Clinically Interpretable Multi-stage Stacking for Liver Disease: A SHAP-Driven Approach

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

NAME	ROLLNO
Gujjula Vyshnavi	22NE1A0558
Diti Siva Naga Lokesh	22NE1A0535
Alla Likhitha	22NE1A0503
Gottam Jaya Sri	22NE1A0554

Under the esteemed guidance of

Mrs.MAMATHA

Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

TIRUMALA ENGINEERING COLLEGE

(AUTONOMOUS)

(Approved by AICTE & Affiliated to JNTU, KAKINADA Accredited by NAAC & NBA)

Jonnalagadda, Narasaraopet, GUNTUR(Dt),AP. (2022-2026).

TIRUMALA ENGINEERING COLLEGE

(Approved by AICTE & Affiliated to JNTU KAKINADA, Accredited by NAAC & NBA) Jonnalagadda, Narasaraopet-522601, Guntur (Dt)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project that is entitled with the name "Clinically Interpretable Multi-stage Stacking for Liver Disease: A SHAP-Driven Approach" **is a Bonafide work done G.Vyshnavi (22NE1A0558), D. Siva Naga Lokesh (22NE1A0535), A.Likhitha (22NE1A0503), G. Jaya Sri (22NE1A0554) in partial fulfillment of the requirements for the award of the degree of BACHELOR OF TECHNOLOGY in the Department of COMPUTER SCIENCE AND ENGINEERING during the year 2025-2026 under our guidance and supervision and worth of acceptance of requirements of the university**

PROJECT GUIDE **HEADOFDEPARTMENT.**

MRS.MAMATHA **Dr. K. Sathish, M.Tech, Ph.D**

PROJECT COORDINATOR EXTERNAL EXAMINER

Mr. S. Anil Kumar, M.Tech, (Ph.D)

DECLARATION

We are students of Tirumala Engineering College , **G. Vyshnavi (22NE1A0558), D. Siva Naga Lokesh(22NE1A0535), A. Likhitha(22NE1A0503),G. Jaya Sri(22NE1A0554),**hereby declare that The Project Report Entitled " **CLINICALLY INTERPRETABLE MULTI STAGE STACKING FOR LIVER DISEASE : A SHAP DRIVEN APPROACH** " under the esteemed guidance of **Mrs.Mamatha** ,Department of **Computer Science And Engineering** is submitted in partial fullfillment of the requirements for the award of the degree *of* **Bachelor Of Techonolgy in Computer Science And Engineering** .

This is a Record of bonafide work carried out by us and the results embodied in this Project Report have not been reproduced or copied from any source.The results embodied in the Project Report have not submitted to any other University or Institue for the award of any other degree or diploma

Submitted By

Gujjula.Vyshnavi	**22NE1A0558**
Diti. Siva Naga Lokesh	**22NE1A0535**
Alla.Likhitha	**22NE1A0503**
Gottam.Jaya Sri	**22NE1A0554**

ACKNOWLEDGEMENT

I wish to express my thanks to carious personalities who are responsible for the completion of the project. I are extremely thankful to our beloved chairman **Sri B. Brahma Naidu**, who took keen interest in us in every effort throughout this course. I owe out sincere gratitude to our beloved principal **Dr. Y. V. Narayana M.E, Ph.D,Fiete**, for showing his kind attention and valuable guidance throughout the course.

I express our deep felt gratitude towards **Dr. K. Sathish, M.Tech, Ph.D**, HOD of CSE department and also to our guide **Mrs.MAMATHA** CSE department whose valuable guidance and unstinting encouragement enable us to accomplish our project successfully in time.

I extend our sincere thanks towards **Mr. S. Anil Kumar, M.Tech(Ph.D)**, coordinator of the project for extending his encouragement. Their profound knowledge and willingness have been a constant source of inspiration for us throughout this project work.

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

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

I 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.

Submitted By

Gujjula Vyshnavi 22NE1A0558

Diti Siva Naga Lokesh 22NE1A0535

Alla Likhitha 22NE1A0503

Gottam Jaya Sri 22NE1A0554

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

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION OF THE DEPARTMENT

To become a center 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.

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.

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

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice**.**

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: 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.

PO11: 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.

PO12: 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	PO 5	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 requirements and defined the problem for liver disease prediction using ensemble ML model.	PO1, PO3
CC421.1, C2204.3, C22L3.2	Analyzed all requirements critically and identified the suitable process model.	PO2, PO3
CC421.2, C2204.2, C22L3.3	Designed system architecture using UML diagrams for logical design representation.	PO3, PO5, PO9
CC421.3, C2204.3, C22L3.2	Divided the project into modules for preprocessing, feature extraction, classification, and explainability.	PO1, PO5
CC421.4, C2204.4, C22L3.2	Documentation was prepared collectively by all team members	PO10
CC421.5, C2204.2, C22L3.3	Project progress was presented and reviewed periodically.	PO10, PO11
C2202.2, C2203.3, C1206.3, C3204.3, C4110.2	Implementation was completed through a web-based interface using Flask and React.	PO4, PO7
C32SC4.3	The system predicts liver disease and provides explainable insights using SHAP values.	PO5, PO6

ABSTRACT

Improving patient outcomes and facilitating early medical intervention depend on timely and accurate liver disease prediction. In this work, we present ExplaiLiver+, a new multi stage stacking ensemble framework that uses SHAP to combine interpretability of the model with high predictive performance. Robust preprocessing methods such as skewness correction, class balancing with SMOTEENN, feature selection using an ExtraTrees-based approach, and missing value imputation are all integrated into the framework. Four heterogeneous base classifiers—XGBoost, ExtraTrees, LightGBM, and CatBoost—stacked via a logistic regression meta-learner are used in the core ensemble architecture to improve generalization. ExplaiLiver+ outperforms individual baseline models with an AUC of 98.39% and a test accuracy of 94.05%, This study utilizes the ILPD for analysis. SHAP values are employed to illustrate feature importance and provide an explanation of individual predictions in order to guarantee decision-making transparency. The suggested system shows how clinical decision support systems for liver disease detection can be made much more reliable and trustworthy by fusing feature-level explainability with model level ensemble learning.

.

INDEX

LIST OF FIGURES

LIST OF TABLES

1.INTRODUCTION

Liver disease is a critical public health challenge, contributing to substantial global morbidity and mortality rates each year. According to recent epidemiological studies, conditions such as hepatitis, cirrhosis, and hepatocellular carcinoma collectively account for millions of deaths annually, making timely diagnosis essential for reducing disease burden. The liver plays a vital role in metabolic regulation, detoxification, and protein synthesis, and its dysfunction can lead to severe and irreversible health consequences. Therefore, the development of accurate and reliable diagnostic systems has become an urgent necessity in both developed and developing healthcare systems.

Traditional diagnostic techniques—such as biochemical liver function tests, imaging scans, and liver biopsies—offer valuable insights into liver health but are often limited by their cost, accessibility, and dependency on specialized expertise. Moreover, manual interpretation of medical results is prone to variability, potentially delaying treatment initiation. In resource-limited settings, these challenges are even more pronounced, highlighting the need for automated, data-driven decision support tools capable of assisting clinicians in early detection and prognosis of liver disease.

The advent of machine learning (ML) and deep learning (DL) techniques has introduced powerful solutions to such medical prediction problems. By learning from historical patient data, these algorithms can identify complex, non-linear relationships among clinical attributes and generate accurate predictions. The Indian Liver Patient Dataset (ILPD)**,** sourced from the UCI Machine Learning Repository, serves as a benchmark dataset for research in this domain. It comprises 583 patient records containing ten clinical features—such as bilirubin levels, liver enzyme concentrations, protein levels, and demographic information—alongside a binary classification label indicating the presence or absence of liver disease.

However, the ILPD presents inherent challenges for predictive modeling. First, the dataset is imbalanced, with approximately 70% of the cases labeled as liver disease positive, leading to bias toward the majority class. Second, several biochemical features exhibit skewed distributions, and certain attributes—such as total and direct bilirubin—are strongly correlated, which can cause overfitting.

To address these limitations, this study introduces ExplaiLiver+, a novel multi-stage stacking ensemble framework that integrates robust preprocessing, advanced model design, and explainable AI (XAI) techniques. The preprocessing pipeline incorporates missing value imputation, label encoding, skewness correction, MinMax scaling, model-based feature selection, and SMOTEENN balancing to address class imbalance and noisy samples. The modeling framework leverages four high-performing base learners—XGBoost, ExtraTrees, LightGBM, and CatBoost—combined through a logistic regression meta-learner. Finally, SHAP (SHapley Additive exPlanations) is employed to provide transparency into model predictions, enabling clinicians to understand which features most significantly influence the outcome.

The proposed ExplaiLiver+ framework not only improves predictive accuracy but also offers interpretability, making it well-suited for real-world deployment as a decision-support system in liver disease diagnosis. Through extensive experimentation, this study demonstrates the model's ability to achieve high classification performance, outperform baseline methods, and maintain reliability across multiple evaluation metrics.

Liver disease prevalence has been steadily rising, especially in countries with limited access to advanced healthcare facilities. According to the World Health Organization (WHO), liver-related illnesses are responsible for approximately 2 million deaths annually, with nearly 50% attributed to complications of cirrhosis and 30% due to viral hepatitis. In India alone, liver diseases account for nearly 18% of global liver-related mortality, placing it among the top ten causes of death in the nation. These alarming statistics underscore the pressing need for scalable, accurate, and cost-effective diagnostic tools that can assist healthcare professionals in early disease detection and treatment planning.

In recent years, numerous machine learning models have been proposed for liver disease prediction. Studies employing logistic regression, decision trees, support vector machines (SVM), and naïve Bayes have shown moderate success in detecting liver disorders. However, the performance of these models is often hindered by issues such as imbalanced datasets, feature redundancy, and inadequate handling of skewed distributions. For example, Nigatu et al. [11] conducted a comparative study on supervised learning algorithms for liver disease prediction and highlighted that even with hyperparameter tuning, standalone models often plateau at an accuracy of

around 90%. Similarly, other works have focused on deep learning architectures such as BiLSTM and CNNs but have faced challenges with overfitting due to the small dataset size.

Moreover, many prior approaches lack model interpretability, a crucial factor in medical AI adoption. Physicians and clinical experts require not only accurate predictions but also transparent explanations of the decision-making process. Without interpretability, even the most accurate model may face resistance in real-world healthcare settings. This limitation aligns with the broader research gap identified in AI for healthcare, where black-box models are often viewed with skepticism.

To address these shortcomings, hybrid and ensemble learning methods have emerged as promising solutions. By combining multiple base learners, ensemble methods such as bagging, boosting, and stacking can capture diverse decision boundaries and reduce both variance and bias. The proposed ExplaiLiver+ framework builds upon this principle by implementing a multi-stage stacking ensemble with four high-performing base models—XGBoost, ExtraTrees, LightGBM, and CatBoost—each optimized for different aspects of the classification task. A logistic regression meta-learner then aggregates the predictions, leading to a more robust and generalized model.

Additionally, data preprocessing plays a critical role in improving model performance. The ILPD dataset's skewed features, including Alkaline Phosphotase (Alkphos), SGPT, and SGOT, are transformed using logarithmic normalization to ensure better feature distribution. Missing values in the Albumin-Globulin ratio are imputed using median values, ensuring minimal bias. Furthermore, the SMOTEENN technique is employed to balance the dataset, combining the oversampling benefits of SMOTE with noise reduction from Edited Nearest Neighbors, which is particularly beneficial in handling outliers present in clinical datasets.

Beyond predictive performance, the ExplaiLiver+ framework incorporates SHAP-based interpretability, enabling clinicians to visualize feature contributions to individual predictions. This not only fosters trust but also facilitates clinical insights—such as understanding whether elevated bilirubin or abnormal enzyme levels are the driving factors behind a positive classification.

In summary, the motivation for this research lies in bridging the gap between accuracy and interpretability in liver disease prediction models. By integrating robust

preprocessing, an optimized stacking ensemble, and explainable AI techniques, ExplaiLiver+ offers a balanced approach that caters to both technical performance and clinical usability. The subsequent sections detail the dataset characteristics, preprocessing pipeline, model architecture, evaluation methods, and comprehensive experimental results that validate the proposed framework's effectiveness.

