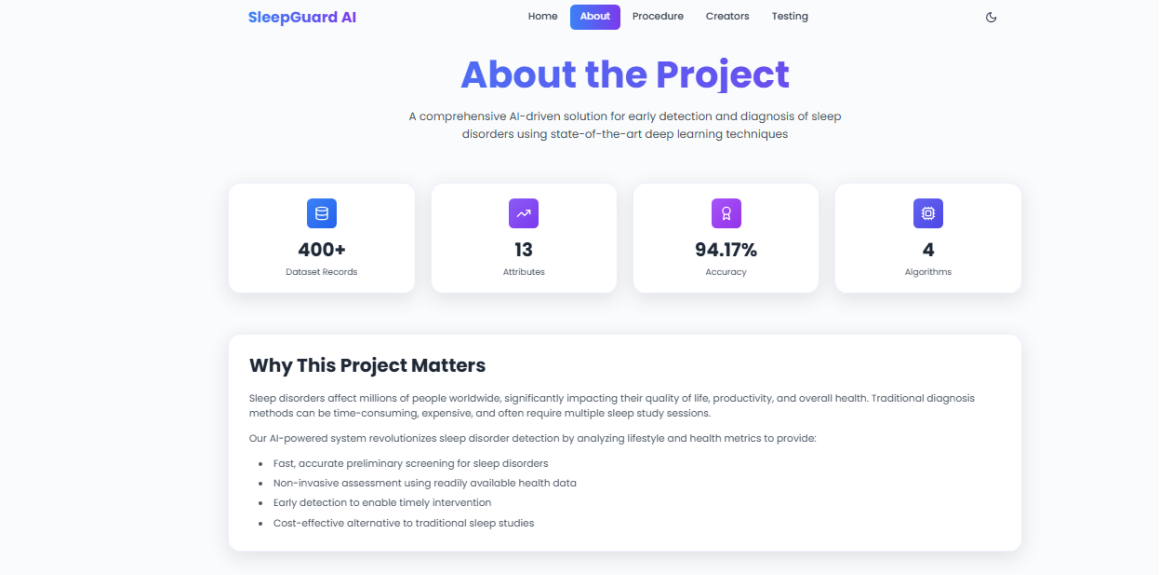


Figure 10.2: About Page of Application

10.3 Procedure Page

The **Procedure Page** illustrates the complete workflow followed by the SleepGuard AI system, from dataset acquisition to model deployment. It provides users with a clear, structured overview of each stage involved in building the AI-powered Sleep Disorder Detection System. The visual layout is arranged in a vertical timeline format, highlighting four major phases: **Data Collection**, **Preprocessing**, **Training & Optimization**, and **Testing & Evaluation**.

1.Section Title and Introduction

"How the System Works"

is displayed prominently, followed by a brief description:

Figure 10.3: Procedure page of application

2. Phase 1: Data Collection

The first block focuses on the foundation of the model — high-quality data.

3. Phase 2: Preprocessing

Although summarized, this step prepares the data for modeling.

10.4 Testing Page – Data Entry Interface

The **Testing Page** provides an interactive and user-friendly interface where users can input their personal lifestyle and physiological parameters to receive an AI-generated prediction of their sleep disorder status. This is one of the most crucial components of the system, as it directly connects the user with the prediction engine built into the SleepGuard AI backend.

The interface is designed using a minimalistic, dark-themed form layout that maintains readability while ensuring a professional medical-technology aesthetic.