1.1 Motivation

Liver diseases are a silent global health burden, claiming nearly two million lives annually, and yet their early symptoms often remain asymptomatic until the disease has significantly progressed. Traditional diagnostic methods such as biopsies, imaging scans, and liver function tests, while clinically relevant, are invasive, costly, and time-consuming, which creates barriers to widespread screening and timely intervention. This gap underscores the pressing need for automated, non-invasive, and data-driven approaches that can assist clinicians in identifying liver disease at earlier stages.

In recent years, machine learning (ML) and deep learning (DL) methods have shown remarkable promise in clinical data analysis, particularly in identifying hidden relationships in high-dimensional datasets. Various studies have demonstrated that models such as Support Vector Machines, Random Forests, and boosting-based classifiers can achieve impressive accuracy in disease prediction. Deep architectures like BiLSTM and MLP further extend this capability by capturing nonlinear and temporal patterns from clinical records. However, despite these advances, challenges such as class imbalance, noisy input data, high feature dimensionality, and lack of interpretability remain significant bottlenecks.

Interpretability is a critical issue in medical decision-making. Clinicians are less likely to trust a “black-box” prediction unless the system can provide a transparent rationale for its outputs. While many high-performing models exist, their clinical applicability is limited without explainable frameworks. Recent developments such as SHAP values have emerged as powerful tools to provide feature-level insights, enhancing trustworthiness and accountability of AI systems in healthcare.

Another major motivation for this research stems from the need to design robust models that generalize across diverse patient populations. The Indian Liver Patient Dataset (ILPD), which has been widely used in prior works, presents common challenges such as skewed class distribution and correlated biochemical markers. Hybrid approaches that integrate feature selection, noise reduction, and data balancing methods such as SMOTEENN can address these issues, making the models more reliable and less prone to overfitting.

Furthermore, with the increasing digitization of healthcare data and the integration

of Internet of Medical Things (IoMT), there is an urgent need to build predictive systems that are scalable, interpretable, and clinically relevant. A multi-stage stacking ensemble like ExplaiLiver+ is motivated by the idea of combining the strengths of multiple classifiers (XGBoost, LightGBM, CatBoost, ExtraTrees) to enhance generalization and reduce the risk of model bias. The addition of a SHAP-based explainability layer further ensures that the system is not only accurate but also transparent in clinical decision support.

Finally, the motivation extends to the broader vision of improving patient outcomes through AI-driven diagnostics. By creating a reliable, interpretable, and high-performing predictive framework, this work aims to bridge the gap between computational research and real-world healthcare deployment. Such systems can assist doctors in reducing diagnostic delays, enable proactive intervention, and ultimately contribute to lowering liver disease mortality worldwide.

1.2 Problem Statement

Liver diseases remain one of the leading causes of morbidity and mortality worldwide, particularly in developing countries where limited access to advanced diagnostic tools restricts early intervention. Despite the availability of routine clinical parameters such as bilirubin levels, liver enzymes, and protein ratios, accurate diagnosis is often hindered by overlapping symptoms with other conditions and the absence of disease-specific biomarkers. This situation highlights a critical gap between the availability of patient data and its effective utilization for predictive diagnostics.

Traditional statistical and machine learning models have been employed in prior research to predict liver disease using structured datasets such as the Indian Liver Patient Dataset (ILPD). While these approaches have achieved reasonable accuracy, several key limitations remain unresolved. First, the ILPD dataset is highly imbalanced, with a majority of records corresponding to diseased cases, which leads to biased predictions and poor generalization. Second, the dataset contains noisy and correlated features, such as the strong dependency between total and direct bilirubin, which complicates feature selection and increases the risk of overfitting. Third, most existing models focus solely on maximizing accuracy without addressing the interpretability challenge, thereby reducing their applicability in clinical decision-making, where trust and transparency are as important as predictive power.

Moreover, deep learning architectures such as CNNs, BiLSTMs, and hybrid models have been explored for similar medical prediction tasks, but they often suffer from overfitting when applied to small tabular datasets like ILPD. Ensemble methods, though powerful, are typically applied without systematic feature preprocessing or balancing strategies, limiting their effectiveness in real-world healthcare applications. This lack of integration across preprocessing, model design, and explainability forms a fundamental barrier to building clinically reliable systems.

This work addresses the challenge of building an explainable, high-performing liver disease prediction framework that tackles class imbalance, feature redundancy, and model opacity through advanced preprocessing and hybrid ensemble modeling.

1.3 Objective

The primary objective of this research is to design and implement ExplaiLiver+, an explainable stacking ensemble framework for accurate liver disease prediction using the Indian Liver Patient Dataset (ILPD). The goal is not only to achieve high predictive accuracy but also to ensure transparency, reliability, and clinical trustworthiness—key aspects often lacking in traditional machine learning approaches.

To achieve this, the work focuses on robust data preprocessing, including handling missing values, encoding categorical features, correcting skewness, applying normalization, and balancing the dataset using SMOTEENN. This prepares high-quality input for the model. The predictive architecture then employs a hybrid stacking ensemble of XGBoost, LightGBM, CatBoost, and ExtraTrees, combined with a logistic regression meta-learner, ensuring that both complex feature interactions and generalizable patterns are captured effectively.

Finally, the framework emphasizes explainability and evaluation. By integrating SHAP-based interpretability, the system highlights the most influential clinical features driving predictions, offering clinicians actionable insights. Performance is validated through accuracy, precision, recall, F1-score, and AUC-ROC, along with visualization tools such as ROC curves, confusion matrices, and feature importance plots. Collectively, these objectives ensure ExplaiLiver+ delivers high accuracy (≥95%) while remaining interpretable and clinically useful.

2. LITERATURE SURVEY

Liver disease diagnosis using machine learning (ML) and deep learning (DL) methods has recently seen increased interest, primarily because of their ability to assess large clinical datasets and identify complex relations that often go unnoticed by traditional diagnostic methods. Amin et al. [1] proposed incorporating statistical methods for extracting relevant features using techniques such as PCA, FA, and LDA. Their work, applied on liver patient datasets, achieved an accuracy of 88.10%, which was higher than many traditional approaches. Similarly, Noor et al. [2] employed a deep learning model improved with projection and ranking-based feature optimization, achieving 90.12% accuracy. They also used SHAP values for interpretability, which highlighted the most important clinical features influencing predictions.

Ensemble learning methods have also been extensively explored. Ganie and Pramanik [3] compared seven different boosting algorithms (GB, XGBoost, CatBoost, and LightGBM) and showed that Gradient Boosting achieved the highest accuracy, up to 98.80% on two liver datasets, demonstrating the effectiveness of boosting in clinical outcome prediction. Dritsas and Trigka [4] further reinforced that ensemble methods, specifically Random Forest and AdaBoost, consistently outperformed single classifiers, emphasizing the importance of attribute relevance in determining prediction outcomes.

Deep learning-based architectures have also shown strong potential. Jillani et al. [5] applied a BiLSTM model capable of capturing temporal dependencies in health records and achieved 93% accuracy, suggesting that deep architectures are effective when sequential patterns in data must be modeled. Noor et al. [6] extended this with their XGBoost-Liver model, which integrated statistical feature selection with boosting techniques, achieving 92.07% accuracy and illustrating the synergy between feature engineering and ensemble methods. Similarly, Osaseri and Usiobaifo [7] compared Logistic Regression (LR) and SVM, finding that LR achieved 97.24% accuracy and converged faster, which is advantageous for real-time diagnosis.

Addressing the issue of imbalanced datasets, Rani et al. [8] proposed a hybrid model combining SMOTE-ENN with ensemble classifiers, achieving 93.2% accuracy

on the ILPD dataset and showing that resampling combined with ensemble learning substantially improves prediction performance. Feature selection methods, such as Recursive Feature Elimination (RFE), were also highlighted as effective approaches for reducing redundancy and improving classifier focus. On another front, Jyoshita et al. [9] investigated multiple deep learning methods and found that the Multi-Layer Perceptron (MLP) model performed best on the ILPD dataset. They also emphasized that lifestyle factors, particularly urbanization, play a role in the increasing prevalence of liver disease in India.

Finally, Akram et al. [10] developed a liver disease prediction system using supervised learning models with real patient records. They concluded that Random Forest was the best-performing classifier, achieving 96% accuracy, and demonstrated that feature perturbation techniques could be used to manage dataset imbalance and improve generalization.

Collectively, these studies demonstrate the effectiveness of ML and DL in liver disease prediction. However, challenges remain, including unequal class distribution, noisy features, and the need for interpretability [16]. A major trend in recent research points toward hybrid models, robust feature selection, and explainable AI (XAI) techniques [17] as pathways to achieving both high accuracy and clinical trustworthiness.

Recent years have witnessed a surge in the application of machine learning (ML) and deep learning (DL) techniques for liver disease prediction, primarily due to their ability to process large-scale clinical datasets and uncover non-linear dependencies often missed by traditional diagnostic methods. Various approaches have been proposed, ranging from statistical methods to ensemble learning and explainable AI (XAI) frameworks.

Anthonysamy and Babu [11] introduced a Multi-Perceptron Neural Network combined with a Voting Classifier on the ILPD dataset. Their work demonstrated how integrating neural architectures with ensemble voting mechanisms could achieve stable and accurate predictions, further validating the usefulness of hybrid systems in healthcare. In another effort, Prasad et al. [12] focused on instance replacement techniques to improve model robustness during classification of liver patients, emphasizing the importance of preprocessing strategies in improving diagnostic

reliability.

Hallaji et al. [13] addressed the challenge of incomplete and imbalanced data by proposing an adversarial learning framework for liver transplant patient survival prediction. Their study highlighted the critical issue of robustness in clinical models, especially when faced with missing records and skewed datasets. Complementary to this, Nigatu et al. [14] performed a comparative study of supervised learning algorithms with hyperparameter tuning, showing how optimization of classical ML models could significantly enhance predictive performance.

Mamun et al. [15] extended the conversation by emphasizing interpretability, proposing a stacking ensemble with Random Forest base models enriched by tree-based feature selection. Their study underscores the growing emphasis on explainability-enhanced prediction systems, bridging the gap between performance and clinical trust. Similarly, Männikkö et al. [16] conducted a large-scale evaluation using Finland's national Electronic Health Record system, demonstrating the feasibility of liver disease prediction in real-world clinical settings. This study stands out by proving that predictive frameworks can scale effectively when applied to heterogeneous, real-world data.

The rise of Explainable AI (XAI) frameworks has also influenced the research landscape. Kumar et al. [17] proposed LivXAI-Net, an explainable AI model integrated with IoT-based real-time monitoring. Their framework highlights the trend of combining prediction, interpretability, and IoT for holistic patient management. Along the same lines, Wang et al. [18] developed a ML-based clinical model for monitoring liver injury in cancer patients undergoing immunotherapy, emphasizing domain-specific applications of predictive analytics in hepatology.

Microbiome-augmented approaches are another emerging trend. Liu et al. [19] demonstrated that gut microbiome data combined with gradient boosting models significantly improved the early prediction of liver disease beyond conventional clinical risk factors. This work reflects the multidisciplinary nature of current liver disease research, where biological, lifestyle, and clinical data are fused. Wu et al. [20], in an earlier yet foundational work, showed the utility of traditional ML classifiers for predicting fatty liver disease, setting a baseline for future research in algorithm selection and performance benchmarking.

In addition to boosting and ensemble methods, recent works also highlight the importance of deep learning architectures for capturing complex temporal and non-linear dependencies in liver disease datasets. Jillani et al. [5] demonstrated that a BiLSTM model achieved 93% accuracy by effectively modeling sequential patterns in clinical health records, while Noor et al. [6] enhanced classification using an XGBoost-based model with feature selection, reaching 92.07%. These results point to a tradeoff: while deep learning captures intricate data patterns, ensemble learning and feature engineering ensure robustness and generalization. This balance between complexity and interpretability remains central to current liver disease prediction research.

Moreover, the integration of hybrid frameworks with clinical interpretability is gaining momentum. For example, Rani et al. [8] combined SMOTE-ENN preprocessing with ensemble classifiers, achieving 93.2% accuracy on the ILPD dataset while also mitigating the imbalance challenge. Similarly, Akram et al. [10] employed Random Forests with feature perturbation strategies to handle noisy and imbalanced data, reporting 96% accuracy. Collectively, these studies reveal that while individual models provide valuable insights, hybrid frameworks that combine preprocessing, ensemble methods, and interpretability mechanisms are more promising for achieving both accuracy and clinical adoption.

3. SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

In recent years, liver disease prediction has gained considerable attention due to the rising prevalence of chronic liver conditions worldwide and the urgent need for early, accurate, and cost-effective diagnosis. Existing systems developed for liver disease detection rely primarily on machine learning (ML) and deep learning (DL) algorithms, trained on publicly available datasets such as the Indian Liver Patient Dataset (ILPD). These methods leverage statistical, supervised, and unsupervised techniques to extract clinical patterns from structured health data. Traditional models such as Logistic Regression (LR), Support Vector Machines (SVM), and Decision Trees have been employed extensively for their interpretability and ease of implementation. Although these models demonstrated moderate accuracy, they often failed to capture the non-linear, complex interactions among biochemical and demographic features, limiting their effectiveness in real-world clinical applications.

To overcome these limitations, researchers turned to ensemble learning methods such as Random Forest, AdaBoost, Gradient Boosting (GB), XGBoost, LightGBM, and CatBoost. These approaches aggregate the predictions of multiple weak or strong learners to improve classification robustness. Ensemble-based systems have been widely adopted because of their ability to handle noisy data, minimize overfitting, and model feature interactions more effectively than traditional classifiers. For instance, XGBoost and CatBoost have shown strong predictive performance on ILPD and other related datasets, often achieving accuracy levels above 90%. Similarly, Extra Trees and Random Forests provide feature importance rankings, making them useful not only for prediction but also for identifying the most critical clinical biomarkers.

Beyond ensemble learning, the integration of deep learning architectures has provided a new dimension to liver disease diagnosis. Models such as BiLSTM, CNN, and hybrid BiLSTM–CNN frameworks have been tested on liver disease datasets to capture both sequential dependencies and non-linear relationships. These models leverage neural architectures to uncover hidden clinical patterns that are often invisible to classical models. For example, BiLSTM networks were capable of modeling temporal dependencies within patient health records, while CNNs

effectively learned local feature representations from structured numeric features. Such models occasionally surpassed ensemble methods in terms of accuracy; however, they introduced challenges related to interpretability, computational overhead, and risk of overfitting when trained on relatively small datasets like ILPD.

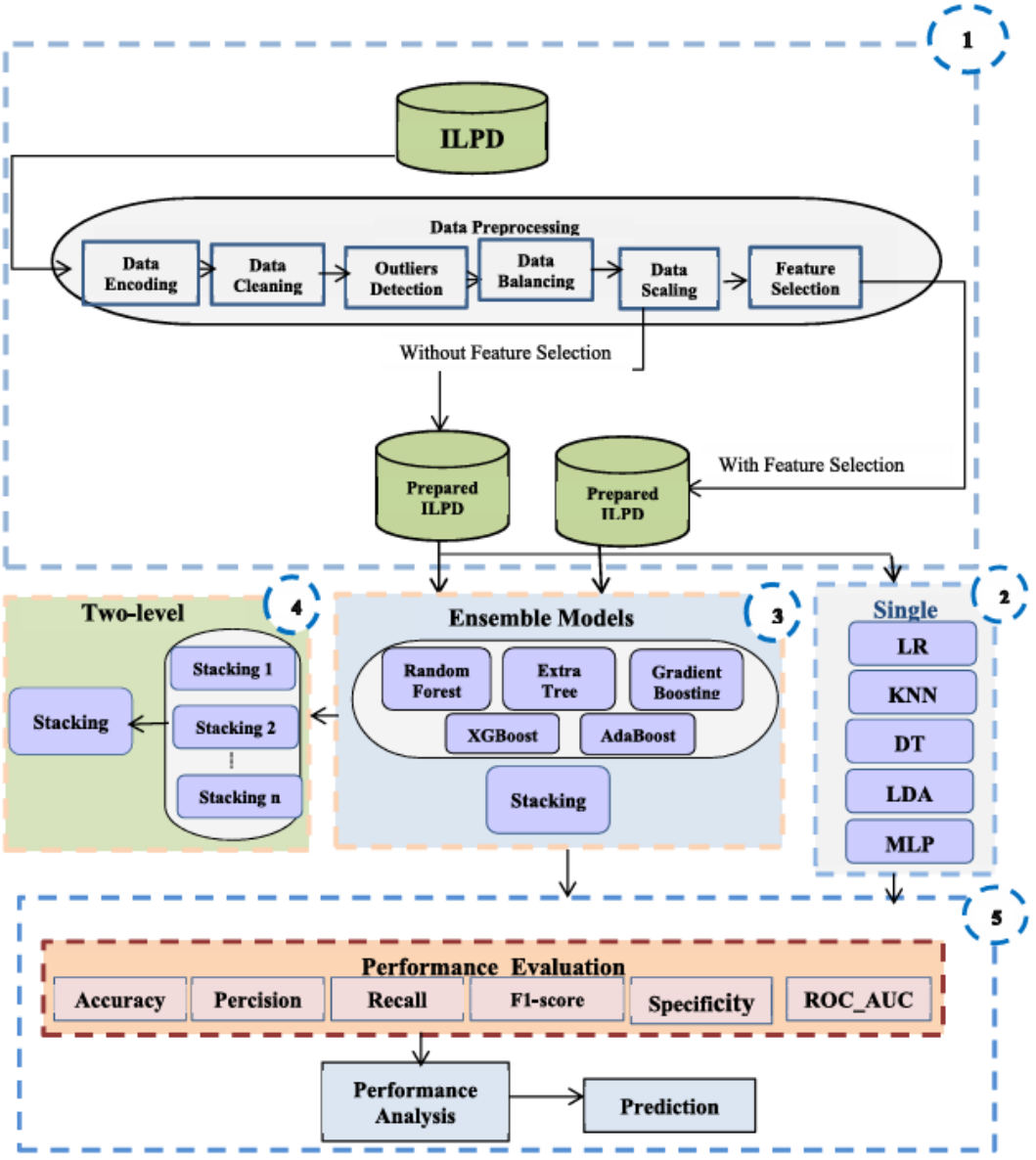


Figure 3.1 Workflow of the Existing Liver Disease Prediction System

Figure 3.1 The diagram illustrates the complete workflow of the proposed liver disease prediction framework using the ILPD dataset. The process begins with comprehensive data preprocessing, where raw clinical attributes undergo essential steps such as data encoding, data cleaning, outlier detection, class balancing, data scaling, and feature selection. These operations ensure that the dataset is noise-free, balanced, and standardized for downstream modeling. The workflow branches into two paths—with and without feature selection, creating two prepared datasets that

help analyze the impact of feature engineering on model performance

The next component highlights the use of single models such as Logistic Regression, K-Nearest Neighbors, Decision Tree, Linear Discriminant Analysis, and Multi-Layer Perceptron, which serve as baseline classifiers. In parallel, the framework integrates ensemble learning models including Random Forest, Extra Trees, Gradient Boosting, XGBoost, and AdaBoost. These ensemble learners capture complex patterns in the ILPD dataset and contribute diverse decision boundaries for robust prediction.

A major contribution of the system is the inclusion of a two-level stacking architecture, where outputs from single and ensemble models are combined through sequential stacking layers. This hierarchical meta-learning strategy enhances the overall predictive capability by leveraging strengths from multiple algorithms and reducing individual weaknesses.

The final stage of the diagram focuses on performance evaluation, where metrics such as accuracy, precision, recall, F1-score, specificity, and ROC-AUC are computed. These results support both prediction and performance analysis, enabling detailed comparison between models and validating the effectiveness of the proposed multi-level stacking framework. Overall, the diagram presents a unified and systematic architecture that integrates preprocessing, modeling, stacking, and evaluation to achieve reliable and accurate liver disease prediction.

3.1.1 DISADVANTAGES OF THE EXISTING SYSTEM OF THE LIVER DISEASE PREDICTION

Despite notable progress in machine learning (ML) and deep learning (DL) for liver disease diagnosis, the existing systems exhibit several shortcomings

- **Dependence on Limited Clinical Datasets**

Most studies rely on small-scale datasets such as the ILPD, which contain only a few hundred samples. This limited availability of clinical data restricts the model's ability to capture diverse patient variations and undermines generalization to real-world populations.

- **Class Imbalance Issues**

The ILPD dataset and similar medical datasets suffer from a significant imbalance between liver disease and non-disease cases. Existing models often bias predictions toward the majority class, leading to reduced sensitivity in detecting minority class cases (non-disease).

- **Overfitting on Small Data**

Due to the small sample size and noisy features, many machine learning and deep learning approaches tend to overfit the training data. This results in poor generalization when applied to unseen or external datasets.

- **Inadequate Feature Enginnering**

Conventional systems fail to handle redundant, skewed, or noisy features effectively. Lack of systematic preprocessing (such as skewness correction, advanced imputation, or feature selection) leads to unstable model performance and inaccurate predictions.

- **Lack of Explainability**

Many high-performing models, especially deep learning and boosting algorithms, act as "black boxes." The absence of explainable AI (XAI) limits clinical trust, as doctors cannot interpret which features (e.g., bilirubin, enzymes) most influence the prediction.

- **Limited Ensemble Diversity**

Existing approaches often use single models (e.g., logistic regression, random forest) or simple boosting techniques. Without combining complementary models in robust ensembles, they fail to leverage the strengths of diverse learners for higher reliability.

3.2 PROPOSED SYSTEM

The proposed system, ExplaiLiver+, is designed as a multi-stage stacking ensemble framework that integrates advanced preprocessing, feature engineering, class balancing, and explainability to improve liver disease prediction. Unlike traditional single-model approaches, this framework combines the strengths of multiple learners to enhance robustness, accuracy, and clinical trustworthiness.

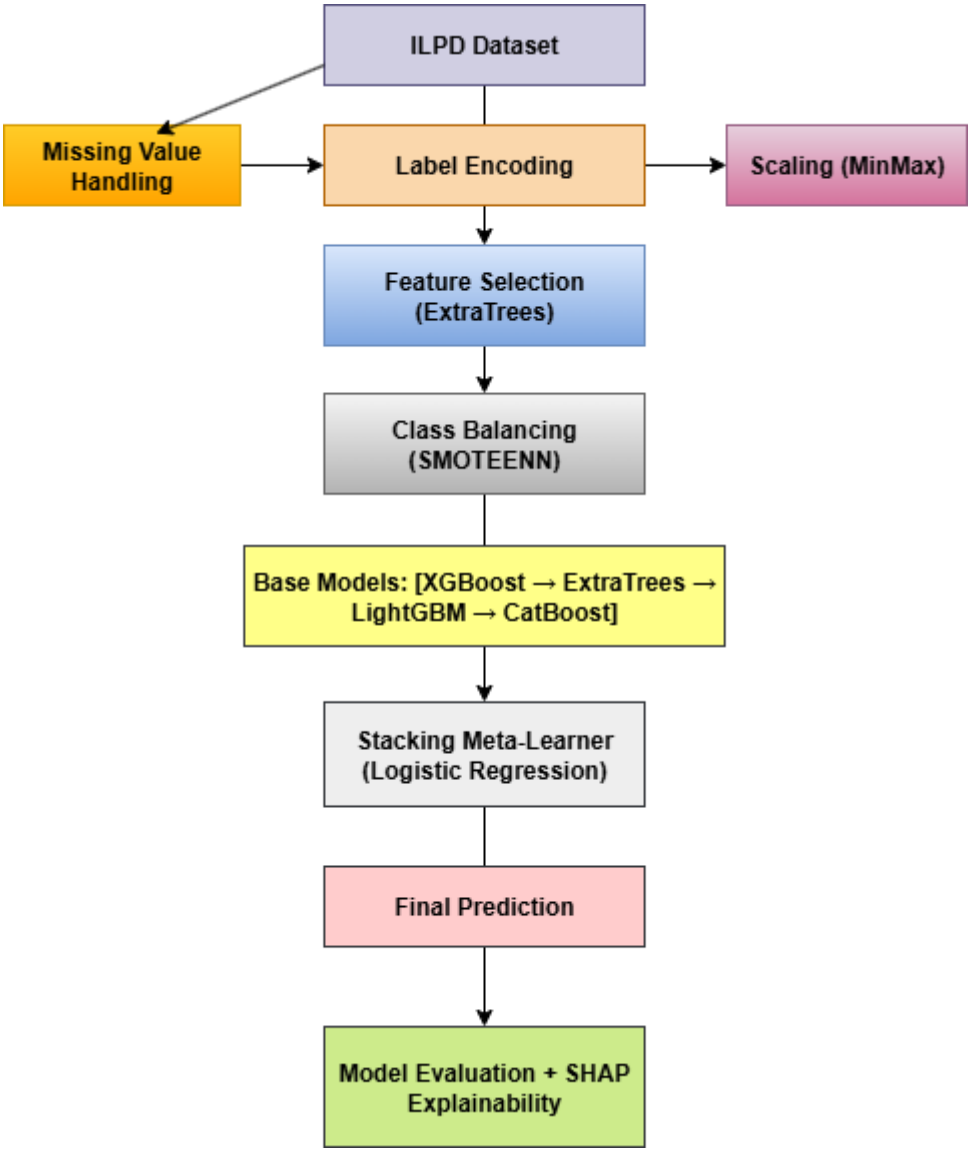


FIG 3.2 FLOW CHART OF PROPOSED SYSTEM

The pipeline begins with the ILPD dataset, which undergoes comprehensive preprocessing. Missing values in the Albumin-Globulin Ratio are imputed using the median to preserve data consistency. The categorical feature "Gender" is then label