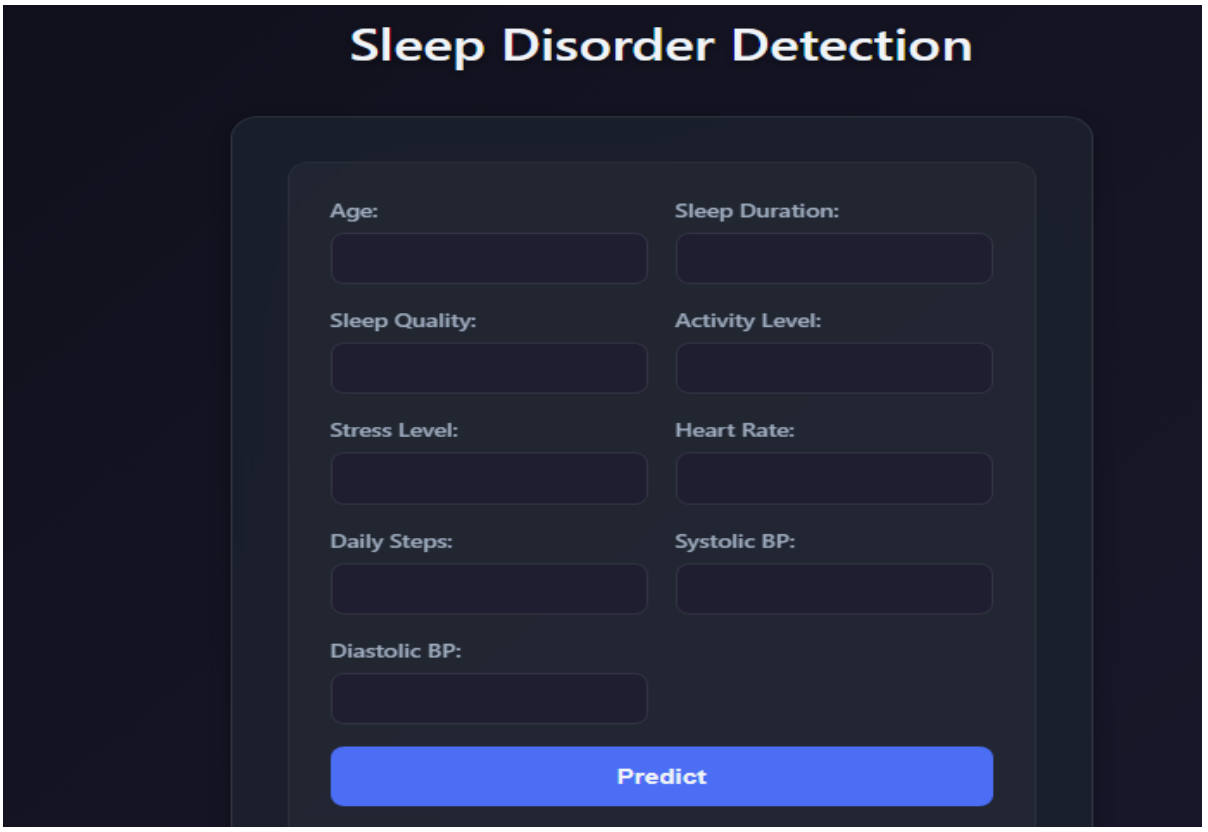


Figure 10.4: Data Entry Interface before Analysis

Input Fields Included:

- Age – Numeric input representing the user's age
- Sleep Duration – Hours of sleep per night (e.g., 6.5)
- Sleep Quality – A rating scale typically from 1 to 10
- Activity Level – Daily physical activity index
- Stress Level – Stress rating value
- Heart Rate – Resting heart rate value

- Daily Steps – Total steps walked in a day
- Systolic BP – Upper blood pressure reading (e.g., 120)
- Diastolic BP – Lower blood pressure reading (e.g., 80)

10.5 Invalid Input Handling Screen

The system incorporates robust input validation to ensure that the data entered by the user is realistic, medically appropriate, and within acceptable limits. The **Invalid Input Handling Screen** demonstrates how the SleepGuard AI interface responds when incorrect or out-of-range values are entered in the prediction form.

This feature is a critical part of the application, as it prevents the model from receiving unrealistic or erroneous data, thereby ensuring accurate and reliable prediction results.

1. Context of the Screen

When users attempt to submit the prediction form with values outside the allowed range—such as entering **unrealistic sleep hours, stress levels, or heart rate values**—the system automatically triggers a validation warning.