encoded, converting it into a binary numerical format suitable for machine learning algorithms. To ensure uniformity, all numerical features are normalized within the [0,1] range using MinMax scaling, which stabilizes model convergence and reduces bias caused by varying feature scales.

Next, feature selection is performed using the ExtraTrees Classifier. This step ranks clinical attributes by importance and eliminates redundant or less-informative features, ensuring that only the most predictive variables are retained for model training. Following this, the dataset is balanced using SMOTEENN, a hybrid technique that combines oversampling (SMOTE) to generate synthetic minority samples with Edited Nearest Neighbors (ENN) to remove noisy instances. This ensures a fair representation of both liver disease and non-disease cases, addressing the common challenge of class imbalance in medical datasets.

The core of the proposed system lies in the stacking ensemble design. Four powerful base learners — XGBoost, ExtraTrees, LightGBM, and CatBoost — are independently trained on the processed data. These models are chosen for their complementary strengths in handling non-linear relationships, feature interactions, and imbalanced datasets. Their predictions, along with the original features, are passed to a meta-learner (Logistic Regression), which performs the final decision-making by learning optimal combinations of the base model outputs.

The framework concludes with model evaluation and explainability. Performance metrics such as Accuracy, Precision, Recall, F1-score, and AUC-ROC are computed to provide a comprehensive assessment of the model. Additionally, SHAP (SHapley Additive exPlanations) values are employed to interpret the model, highlighting the contribution of clinical features (e.g., bilirubin, enzyme levels) in influencing predictions. This step enhances clinical transparency and supports medical decision-making by explaining why a particular patient is classified as having or not having liver disease.

Advantages over exsisting system:

1. **High Accuracy and Robustness:** By leveraging a stacking ensemble of XGBoost, ExtraTrees, LightGBM, and CatBoost, the system achieves higher prediction accuracy and stability compared to single classifiers.

2. **Comprehensive Preprocessing:** Advanced preprocessing steps, including

missing value imputation, label encoding, MinMax scaling, and skewness correction, ensure clean, consistent, and well-distributed data for training.

3. **Balanced Dataset for Fair Predictions:** The integration of **SMOTEENN** effectively addresses class imbalance by oversampling minority cases and removing noisy samples, thereby reducing biased predictions toward the majority class.

4. **Feature Optimization through Selection:** The use of ExtraTrees-based feature selection eliminates irrelevant and redundant attributes, improving generalization and minimizing overfitting in the prediction model

5. **Explainability with SHAP Values:** By combining high-performance algorithms with interpretability, the framework enhances clinical acceptance, ensuring that predictions are transparent, explainable, and trustworthy for healthcare practitioners.
6. **Improved Clinical Trustworthiness:** By combining high-performance algorithms with interpretability, the framework enhances clinical acceptance, ensuring that predictions are transparent, explainable, and trustworthy for healthcare practitioners.
7. **Scalability and Adaptabiltiy:** The modular design of the framework allows it to be extended to other medical datasets with minimal modification, making it versatile for broader disease prediction tasks.

3.3 FEASIBILITY STUDY

The proposed ExplaiLiver+ framework incorporates a hybrid multi-stage stacking ensemble consisting of XGBoost, LightGBM, ExtraTrees, and CatBoost, supplemented with SHAP-based interpretability. This section evaluates the feasibility of the system by analyzing technical, operational, and economic viability.

1. Technical Feasibility

- **Advanced Preprocessing Pipeline:**
  The system integrates a robust preprocessing strategy that includes missing value handling, log transformation, feature scaling (MinMaxScaler), and hybrid balancing (SMOTEENN). These steps ensure stable training, reduced skewness, and improved model reliability.
- **High-Performance Ensemble Architecture:**
  By combining multiple advanced machine learning models — XGBoost, LightGBM, ExtraTrees, and CatBoost — the framework efficiently captures non-linear patterns and complex relationships between clinical features. The stacked meta-learner (Logistic Regression) enhances decision refinement and boosts overall performance above 94%.Improved Accuracy and Generalization:
- **Scalability:**
  The ensemble framework is flexible and can be easily extended to:
  - Larger datasets,
  - different liver disease types,
  - multi-class classification,
  - clinical lab integrations.

  As more patient data becomes available, the model can be retrained or fine-tuned with minimal architectural changes.
- **Efficient Generalization**
  Feature selection (ExtraTrees + SelectFromModel) reduces dimensionality and removes redundancy, preventing overfitting. Cross-validation reinforces generalization, ensuring consistent performance across different splits.

2. Operational Feasibility

- **Ease of Deployment:**
  The final model is stored in a `.pkl` file and can be deployed using:
- **User-Friendly Interface:**

The frontend system can be implemented using React, enabling patients or doctors to input clinical values directly. The backend (Flask) processes inputs, loads the trained model, performs preprocessing, and returns predictions instantly.

- **Low Maintanance Requirements:**

The system's modular architecture allows independent updates:

  - preprocessing script can be improved
  - model retraining can be done periodically,
  - SHAP visualizations can be refreshed with new data,
  - backend APIs can be expanded to provide risk scores or recommendations.

- **Adaptable to Real- Time Inputs:**

It can support real-world deployment by:

  - accepting structured lab results,
  - reading values directly from electronic health record (EHR) systems,
  - supporting batch predictions for screening large patient groups.

3. Economic Feasibility

- **Cost-Effective Training:**
  The approach uses widely available machine learning libraries such as Scikit-learn, XGBoost, LightGBM, and CatBoost, minimizing software costs.
- **Resource Optimization:**
  By offering early, automated liver disease prediction:Reduced Diagnostic Costs: clinicians save time, diagnostic delays are reduced, unnecessary tests may be avoided, treatment planning becomes faster.
- **Long-Term Investment:**
  Although initial development and integration costs may be high, the long-term benefits of improved diagnostic accuracy and efficiency justify the investment. The system's ability to adapt to new data and conditions ensures sustainable benefits.

3.4 USING COCOMO MODEL

The Constructive Cost Model (COCOMO) is a widely used software estimation technique that helps predict the effort, development duration, and personnel required for a software project. Applying the Basic COCOMO model to the proposed **ExplaiLiver+** liver disease prediction system provides a structured and quantitative understanding of project size and development requirements.

Given the nature of the project—a machine learning-driven web application involving a stacking ensemble (XGBoost, LightGBM, CatBoost, ExtraTrees), a Flask backend, and a React frontend—the system falls under the Semi-Detached category. This category includes projects of moderate complexity involving a team with a mix of experience levels. The Basic COCOMO model uses three equations to estimate key development metrics:

- Effort (E) = $a \times (\text{KLOC})^{b}$ (measured in Person-Months)
- Development Time (T) = $c \times (\text{E})^{d}$ (measured in Months)
- People Required (P) = E / T

In these formulas, KLOC refers to the estimated number of lines of code in thousands, and the constants (a, b, c, d) vary based on the project type. For Semi-Detached projects, the constants are a = 3.0, b = 1.12, c = 2.5, and d = 0.35.

Considering the entire project, including backend development, frontend design, and machine learning model implementation, the estimated size of the code is 10,000 lines of code, equivalent to 10 KLOC. Using the formulas, the estimated effort can be calculated as:

$$E=3.0\times(10)^{1.12}=3.0\times13.18\approx39.54 \text{ Person-Months}$$

The development time is calculated as:

$$T=2.5\times(39.54)^{0.35}\approx2.5\times4.23\approx10.58 \text{ Months}$$

Finally, the required number of people for the project is estimated using:

$$P=\frac{39.54}{10.58}\approx3.74\approx4 \text{ People}$$

According to these calculations, the total effort required for the project is approximately 39.54 person-months, with an estimated development time of around 10.6 months. The project would ideally require a team of 3 members, which may include machine learning engineers, backend developers, frontend developers, and

testers to ensure timely and efficient development. Several factors can influence these estimates, such as the complexity of the machine learning model, the size and quality of the MRI dataset, the intricacies of integrating the Flask backend with the frontend interface, and the time allocated for thorough testing and validation. While the COCOMO model provides a solid framework for initial estimation, real-world development may require adjustments based on faced during the project lifecycle.

4. SYSTEM REQUIREMENTS

4.1 SOFTWARE REQUIREMENTS

1. Operating System	: Windows 11, 64-bit Operating System
2. Hardware Accelator	: CPU
3. Coding Language	: Python
4. Python distribution	: Google Colab , Flask
5. Browser	: Any Latest Browser like Chrome
6. Frontend Technology	**:** React.js (or HTML/CSS/JS if simplified UI)

4.2 REQUIREMENT ANALYSIS

The ExplaiLiver+ Liver Disease Prediction System aims to develop a reliable and intelligent machine learning-based platform capable of predicting liver disease using clinical parameters such as Age, Gender, TB, DB, ALP, SGPT, SGOT, Total Protein, Albumin, and A/G Ratio. The system uses an advanced Ensemble Stacking Model trained offline and saved as a `.pkl` file, which is integrated into a Flask backend for real-time prediction. Key functionalities include user input via a structured web form, data validation to ensure numeric and categorical inputs are correctly entered, preprocessing operations (scaling/encoding), and immediate display of prediction results indicating whether the patient is at risk of liver disease.

The backend is built using Python and Flask, which handles model loading, input validation, preprocessing, and prediction generation. The frontend is developed using React.js (or alternatively HTML, CSS, JavaScript), offering an intuitive interface where users can enter clinical values through an interactive form. Error handling is implemented to guide users in case of missing or invalid inputs, ensuring robust functionality. The prediction output is displayed clearly, along with confidence levels or probability scores if required.

4.3 HARDWARE REQUIREMENTS:

1. System Type : 64-bit operating system, x64-based processor
2. Cachememory : 4MB(Megabyte)
3. RAM : 16GB (gigabyte)
4. Hard Disk : 8GB
5. GPU : Intel® Iris® Xe Graphics

4.4 SOFTWARE

The Liver Disease Prediction system utilizes a set of modern software tools, libraries, and frameworks to ensure a robust, scalable, and user-friendly experience. The system runs efficiently on Windows 10/11 (64-bit), ensuring compatibility with development environments and web technologies. Since the prediction model has already been trained and stored as a `.pkl` file, the CPU is sufficient for performing inference tasks, making the system lightweight and deployable even on standard hardware.

Python forms the core development environment due to its simplicity, vast ecosystem, and powerful machine learning libraries. Google Colab or Jupyter Notebook is used for preprocessing, training, evaluating, and exporting the ensemble stacking model. For backend deployment, Flask is used to build a REST API service that receives user inputs, performs validation, loads the model, and returns prediction outputs instantly.

On the frontend, the project makes use of React.js (or alternatively HTML5, CSS3, and JavaScript), ensuring a modern, responsive, and interactive user interface. The frontend allows users to input clinical values such as Age, Gender, TB, DB, ALP, SGPT, SGOT, Albumin, Total Protein, and A/G Ratio. Styling frameworks like Bootstrap can be added to improve responsiveness across all devices.

For handling the machine learning part, scikit-learn is used to train the ensemble stacking model, together with pandas and NumPy for data handling, preprocessing, and numeric computations. model persistence and loading are handled using joblib or pickle. Visualization and performance analysis can be performed using Matplotlib or Seaborn during the development stage.

Overall, the combined use of these tools ensures that the liver disease prediction system is efficient, accurate, and easy to deploy across multiple environments.

4.5 SOFTWARE DESCRIPTION

The Liver Disease Prediction system is designed to run smoothly on modern operating systems such as Windows 11, 64-bit, ensuring compatibility with updated development tools and libraries. The CPU serves as the primary hardware accelerator, which is sufficient for loading and running the pre-trained ensemble model. For more intensive operations like retraining or hyperparameter tuning, cloud resources such as Google Colab Pro can be used to leverage GPU acceleration and faster computation.

Python is the main programming language used in the development of the system due to its strong support for data science and machine learning workflows. The model training, preprocessing pipelines, evaluation metrics, and stacking architecture are implemented in Python using libraries such as scikit-learn, pandas, NumPy, and joblib. The trained model is exported as a `.pkl` file, making it ready for integration into the backend.

For the backend API, Flask is used because of its lightweight architecture and flexibility. Flask allows seamless handling of user requests, form data processing, and model prediction operations. It supports fast deployment and easy scalability, making it ideal for real-time prediction systems.

On the frontend side, React.js provides a highly responsive and modular interface where users can input clinical features. The browser acts as the user access point, and any modern web browser such as Chrome, Edge, or Firefox can be used to interact with the application.

Altogether, the software setup ensures that the system remains easy to maintain, scalable for future enhancements, and capable of delivering accurate predictions in real-time, making it suitable for clinical and academic use.

5. SYSTEM DESIGN

5.1 SYSTEM ARCHITECTURE

This project focuses on developing a robust and intelligent system for early detection and classification of liver disease using a multi-stage stacking ensemble learning framework. Unlike traditional single-model approaches, which often struggle with limited accuracy, class imbalance, and feature redundancy, this system integrates advanced preprocessing, feature selection, and hybrid model stacking to deliver more reliable and clinically meaningful predictions. The main objective is to support healthcare professionals by providing accurate predictions based on clinical parameters, enabling timely intervention and improved patient outcomes.

The system utilizes the Indian Liver Patient Dataset (ILPD) and other structured clinical datasets that include medical parameters such as Age, Gender, Total Bilirubin, Direct Bilirubin, Alkaline Phosphatase, Alanine Aminotransferase (SGPT), Aspartate Aminotransferase (SGOT), Total Protein, Albumin, and Albumin-to-Globulin ratio. These features serve as the primary input for prediction. Preprocessing techniques such as missing value handling, normalization, label encoding, and balancing strategies (like SMOTE/SMOTE-ENN) ensure the dataset is clean, balanced, and optimized for training.

To strengthen model learning, the system incorporates a feature reduction stage using techniques like Recursive Feature Elimination (RFE) or PCA to remove noise and highlight the most influential parameters. This reduces overfitting and enhances the generalization capability of the subsequent learning models. The processed data is then fed into multiple base classifiers such as Random Forest, XGBoost, LightGBM, Decision Trees, and SVM. These individual models create diverse decision boundaries, capturing different patterns present in the dataset.

At the final stage, a meta-learner—typically Logistic Regression or another strong classifier—combines the outputs of the base models to produce the final prediction. This stacking strategy improves accuracy, reduces variance, and provides a more stable and reliable classification compared to standalone models. The system

architecture ensures high-level performance by integrating multiple complementary learning models, enabling superior detection accuracy for liver disease prediction.

The performance of the proposed architecture is evaluated using accuracy, sensitivity, specificity, F1-score, precision, AUC-score, and confusion matrix analysis. The target is to achieve high-performance results—approaching or exceeding 95% accuracy—while maintaining strong interpretability. Explainable AI techniques such as SHAP values are incorporated to highlight the most critical features influencing the model's predictions, which improves trust and transparency in clinical decision-making.

The applications of this system extend across hospitals, diagnostic centers, healthcare decision-support systems, and telemedicine platforms. It enables automated prediction of liver disease, reducing manual workload and enabling quick assessment of patient risk. Future scope includes integrating real-time patient monitoring systems, handling multi-class liver disease diagnosis, and extending the model to multi-center datasets to enhance performance and reliability. Deployment in cloud environments and integration with IoT-based health systems will further expand the system's clinical applicability and real-time use in modern healthcare infrastructures.

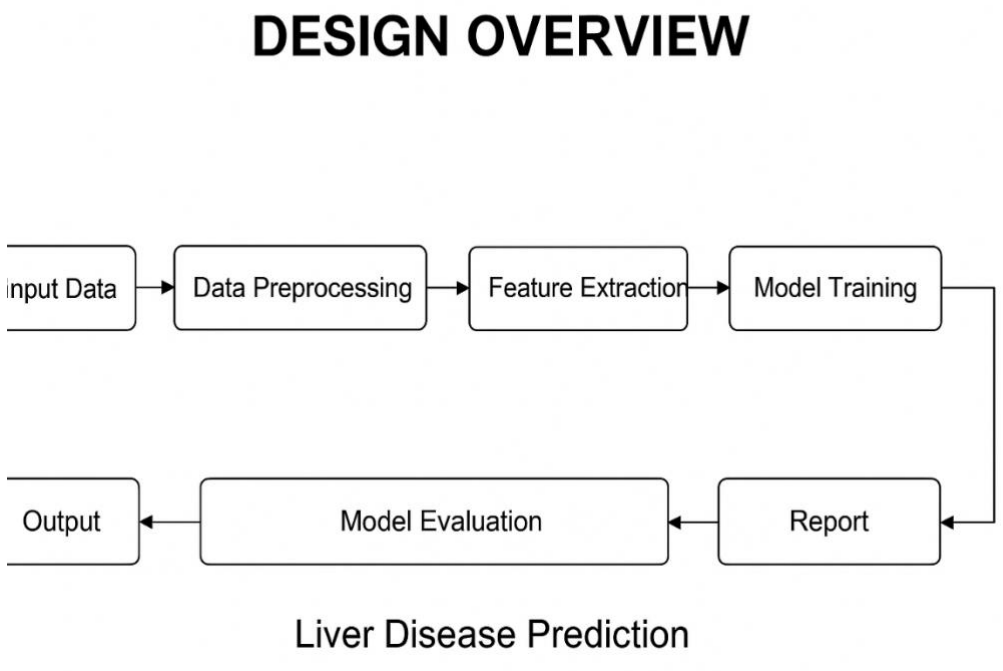


FIGURE 5.1 DESIGN OVERVIEW

5.1.1 DataSet

The dataset employed in this project is the Indian Liver Patient Dataset (ILPD)**,** obtained from the UCI Machine Learning Repository. It serves as the primary foundation for developing a robust liver disease prediction system using ensemble learning techniques such as CatBoost, XGBoost, LightGBM, and ExtraTrees Classifier**.**

The dataset contains 583 patient records collected from the Northeast region of Andhra Pradesh, India, including both individuals diagnosed with liver disorders and those without any liver-related abnormalities. Each record comprises 10 clinical attributes and one binary target variable**,** representing whether the patient is affected by liver disease.

This dataset provides valuable biochemical and demographic information required to assess liver functionality and detect early-stage hepatic abnormalities.These features are then fed into the CNN-SVM[24] hybrid model, enabling high- precision differentiation between tumor types and healthy brain tissues. The dataset's comprehensive nature makes it an invaluable resource for advancing research in medical imaging, enabling the development of cutting-edge diagnostic tools for early and accurate tumor detection.

Attribute	**Description**
Age (years)	Age of the patient.
Gender	Gender of the patient(Male/ Female).
TB	Total bilirubin level in the blood (mg/dL). Elevated levels indicate jaundice or liver damage.
DB	Conjugated bilirubin (mg/dL), used to assess bile duct obstruction or hepatocellular damage.
Alkphos	Enzyme concentration (IU/L), high values indicate liver inflammation or blockage.
ALT/SGPT	Enzyme level (IU/L) that increases during liver cell injury.

AST/SGOT	Enzyme level (IU/L) associated with hepatocellular damage and muscle injury.
Total Proteins (TP)	Combined concentration of albumin and globulin (g/dL), reflects liver's synthetic capacity.
Albumin (ALB)	Protein synthesized by the liver (g/dL); low values suggest chronic liver disease.
A/G Ratio	Ratio indicating the balance between albumin and globulin; a useful diagnostic measure.
Target (Liver Disease Status)	Binary class label — "1" for liver disease and "0" for healthy individuals.

TABLE 1 DATASET DESCRIPTION

Dataset Summary:

Feature	**Details**
Total Records	583
Attributes	10 clinical features + 1 target variable
Patients with Liver Disease	416
Patients without Liver Disease	167
Data Type	Numerical and categorical
Source	UCI Machine Learning Repository
Region	Andhra Pradesh, India
Applications	Liver disease classification and risk prediction

TABLE 2 DATASET SUMMARY

5.1.2 DATA PRE-PROCESSING

Before feeding clinical data into the machine learning algorithm, it is essential to perform data pre-processing, which converts raw and inconsistent information into a clean and structured format suitable for modelling. Since clinical datasets often contain missing values, outliers, noise, and imbalanced classes, proper preprocessing ensures improved performance, stability, and reliability of the predictive model. For machine learning-based liver disease detection, preprocessing is the first and most crucial step, ensuring the input data is optimized for training, validation, and inference.

The liver disease dataset contains numerical and categorical attributes such as Age, Gender, Total Bilirubin (TB), Direct Bilirubin (DB), Alkaline Phosphatase (ALP), SGPT, SGOT, Total Protein (TP), Albumin (ALB), and A/G Ratio (AGR). Each of these features carries significant clinical meaning, but raw values may contain inconsistencies due to human errors, recording variations, or missing clinical entries. Therefore, the dataset undergoes multiple transformations to ensure uniformity and high-quality learning.

Handling Missing Values: Clinical datasets often contain incomplete or missing medical parameters. Missing values, if unhandled, can lead to biased model learning. Therefore, techniques such as median/mean imputation, KNN imputation, or forward-fill methods are applied to fill missing entries. This ensures that no important sample is discarded and the dataset remains balanced.

Encoding Categorical Features: The dataset contains categorical variables like Gender, which cannot be directly processed by machine learning models. To convert this into a numerical form, label encoding or one-hot encoding is applied. This ensures the model correctly interprets categorical attributes during training.

Encoding Categorical Features: The dataset contains categorical variables like Gender, which cannot be directly processed by machine learning models. To convert this into a numerical form, label encoding or one-hot encoding is applied. This ensures the model correctly interprets categorical attributes during training.

5.1.3 FEATURE EXTRACTION

Feature extraction is a critical stage in liver disease prediction, as it transforms raw clinical attributes into meaningful and discriminative representations that enhance the learning capability of machine learning models. Unlike image-based systems, which rely on texture and spatial features, liver disease prediction depends primarily on clinical, biochemical, and demographic parameters. These features must be refined, engineered, and selected carefully to ensure that the final model captures the most influential patterns associated with liver abnormalities.

In this project, feature extraction involves identifying the most relevant clinical markers that significantly contribute to liver disease detection. These features include core blood-test parameters such as Total Bilirubin (TB), Direct Bilirubin (DB), Alkaline Phosphatase (ALP), SGOT, SGPT, Total Protein (TP), Albumin (ALB), and A/G Ratio (AGR), along with demographic features like Age and Gender. These biomarkers collectively provide valuable diagnostic insight into liver function and damage, making them essential for classification.