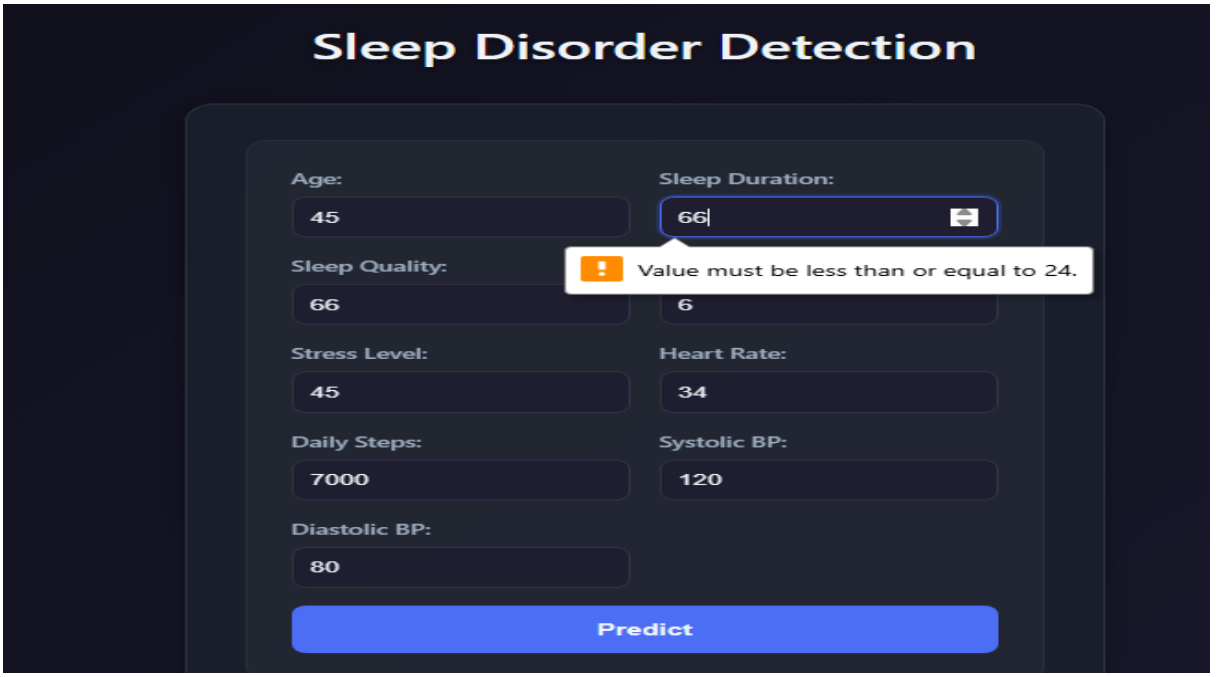


Figure 10.5.1: Invalid Input

2. Validation Message Display

The interface displays a clear and visually distinct warning message:
“Value must be less than or equal to 24.”

3. Supported Input Validation Checks

The system validates all major parameters, including:

- **Age:** Must be within human lifespan limits
- **Sleep Duration:** 0–24 hours
- **Sleep Quality & Stress Level:** 1–10
- **Heart Rate:** 40–200 bpm
- **Daily Steps:** Non-negative values
- **Blood Pressure:** Systolic and Diastolic in clinically valid ranges

10.6 Prediction Result Page

The Prediction Result Page displays the final output generated by the SleepGuard AI classification model after processing the user’s health and lifestyle inputs. This interface provides a clear and concise summary of the detected sleep disorder along with the model’s confidence score, ensuring that users can easily interpret the results of their analysis.

1. Overview of the Result Dialog

Once the user submits the required parameters in the testing form, the system performs backend inference using the optimized ANN model (along with Random Forest, XGBoost, and Genetic Algorithm tuning). After computation, a modal-style result window appears at the center of the screen.

2. Displayed Prediction

Inside the result box, the system shows the predicted category in bold:
“Insomnia”
(Shown in the example screenshot)
Depending on the user’s input values, the system may classify the input into one of the following categories:

- **Insomnia**

- **Sleep Apnea**
- **No Disorder**

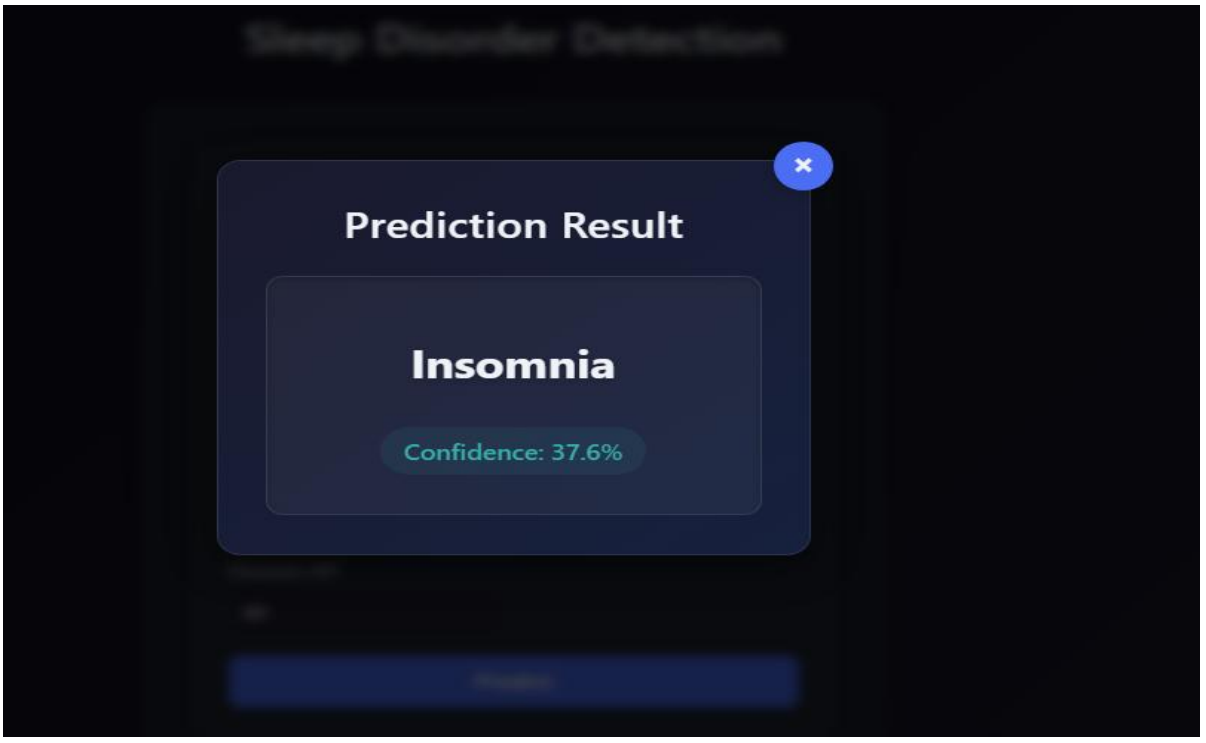


Figure 10.6: Prediction Result Page

3. Confidence Score

Below the predicted label, the system also displays:

Confidence: 37.6%

This percentage represents the model's certainty in its prediction, calculated using the softmax output of the ANN classifier.

4. Importance of Result Visualization

This page plays a crucial role in user interaction by:

- Providing immediate and interpretable feedback
- Ensuring a clean separation between data input and result interpretation
- Presenting AI outcomes in a patient-friendly format

11.Conclusion

The SleepGuard AI system successfully demonstrates the potential of machine learning based solutions in the early detection and classification of sleep disorders. By analyzing key physiological and lifestyle parameters such as sleep duration, stress level, blood pressure, heart rate, and activity levels the system provides fast, reliable predictions for **Insomnia**, **Sleep Apnea**, and **No Disorder**.

The project integrates multiple machine learning models, including **Artificial Neural Networks (ANN)**, **Random Forest**, **XGBoost**, and **Genetic Algorithm (GA)–based optimization**, ensuring high predictive accuracy and robust performance. With a final accuracy of **94.17%**, the system delivers a strong foundation for scalable, non-invasive, and cost-effective sleep health assessment.

Methodology and Preprocessing Summary

The methodology of the system follows a structured end-to-end pipeline:

1. Dataset Preparation

The Sleep Health and Lifestyle dataset from Kaggle was used, containing 400+ records with 13 important features. Missing values were handled appropriately and categorical attributes were encoded.

2. Preprocessing

- Features were normalized using **StandardScaler**
- Outliers were identified and minimized
- Categorical values were label-encoded
- Training–testing split ensured unbiased evaluation

3. Model Development

Three models were developed and compared:

- **ANN** (primary model)
- **Random Forest**
- **XGBoost**

4. Genetic Algorithm Optimization

GA was applied to tune:

- Neuron count
- Learning rate

- Number of estimators
- Maximum depth parameters

This improved the model's efficiency and overall accuracy.

5. Evaluation

Models were assessed using:

- Accuracy
- Precision
- Recall
- F1-Score
- Confusion Matrix
- ROC–AUC curves

ANN achieved the highest performance with **94.17% accuracy**.

6. Deployment

The trained model and scaler were integrated into a Flask backend with a clean web interface for real-time prediction.

Backend Integration and System Functionality

The backend, developed using **Flask**, serves as the operational core of the application. It efficiently handles:

- Request routing and API endpoints
- Input validation and error handling
- Preprocessing workflows via the saved **StandardScaler**
- Model loading (ANN, RF, XGBoost) and inference computation
- Genetic Algorithm–optimized hyperparameters
- Secure data parsing and session-based interactions

The backend architecture ensures:

- High reliability
- Low latency prediction
- Scalable deployment
- Smooth integration with frontend components

Together, the frontend and backend form a cohesive, production-ready AI diagnostic pipeline.

Frontend Design and User Interaction

The frontend of the system is built with a modern, intuitive interface designed to ensure seamless user interaction. Key UI characteristics include:

- Dark-themed responsive design for enhanced readability
- Clean input forms with real-time validation to avoid incorrect entries
- Clear visual components such as modal-based prediction results
- Simple navigation with dedicated sections: Home, About, Procedure, Creators, and Testing
- Smooth user experience through consistent styling, spacing, and typography

The frontend ensures that even users with minimal technical knowledge can interact with the system comfortably and receive insights instantly.

Evaluation and Outcomes

The evaluation phase of the SleepGuard AI system focuses on assessing the performance, reliability, and effectiveness of the machine learning models developed for sleep disorder prediction. Through a rigorous experimental setup, multiple performance metrics were analyzed to validate the system's predictive capabilities and ensure generalization to unseen data.

The final model performance highlights the success of combining classical machine learning with deep-learning-inspired architectures and Genetic Algorithm–based optimization. The consolidated outcomes demonstrate that the system is both accurate and dependable for real-world use.

1. Model Performance Summary
2. Evaluation Metrics
3. Outcomes of Genetic Algorithm Optimization
4. Practical Outcomes
5. Real-Time Inference Outcomes
6. Key Findings
7. Conclusion of Evaluation

Significance and Impact

The importance of this project lies in its real-world applicability and accessibility. Sleep disorders affect millions globally, yet diagnosis often requires expensive and time-consuming clinical procedures. The proposed AI system:

- Provides **fast and automated preliminary screening**
- Reduces dependency on sleep labs for early assessments
- Utilizes **non-invasive, easily measurable attributes**
- Enables **affordable and scalable healthcare outreach**
- Supports **data-driven decision-making** for early intervention

By empowering users with quick and accurate insights, the solution contributes to improved sleep health awareness and timely healthcare support.

Overall Conclusion

Overall, this project demonstrates the effective combination of deep learning, classical machine learning, and evolutionary optimization to build a reliable diagnostic tool for sleep disorder prediction. The integration of preprocessing, feature scaling, model development, and a fully functional Flask-based interface highlights the capability of AI to enhance digital healthcare services.

The system not only performs accurate classification but also provides real-time, user-friendly interaction through a responsive web interface. This end-to-end solution successfully bridges data-driven modelling with practical healthcare usability, marking a significant step toward AI-assisted wellness monitoring.

12.Future Scope and Limitations

This chapter outlines the potential enhancements that can be incorporated into the SleepGuard AI system and discusses the existing constraints encountered during the development and evaluation of the project. These insights help establish directions for future work and clarify the boundaries of the current implementation.