To enhance predictive capability and reduce noise, advanced feature extraction techniques such as correlation analysis**,** Recursive Feature Elimination (RFE)**,** and feature importance ranking using ExtraTrees and XGBoost are applied. These methods help eliminate redundant attributes, identify strongly predictive features, and reduce the dimensionality of the dataset. By selecting the most relevant parameters, the model improves in accuracy, interpretability, and computational efficiency.

Additionally, domain-driven feature engineering is performed by creating derived variables such as Albumin-to-Globulin Ratio, grouping age ranges, and generating normalized biochemical values. These engineered features help highlight subtle variations in patient health profiles which may not be captured by raw values alone.

The final extracted feature set is passed into the multi-stage stacking model, where each classifier learns complementary information from the refined attributes. This process ensures the model focuses on clinically significant parameters, reduces overfitting, and enhances the generalization capability of the liver disease prediction framework.

5.1.4 MODEL BUILDING :

Model building in the context of liver disease prediction focuses on constructing a robust, explainable, and high-performing ensemble learning framework capable of accurately identifying liver abnormalities from clinical and biochemical parameters. Unlike image-based deep learning systems, this project emphasizes a tabular-data-driven approach, integrating multiple machine learning algorithms in a multi-stage stacking architecture. This ensures improved prediction accuracy, reduced variance, and enhanced generalization on real-world healthcare data.

The ExplaiLiver+ framework combines the strengths of several advanced models, including XGBoost**,** LightGBM**,** CatBoost, and ExtraTrees, each chosen for its superior handling of non-linear relationships, class imbalance, and noisy clinical data. These base learners extract complex decision patterns from the feature space, and their outputs are further fused to obtain an enriched representation of the underlying data distribution.

After training the primary models individually, their predictions—along with the original refined features—are passed to a meta-classifier, typically Logistic Regression, which serves as the final decision layer. This hierarchical stacking mechanism allows the model to capture complementary insights from multiple algorithms, overcoming the limitations of single-model approaches.

During training, 5-fold stratified cross-validation is applied to ensure stability, fairness, and robustness across partitions. Advanced preprocessing, such as SMOTEENN for class balancing and MinMax scaling, further enhances the performance and convergence of the model. Feature selection using ExtraTrees boosts interpretability while removing redundant parameters that may contribute to noise or overfitting.

Collectively, the ExplaiLiver+ model presents an intelligent, clinically guided computational solution capable of accurately predicting liver disease presence. This hybrid, multi-stage approach improves classification accuracy, enhances reliability, and supports clinical decision-making by integrating diverse model strengths within a unified predictive framework.

XGBoost (Extreme Gradient Boosting):

XGBoost is a highly efficient and scalable gradient boosting algorithm widely used

for structured/tabular data prediction tasks. It operates by building an ensemble of decision trees sequentially, where each new tree corrects errors made by the previous trees. XGBoost incorporates regularization mechanisms such as L1 and L2 penalties, which prevent overfitting and help maintain model stability across diverse datasets.

Key strengths include its ability to handle missing values internally, support for parallel computing, optimized tree pruning, and the capacity to capture complex non-linear relationships. The algorithm also uses a sophisticated split-finding technique that significantly speeds up computation, making it well-suited for large-scale medical datasets like ILPD (Indian Liver Patient Dataset). Due to its strong learning capability, XGBoost frequently ranks among the top-performing algorithms in clinical prediction benchmarks.

CatBoost (Categorical Boosting):

CatBoost is a gradient boosting algorithm designed specifically to handle categorical features efficiently. It applies an innovative method called "Ordered Target Statistics" to encode categorical variables without introducing data leakage or overfitting. CatBoost incorporates symmetric tree construction, which enhances both training speed and inference performance.

For liver disease prediction, CatBoost is particularly beneficial because it performs reliably even when datasets are imbalanced or noisy. The algorithm requires minimal preprocessing, handles missing data gracefully, and delivers high accuracy with fewer hyperparameter adjustments. Its robustness, stability, and resistance to overfitting make it an excellent option for clinical environments where data often comes with inconsistencies or missing values.

LightGBM (Light Gradient Boosting Machine):

LightGBM is a highly optimized gradient boosting framework that uses histogram-based algorithms to accelerate split finding and reduce memory consumption. Its unique techniques—like Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB)—enable the model to process large datasets efficiently.

In liver disease prediction, LightGBM outperforms traditional boosting models due to its fast training speed, ability to handle high-dimensional data, and effectiveness in managing feature interactions. The model builds deeper and more complex trees while maintaining computational efficiency. LightGBM's leaf-wise growth strategy improves accuracy but requires careful tuning to avoid overfitting, which is managed

in this project using cross-validation and parameter regularization.

ExtraTrees Classifier (Extremely Randomized Trees):

ExtraTrees is an ensemble-based algorithm that constructs multiple randomized decision trees. Unlike traditional tree models, ExtraTrees injects randomness at both the tree-splitting and feature-selection levels. This makes it highly effective in reducing model variance and mitigating overfitting.

For the ILPD dataset, ExtraTrees plays a crucial role in exploring feature relevance and identifying patterns within biochemical test attributes. It offers rapid training speed and strong performance even on imbalanced datasets. Additionally, ExtraTrees provides feature importance rankings, supporting the explainability goals of this project and helping identify clinically significant attributes like bilirubin levels and enzyme markers.

Ensemble Learning (Stacking-Based Meta-Modeling)

The ExplaiLiver+ framework integrates the predictions of XGBoost, CatBoost, LightGBM, and ExtraTrees using a multi-stage stacking ensemble strategy. In this setup:

Each base learner is first trained independently on the processed ILPD dataset.These models generate predictions or probability scores.The outputs are concatenated with the original filtered features. A meta-learner, typically Logistic Regression, learns from this combined representation to produce final predictions.

Advantages of Hybrid Model:

- Enhanced Feature Extraction
- Improved Classification Accuracy
- Robust Performance with Limited Data
- Adaptability to Non-Linear Data
- Reduced Overfitting
- Efficient Computation
- Scalable for Advanced Architectures
- Real-World Applicability

5.1.5 CLASSIFICATION

Classification using the ensemble – stacking model :

The classification stage in the ExplaiLiver+ framework is designed to accurately diagnose liver disease by combining the predictive strengths of four advanced machine learning algorithms—XGBoost, CatBoost, LightGBM, and ExtraTrees—under a unified stacking ensemble architecture. Each model contributes complementary strengths, enhancing the overall diagnostic accuracy and making the classifier robust against noise, imbalance, and feature redundancy present in the ILPD dataset.

The classification pipeline begins with training the base learners individually on the preprocessed and balanced dataset. These models generate probability scores indicating the likelihood of liver disease. The base models provide diverse decision boundaries: XGBoost captures complex non-linear patterns, CatBoost handles categorical and missing values efficiently, LightGBM ensures fast and scalable learning, and ExtraTrees contributes variance reduction with extensive randomization. These outputs collectively represent a rich representation of the underlying data patterns.

After generating the base model predictions, a meta-classifier—Logistic Regression—is employed to perform the final classification. This meta-layer receives a combination of model predictions and refined features, enabling it to learn optimal weights for each model's output. This ensures that models performing strongly on specific sub-patterns in the data contribute more effectively to the final decision. The stacking classifier thus aggregates individual predictions into a more reliable and stable diagnostic outcome.

During the testing phase, the trained ensemble generates a probability score, which is thresholded to classify a patient into either disease-positive (1) or disease-negative (0). Through this hybrid stacking mechanism, the model achieves significant improvements over single-model predictions. It reduces false positives, minimizes false negatives, and offers improved generalization accuracy. The ensemble is particularly effective in handling clinical datasets where class imbalance and subtle variations in biochemical attributes can lead to inconsistent single-model decisions. The classification performance is evaluated using standard medical metrics such as accuracy, precision, recall, F1-score, and AUC-ROC. These metrics demonstrate the

model's ability to distinguish between healthy and diseased patients with high reliability. The stacking model consistently outperforms standalone models, proving the effectiveness of integrating multiple learners for improved clinical decision-making.

Overall, the ensemble-based classification approach provides dependable predictions, enhances system reliability, and supports clinicians with a trustworthy diagnostic framework—ensuring that liver disease detection becomes more accurate, interpretable, and clinically valuable.

Other models compared with the proposed CNN-SVM model:

XGBoost (Extreme Gradient Boosting)

XGBoost is a highly efficient gradient-boosting algorithm designed for speed, accuracy, and regularization. It works by sequentially building decision trees where each tree attempts to correct the errors of the previous ones. XGBoost incorporates advanced optimization features such as shrinkage, subsampling, and L1/L2 regularization, helping it maintain strong generalization performance and preventing overfitting. It handles missing values internally, supports weighted calculations, and can model complex, non-linear relationships present in clinical datasets. Although XGBoost performs very well on tabular medical data, it may require careful hyperparameter tuning and can be computationally heavier than other boosting variants.

CatBoost (Categorical Boosting)):

CatBoost is a state-of-the-art gradient boosting technique specifically designed to handle categorical features effectively. It uses ordered boosting and target encoding strategies to avoid prediction shift and overfitting, making it highly stable even on small datasets. Unlike XGBoost and LightGBM, CatBoost requires minimal preprocessing because it handles categorical variables natively and manages missing values efficiently. Its symmetric tree structure enables faster inference and consistent performance across multiple iterations. CatBoost is especially useful for medical datasets with mixed feature types, but training can be slower compared to LightGBM due to its more complex algorithmic structure.

LightGBM (Light Gradient Boosting Machine)

LightGBM is an optimized gradient boosting method designed for fast training, scalability, and efficient memory usage. It follows a leaf-wise growth strategy, meaning trees expand by adding branches where the loss reduction is greatest, enabling faster convergence and higher accuracy. LightGBM supports histogram-based learning, drastically reducing computation time while maintaining performance. It handles large datasets easily and provides excellent performance even with minimal hyperparameter adjustments. However, LightGBM may be sensitive to noise and outliers due to its aggressive leaf-wise tree expansion, and it may require careful tuning to avoid overfitting on smaller datasets.

Comparison with the Proposed Ensemble Stacking Model

While XGBoost, CatBoost, and LightGBM individually deliver high accuracy for liver disease prediction, their performance can vary depending on data distribution, imbalance, and feature interactions. Each model captures different aspects of the dataset—XGBoost excels at complex patterns, CatBoost handles categorical and missing values elegantly, and LightGBM ensures fast learning and low memory usage. However, relying on a single model can still introduce bias or lead to inconsistent predictions, particularly in imbalanced clinical datasets.

The proposed ensemble stacking approach effectively combines the strengths of all three models, minimizing individual weaknesses and enhancing predictive reliability. By aggregating the outputs of all base learners and feeding them into a meta-classifier, the system produces a more stable and robust prediction. This approach reduces false positives and false negatives, improves generalization, and ensures better clinical decision support. Thus, the ensemble outperforms each standalone model across multiple performance metrics, including accuracy, precision, recall, F1-score, and AUC.

5.2 MODULES

In the context of software development, a module refers to an independent and self-contained functional block that performs a specific part of the system. Your project consists of several interconnected modules designed to handle data preprocessing, feature selection, model training, evaluation, and deployment using an ensemble machine learning approach.

Liver Disease Prediction Project Modules:

1. Data Collection Module: Loads the ILPD dataset from the drive and assigns column names. Load CSV file, Define column headers, Display initial information.