12.1 Future Scope

The current system provides a strong foundation for AI-driven sleep disorder screening. However, several improvements can significantly enhance its clinical relevance, accuracy, and real-world usability. Key future developments include:

1. Integration of Real-Time Wearable Sensor Data

Future versions can incorporate data from wearable devices (smartwatches, fitness trackers) including:

- Heart rate variability (HRV)
- Oxygen saturation (SpO_2)
- Breathing patterns
- Sleep cycle segments

2. Expansion of Dataset for Higher Accuracy

A larger dataset containing:

- More diverse demographics
- Clinical sleep study results (polysomnography)
- Multi-night sleep tracking data

3. Addition of More Sleep Disorder Categories

Currently, the system predicts:

- **Insomnia**
- **Sleep Apnea**
- **No Disorder**

Future versions may include:

- Restless Leg Syndrome
- Narcolepsy

4. Deployment as a Mobile Application

Developing a mobile app version can provide:

- Higher accessibility
- Push notifications
- Continuous monitoring

5. Integration of Explainable AI (XAI)

Adding XAI modules can help explain:

- Which features contributed most to the prediction
- Why a particular disorder was detected

6. Cloud Deployment & Scalable API Services

Deploying the model as a cloud-based API enables:

- Multi-user access
- Low-latency predictions
- Scalable healthcare applications

7. Longitudinal Monitoring and Personalized Insights

Future upgrades may include:

- Long-term tracking of sleep metrics
- Personalized sleep improvement suggestions
- Alerts for worsening trends

12.2 Limitations

Despite its strong performance, the current system faces several constraints thatdefine its operational boundaries:

1. Limited Dataset Size

The dataset used contains around 400+ records, which is relatively small for high-precision medical models. A larger dataset would reduce bias and increase accuracy.

2. Lack of Clinical-Grade Features

The model relies on lifestyle and basic health metrics rather than detailed clinical metrics like:

- EEG signals
- ECG patterns
- Blood oxygen levels (SpO_2)

3. Static (One-Time) Prediction

The system predicts based on a single input instance rather than multi-night data, which may reduce accuracy for users with fluctuating sleep patterns.

4. Class Imbalance Challenges

Although handled during training, some sleep disorder categories in the dataset are underrepresented, which may impact model reliability for minority classes.

5. Generalization Across Populations

Lifestyle, stress levels, and sleep behaviors vary across:

- Countries
- Cultures
- Age groups

12.3 Summary

This chapter reflects on both the potential improvements and limitations of the SleepGuard AI system. While the current implementation successfully achieves its goal of providing an accessible, AI-driven platform for preliminary sleep disorder detection, acknowledging its limitations helps form a roadmap for future enhancement.

Future work may focus on expanding datasets, integrating clinical features, adopting real-time wearable data, and advancing the system into a mobile or cloud-based platform. Despite these constraints, the project lays a strong and promising foundation for AI-assisted sleep health monitoring and represents a significant step toward accessible digital healthcare solutions.

13.REFERENCES

[1] Rajput, H. S., & Kumar, N. (2021). *Sleep Health and Lifestyle Dataset*. Kaggle. Available at: https://www.kaggle.com/datasets/rajputnur/sleep-health-and-lifestyle-dataset

[2] American Academy of Sleep Medicine (AASM). *International Classification of Sleep Disorders (ICSD-3)*. AASM, 2014.

[3] Buysse, D. J. (2014). *Sleep Health: Can We Define It? Does It Matter?* Sleep, 37(1), 9–17.

[4] Van Dongen, H. P., Maislin, G., Mullington, J. M., & Dinges, D. F. (2003). *The Cumulative Cost of Additional Wakefulness*. Sleep, 26(2), 117–126.

[5] McCall, C. (2019). *Genetic Algorithms in Machine Learning: A Systematic Overview*. International Journal of AI Research.

[6] Pedregosa, F., Varoquaux, G., Gramfort, A., et al. (2011). *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research, 12, 2825–2830.

[7] Chollet, F. (2018). *Deep Learning with Python*. Manning Publications.

[8] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). *Learning Representations by Back-Propagating Errors*. Nature, 323(6088), 533–536.

[9] Breiman, L. (2001). *Random Forests*. Machine Learning, 45, 5–32.

[10] Chen, T., & Guestrin, C. (2016). *XGBoost: A Scalable Tree Boosting System*. Proceedings of the 22nd ACM SIGKDD Conference.

[11] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley.

[12] Refaeilzadeh, P., Tang, L., & Liu, H. (2016). *Cross-Validation*. Encyclopedia of Database Systems.