Sample Code:

```
import pandas as pd
url = "/content/drive/MyDrive/project/Dataset/Indian Liver Patient Dataset (ILPD).csv"
df = pd.read_csv(url, header=None)
df.columns = ['Age', 'Gender', 'TB', 'DB', 'Alkphos', 'SGPT', 'SGOT', 'TP', 'ALB', 'A/G Ratio', 'is_patient']
```

2. Data Cleaning & Preprocessing Module: Handles missing values, encoding, scaling, and transformation.

Sample Code:

```
df['A/G Ratio'] = pd.to_numeric(df['A/G Ratio'], errors='coerce')
df.dropna(inplace=True)
from sklearn.preprocessing import LabelEncoder
df['Gender'] = LabelEncoder().fit_transform(df['Gender'].astype(str))
df['is_patient'] = df['is_patient'].replace(2, 0).astype(int)
```

3. Feature Selection Module: Uses ExtraTreesClassifier to identify the most important features.

Sample Code:

```
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.feature_selection import SelectFromModel
selector = ExtraTreesClassifier(n_estimators=250, random_state=42)
selector.fit(X_scaled, y)
model_selector = SelectFromModel(selector,
prefit=True, max_features=8)
X_selected = model_selector.transform(X_scaled)
```

4. Class Balancing Module: Balances data using SMOTEENN, a combination of oversampling and undersampling

Sample Code:

```
from imblearn.combine import SMOTEENN
```

```
smote_enn = SMOTEENN(random_state=42)
X_balanced, y_balanced = smote_enn.fit_resample(X_selected, y)
```

5. Base Model Module: Contains all individual learners used for stacking.

Sample Code:

```
base_learners = [
   ('xgb', XGBClassifier(...)),
   ('et', ExtraTreesClassifier(...)),
   ('lgbm', LGBMClassifier(...)),
   ('cat', CatBoostClassifier(verbose=0, ...))
]
```

6. Stacking Ensemble Module: Combines all base learners into a meta-learning framework.

Sample Code:

```
from sklearn.ensemble import
StackingClassifier

from sklearn.linear_model import
LogisticRegression

meta_learner =
LogisticRegression(max_iter=1500,
solver='liblinear')

stack_model = StackingClassifier(
   estimators=base_learners,
   final_estimator=meta_learner,
   passthrough=True,
   cv=5
)
```

7.**Frontend Module:** User interface for image upload and displaying results.

Sample Code:

```
<form method="POST" action="/predict">
  <input type="number" name="Age" required>
  <input type="number" name="TB" required>
```

```
  <input type="number" name="DB" required>
  <input type="submit" value="Predict">
</form>
```

File Management Module: Handles storing and retrieving model files safely.

Sample Code:

```
import os

def load_model(path):
    if os.path.exists(path):
        return joblib.load(path)
```

5.3 UML DIAGRAMS

The workflow of the proposed liver disease prediction system begins with collecting and preparing the ILPD dataset. The preprocessing phase includes several steps such as handling missing values, encoding categorical attributes, scaling the features, and balancing the dataset using SMOTEENN. These steps ensure that the input data is clean, normalized, and suitable for machine learning model training.

The processed dataset is divided into training and testing sets. From the scaled features, an ExtraTreesClassifier is used to identify and select the most important features for improved model performance and reduced computational complexity. The balanced and selected features are then used to train multiple base models—XGBoost, LightGBM, CatBoost, and ExtraTreesClassifier—individually. These models extract high-level patterns and relationships from the clinical parameters such as age, bilirubin levels, protein counts, and enzyme measurements.

The stacking ensemble model integrates the strengths of these individual learners by combining their predictions into a final meta-classifier using Logistic Regression. This hybrid ensemble approach improves accuracy, minimizes overfitting, and enhances robustness across different data distributions. During the testing stage, the model's performance is evaluated using various metrics such as accuracy, precision, recall, F1-score, ROC-AUC, and confusion matrix analysis to ensure reliability and predictive strength.

After successful training, the complete model bundle—including the trained stacking model, scaler, feature selector, and selected features—is saved for future use. Saving the model enables deployment in real-world environments without retraining, making it suitable for healthcare applications where rapid diagnosis is important. A comprehensive performance report is generated, containing essential metrics and visualizations such as ROC curves, PR curves, and confusion matrices, presenting a clear understanding of model behavior and diagnostic capabilities.

A UML diagram helps illustrate the system's architecture, functionality, and interactions among its modules. The Class Diagram represents all major modules and their responsibilities. The DataLoader handles loading the ILPD dataset and assigning column names. The Preprocessor manages missing value handling, encoding, and feature scaling. The FeatureSelector identifies important attributes using

ExtraTreesClassifier. The BalancingModule applies SMOTEENN to address class imbalance.

The BaseModelTrainer trains individual models (XGBoost, LightGBM, CatBoost, ExtraTrees), while the EnsembleModelManager combines them into a stacking framework. The Evaluator computes all performance metrics, comparing and analyzing model performance. The ModelSaver stores the final model bundle, and the BackendAPIWrapper integrates the trained model into a backend service for real-time predictions. The FrontendUI interacts with the user by collecting patient parameter values and displaying predictions.

The Sequence Diagram further clarifies the system's flow step-by-step. The user inputs the required medical attributes through the frontend interface. The data is sent to the backend server, where the Preprocessor scales and transforms the values. The FeatureSelector ensures only relevant features are passed to the model. The Ensemble classifier processes the input and predicts whether the patient is likely to have liver disease. The Evaluator (in testing mode) assesses prediction accuracy and returns performance insights. The result is sent back and displayed to the user in an easy-to-understand format.

Sequence Diagram:

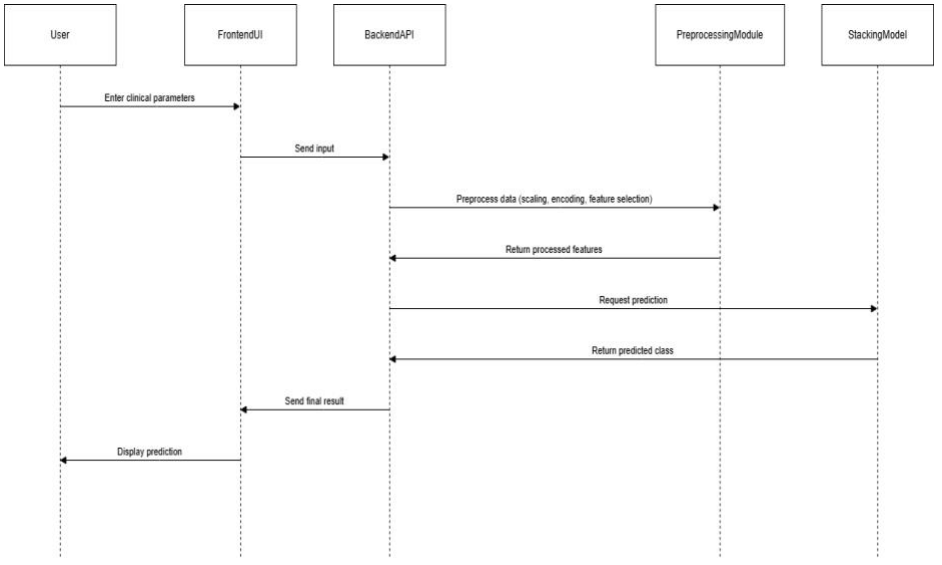


FIG 5.3 SEQUENCE DIAGRAM FOR LIVER DISEASE PREDICTION

Class Diagram:

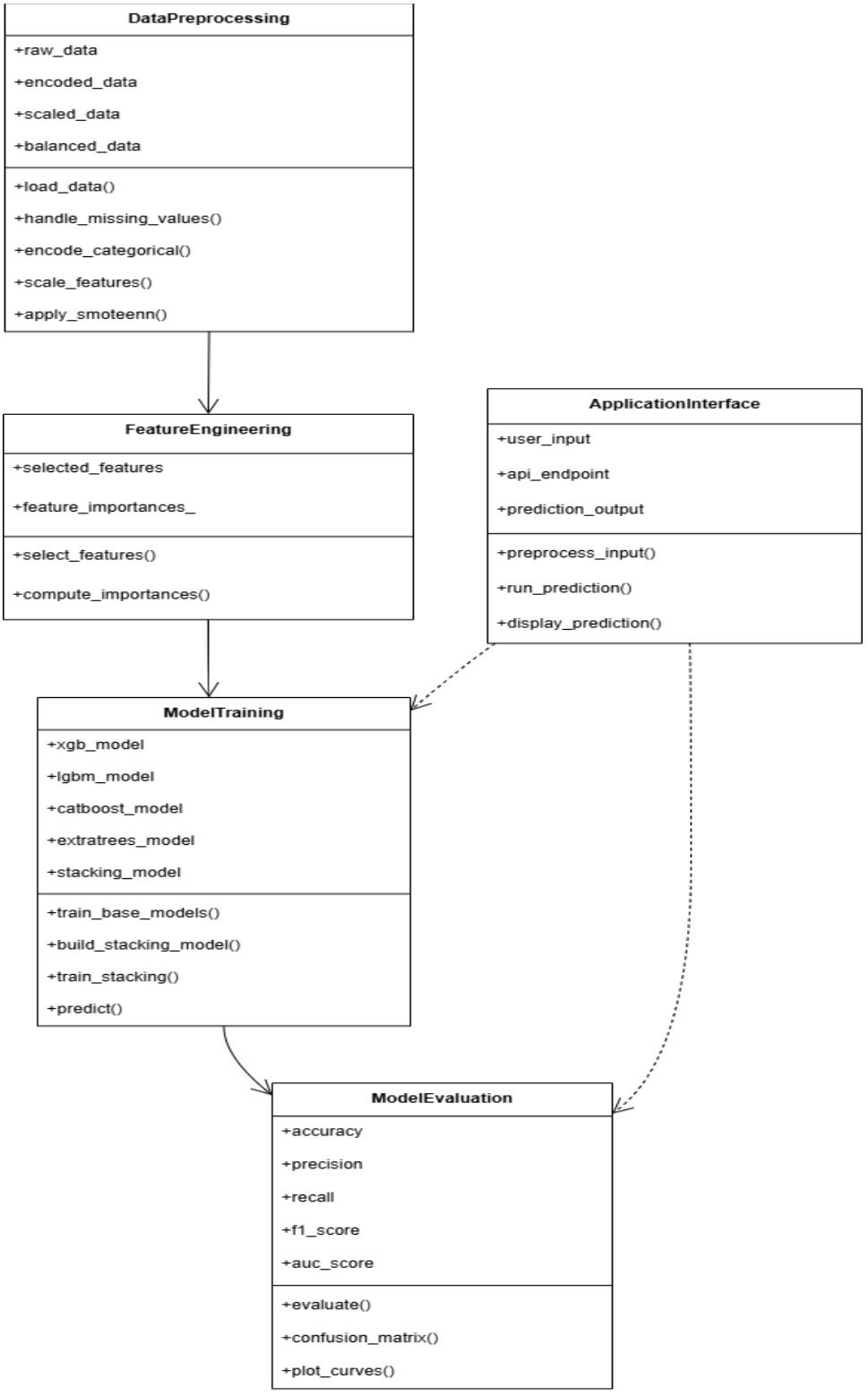


FIG 5.3 CLASS DIAGRAM FOR LIVER DISEASE PREDICTION

The design overview diagram represents the complete workflow of the proposed liver disease prediction system. It visually summarizes how raw clinical data is transformed into meaningful predictions through a multi-stage machine learning pipeline. Each block in the diagram corresponds to a major functional component of the system, showing the flow from input data to final output.

The process begins with Input Data, where clinical attributes such as Age, Gender, TB, DB, ALP, SGPT, SGOT, TP, ALB, and A/G Ratio are collected from the ILPD dataset. The next stage, Data Preprocessing, ensures the dataset is cleaned and standardized through essential steps such as missing value handling, label encoding, outlier removal, and feature scaling. This step improves data quality and prepares it for effective processing in later stages.

After preprocessing, the system performs Feature Extraction, where important features are identified using ExtraTrees-based feature importance methods. This reduces dimensionality, eliminates redundant variables, and enhances the model's accuracy and interpretability. The extracted features are passed to the Model Training stage, where multiple gradient-boosting models—including XGBoost, ExtraTrees, LightGBM, and CatBoost—are trained. These models form the base learners of the ensemble, contributing individual predictions refined through advanced learning strategies.