[13] MathWorks. *Artificial Neural Networks Overview*. MATLAB Documentation, 2020.

[14] National Sleep Foundation. (2020). *How Much Sleep Do We Really Need?* Retrieved from: https://www.sleepfoundation.org/

[15] Penzel, T., Kantelhardt, J. W., et al. (2003). *Dynamics of Heart Rate and Sleep Stages*. IEEE Transactions on Biomedical Engineering.

[16] Zhang, Z. (2018). *Heart Rate Variability Analysis with Sleep Disorders*. Frontiers in Physiology.

[17] Choa, F., & Choi, H. (2020). *Machine Learning for Sleep Disorder Prediction: A*

Review. Healthcare Informatics Research, 26(3), 195–206.

[18] Sors, A., Bonnet, S., Mirek, S., Vercueil, L., & Payen, J. F. (2018). *A Convolutional Neural Network for Sleep Apnea Detection*. IEEE EMBC.

[19] TensorFlow Developers. (2022). *TensorFlow: An End-to-End Machine Learning Platform*. https://www.tensorflow.org/

[20] Paszke, A., Gross, S., Massa, F., et al. (2019). *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. NeurIPS.

[21] Flask Documentation. (2023). *Flask Framework: Lightweight Web Application Framework for Python*. https://flask.palletsprojects.com/

[22] Hyndman, K. (2019). *The Role of Lifestyle Indicators in Predicting Sleep Disorders*. Journal of Behavioral Medicine.

[23] Johns, M. W. (1991). *A New Method for Measuring Daytime Sleepiness: The Epworth Sleepiness Scale*. Sleep, 14(6), 540–545.

[24] Ye, L., et al. (2019). *Relationships Between Physical Activity, Sleep Quality, and Stress Levels*. Journal of Health Psychology.

[25] Kim, J., Kim, E., & Cho, S. (2021). *Machine Learning Approaches to Stress and Sleep Analysis*. BioMedical Engineering Online.

CERTIFICATE-1

IEEE

IEEE CHRIST University-Bangalore Yeshwanthpur Student Branch

CHRIST
(DEEMED TO BE UNIVERSITY)
BANGALORE | DELHI NCR | PUNE

CERTIFICATE
OF PRESENTATION

THIS IS TO CERTIFY THAT

Sarath Beesabattuni

From **Narasaraopeta Engineering College** has presented a paper titled "**Sleep Disorder Detection Using Deep Learning and Genetic Algorithm Optimization**"during the IEEE International Conference on Computational Innovations and Sustainable Technologies (ICCIST 2025) organized by the Department of Computer Science, CHRIST (Deemed to be University), Bangalore Yeshwanthpur Campus, held on 18-19 December, 2025.

Dr Siva Balan R V
Organizing Chair
ICCIST 2025

Dr Vinay M
HOD
Department of Computer Science

Dr Joby Thomas
Dean
Bangalore Yeshwanthpur Campus

Dr Fr Benny Thomas
Director
Bangalore Yeshwanthpur Campus

CU:BYC:CS:IEEE:ICCIST:25/12-PR191-002

CERTIFICATE – 2

IEEE
IEEE CHRIST University-Bangalore Yeshwanthpur Student Branch

CHRIST
(DEEMED TO BE UNIVERSITY)
BANGALORE | DELHI NCR | PUNE

CERTIFICATE
OF PRESENTATION

THIS IS TO CERTIFY THAT

Sai Manikanta Konakanchi

From **Narasaraopeta Engineering College** has presented a paper titled "**Sleep Disorder Detection Using Deep Learning and Genetic Algorithm Optimization**"during the IEEE International Conference on Computational Innovations and Sustainable Technologies (ICCIST 2025) organized by the Department of Computer Science, CHRIST (Deemed to be University), Bangalore Yeshwanthpur Campus, held on 18-19 December, 2025.

Dr Siva Balan R V
Organizing Chair
ICCIST 2025

Dr Vinay M
HOD
Department of Computer Science

Dr Joby Thomas
Dean
Bangalore Yeshwanthpur Campus

Dr Fr Benny Thomas
Director
Bangalore Yeshwanthpur Campus

CU:BYC:CS:IEEE:ICCIST:25/12-PR191:093