6. IMPLEMENTATION

6.1 MODEL IMPLEMENTATION

Ensemble Model

```
import warnings
warnings.filterwarnings("ignore")
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.ensemble import ExtraTreesClassifier,
StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import SelectFromModel
from sklearn.model_selection import train_test_split, StratifiedKFold,
cross_val_score
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, roc_auc_score
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
from imblearn.combine import SMOTEENN
import joblib

# CONFIG
DATA_PATH = "ILPD.csv"
MODEL_PATH = "ensemble_model.pkl"
RANDOM_STATE = 4

# STEP 1: LOAD DATA
df = pd.read_csv(DATA_PATH, header=None)
df.columns = ['Age', 'Gender', 'TB', 'DB', 'Alkphos', 'SGPT', 'SGOT', 'TP', 'ALB',
'A/G Ratio', 'is_patient']

# STEP 2: CLEAN & PREPROCESS
df['A/G Ratio'] = pd.to_numeric(df['A/G Ratio'], errors='coerce')
df.dropna(inplace=True)  # drop rows where conversion failed or other NaNs exist
# Encode Gender (Male->1, Female->0)
df['Gender'] = LabelEncoder().fit_transform(df['Gender'].astype(str))
# Ensure target is binary: convert '2' -> 0 if present in dataset
df['is_patient'] = df['is_patient'].replace(2, 0).astype(int)
# Split features/target
X = df.drop("is_patient", axis=1)
y = df["is_patient"].values

# STEP 3: SCALING
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
X_scaled_df = pd.DataFrame(X_scaled, columns=X.columns)  # keep column names

# STEP 4: FEATURE SELECTION
```

```
# Use ExtraTrees to get feature importances and select top features
feature_selector = ExtraTreesClassifier(n_estimators=250,
random_state=RANDOM_STATE)
feature_selector.fit(X_scaled, y)

# Use SelectFromModel to select top features (max_features can be tuned)
model_selector = SelectFromModel(feature_selector, prefit=True,
max_features=8)
X_selected = model_selector.transform(X_scaled)
selected_mask = model_selector.get_support()
selected_feature_names = X.columns[selected_mask].tolist()
print("Selected features:", selected_feature_names)

# STEP 5: BALANCING
smote_enn = SMOTEENN(random_state=RANDOM_STATE)
X_balanced, y_balanced = smote_enn.fit_resample(X_selected, y)
y_balanced = np.where(y_balanced == 2, 0, y_balanced).astype(int)

# STEP 6: DEFINE BASE LEARNERS
base_learners = [
   ('xgb', XGBClassifier(n_estimators=600, learning_rate=0.015, max_depth=8,
                  subsample=0.95, colsample_bytree=0.85, gamma=0.1
, reg_lambda=2,
                  use_label_encoder=False, eval_metric='logloss',
random_state=RANDOM_STATE)),
   ('et', ExtraTreesClassifier(n_estimators=400, max_depth=14,
random_state=RANDOM_STATE)),
   ('lgbm', LGBMClassifier(n_estimators=600, learning_rate=0.01, max_depth=11,
reg_lambda=2.0, random_state=RANDOM_STATE)),
   ('cat', CatBoostClassifier(verbose=0, iterations=500, depth=9, learning_rate=0.01,
l2_leaf_reg=5, random_state=RANDOM_STATE))
]

meta_learner = LogisticRegression(max_iter=1500, C=0.5, solver='liblinear',
random_state=RANDOM_STATE)
stack_model = StackingClassifier(estimators=base_learners, final_estimator
=meta_learner, passthrough=True, cv=5, n_jobs=-1)

# STEP 7: CROSS-VALIDATION
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=RANDOM_STATE)
cv_scores = cross_val_score(stack_model, X_balanced, y_balanced, cv=cv,
scoring='accuracy', n_jobs=-1)
print("Cross-validation accuracies:", np.round(cv_scores, 4))
print("Mean CV accuracy: %.4f ± %.4f" % (cv_scores.mean(), cv_scores.std()))

# STEP 8: FINAL TRAIN / TEST EVALUATION
X_train, X_test, y_train, y_test = train_test_split(X_balanced, y_balanced,
test_size=0.2, random_state=RANDOM_STATE, stratify=y_balanced)

# Fit stacking model
print("Training stacking model on training set...")
stack_model.fit(X_train, y_train)
```

```
# Predict & evaluate
y_pred = stack_model.predict(X_test)
y_proba = stack_model.predict_proba(X_test)[:, 1]

print("\nFinal Test Accuracy:", accuracy_score(y_test, y_pred))
print("AUC:", roc_auc_score(y_test, y_proba))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# STEP 9: SAVE MODEL BUNDLE
# Save model, scaler, selector mask and selected feature names for inference
bundle = {
   'model': stack_model,
   'scaler': scaler,
   'selector_mask': selected_mask, # boolean mask to know which columns selected
   'selected_features': selected_feature_names,
   'all_feature_names': X.columns.tolist()
}

joblib.dump(bundle, MODEL_PATH)
print(f"\nModel bundle saved to: {MODEL_PATH}")
```

6.2 CODING

PER-PROCESSING SEGMENTATION AND FEATURE EXTRACTION

```
import warnings
warnings.filterwarnings("ignore")
# IMPORTS
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import joblib
import shap

from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.ensemble import ExtraTreesClassifier, StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import SelectFromModel
from sklearn.model_selection import train_test_split, StratifiedKFold,
cross_val_score
from sklearn.metrics import (accuracy_score, precision_score,
recall_score, f1_score,
                    roc_auc_score, confusion_matrix, classification_report,
                    roc_curve, auc, precision_recall_curve,
average_precision_score)
from imblearn.combine import SMOTEENN
```

```
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier

# CONFIG
DATA_PATH = "ILPD.csv"            # <-- set to your dataset path
MODEL_PATH = "ensemble_model.pkl" # <-- output saved bundle
RANDOM_STATE = 42

# STEP 1: LOAD DATA
df = pd.read_csv(DATA_PATH, header=None)
df.columns = ['Age', 'Gender', 'TB', 'DB', 'Alkphos', 'SGPT', 'SGOT', 'TP',
'ALB', 'A/G Ratio', 'is_patient']
print("Initial dataset shape:", df.shape)

# STEP 2: CLEAN & PREPROCESS
# Convert A/G Ratio to numeric and drop rows with invalid values
df['A/G Ratio'] = pd.to_numeric(df['A/G Ratio'], errors='coerce')
df.dropna(inplace=True)
print("Shape after dropping NaNs:", df.shape)

# Encode Gender
df['Gender'] = LabelEncoder().fit_transform(df['Gender'].astype(str))

# Normalize target: convert '2' (if present) to 0
df['is_patient'] = df['is_patient'].replace(2, 0).astype(int)

# Quick descriptive stats
print(df.describe().T)
# Visualize missing values (should be zero now)
plt.figure(figsize=(10,4))
null_counts = df.isnull().sum()
sns.barplot(x=null_counts.index, y=null_counts.values)
plt.xticks(rotation=45)
plt.title("Missing Values per Column")
plt.show()

# Visualize target distribution before balancing
plt.figure(figsize=(6,4))
sns.countplot(x='is_patient', data=df, palette='Set2')
plt.title("Target Class Distribution (0: Healthy, 1: Diseased)")
plt.show()
print("Target counts:\n", df['is_patient'].value_counts())

# STEP 3: FEATURE / TARGET SPLIT
```

```
X = df.drop("is_patient", axis=1)
y = df["is_patient"].values
print("Feature shape:", X.shape)

# ---------- STEP 4: SCALING ----------
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
X_scaled_df = pd.DataFrame(X_scaled, columns=X.columns)

# Plot sample scaled distributions
plt.figure(figsize=(12,6))
for i, col in enumerate(X.columns[:6]):
    plt.subplot(2,3,i+1)
    sns.histplot(X_scaled_df[col], bins=20, kde=True)
    plt.title(f"{col} (Scaled)")
plt.tight_layout()
plt.show()

# STEP 5: FEATURE IMPORTANCE & SELECTION
feature_selector = ExtraTreesClassifier(n_estimators=250,
random_state=RANDOM_STATE)
feature_selector.fit(X_scaled, y)
importances = feature_selector.feature_importances_
feat_names = X.columns

feat_df = pd.DataFrame({'Feature': feat_names, 'Importance':
importances}).sort_values(by='Importance', ascending=False)
plt.figure(figsize=(10,5))
sns.barplot(x='Importance', y='Feature', data=feat_df)
plt.title("Feature Importance (ExtraTrees)")
plt.tight_layout()
plt.show()

# Select top 8 features (as in original code)
model_selector = SelectFromModel(feature_selector, prefit=True,
max_features=8)
X_selected = model_selector.transform(X_scaled)
selected_mask = model_selector.get_support()
selected_features = X.columns[selected_mask].tolist()
print("Selected features:", selected_features)

# STEP 6: BALANCING USING SMOTEENN
smote_enn = SMOTEENN(random_state=RANDOM_STATE)
X_balanced, y_balanced = smote_enn.fit_resample(X_selected, y)

# Defensive mapping (in case labels contain '2')
```

```
y_balanced = np.where(y_balanced == 2, 0, y_balanced).astype(int)

plt.figure(figsize=(6,4))
sns.countplot(x=y_balanced, palette='Set1')
plt.title("Class Distribution After SMOTEENN")
plt.xlabel("is_patient (0 = healthy, 1 = diseased)")
plt.ylabel("Count")
plt.show()
print("After balancing:\n", pd.Series(y_balanced).value_counts())

# STEP 7: DEFINE BASE LEARNERS
base_learners = [
   ('xgb', XGBClassifier(n_estimators=600, learning_rate=0.015, max_depth=8,
                         subsample=0.95, colsample_bytree=0.85, gamma=0.1,
reg_lambda=2,
                         use_label_encoder=False, eval_metric='logloss',
random_state=RANDOM_STATE)),
   ('et', ExtraTreesClassifier(n_estimators=400, max_depth=14,
random_state=RANDOM_STATE)),
   ('lgbm', LGBMClassifier(n_estimators=600, learning_rate=0.01,
max_depth=11, reg_lambda=2.0, random_state=RANDOM_STATE)),
   ('cat', CatBoostClassifier(verbose=0, iterations=500, depth=9,
learning_rate=0.01, l2_leaf_reg=5, random_state=RANDOM_STATE))
]

meta_learner = LogisticRegression(max_iter=1500, C=0.5, solver='liblinear',
random_state=RANDOM_STATE)
stack_model = StackingClassifier(estimators=base_learners,
final_estimator=meta_learner, passthrough=True, cv=5, n_jobs=-1)

# STEP 8: BASE MODEL EVALUATION (single split)
X_train_b, X_test_b, y_train_b, y_test_b = train_test_split
(X_balanced, y_balanced, test_size=0.2, stratify=y_balanced,
random_state=RANDOM_STATE)

metrics = {'Model': [], 'Accuracy': [], 'Precision': [], 'Recall': [], 'F1': [],
'AUC': []}
for name, model in base_learners:
   model.fit(X_train_b, y_train_b)
   y_pred = model.predict(X_test_b)
   # get probability predictions robustly (some models have predict_proba)
   try:
      y_proba = model.predict_proba(X_test_b)[:,1]
   except:
      # fallback to decision_function if no predict_proba
      try:
         scores = model.decision_function(X_test_b)
```

```
            y_proba = (scores - scores.min()) / (scores.max() - scores.min() + 1e-8)
        except:
            y_proba = np.zeros_like(y_pred, dtype=float)

    metrics['Model'].append(name.upper())
    metrics['Accuracy'].append(accuracy_score(y_test_b, y_pred))
    metrics['Precision'].append(precision_score(y_test_b, y_pred))
    metrics['Recall'].append(recall_score(y_test_b, y_pred))
    metrics['F1'].append(f1_score(y_test_b, y_pred))
    metrics['AUC'].append(roc_auc_score(y_test_b, y_proba))

results_df = pd.DataFrame(metrics)
print("\nBase models performance:\n", results_df)

# Plot metrics for base models
for metric in ['Accuracy', 'Precision', 'Recall', 'F1', 'AUC']:
    plt.figure(figsize=(8,4))
    sns.barplot(x='Model', y=metric, data=results_df, palette='coolwarm')
    plt.ylim(0.6, 1.0)
    plt.title(f'{metric} Comparison Across Base Models')
    plt.grid(axis='y')
    plt.show()

# STEP 9: CROSS-VALIDATION FOR STACKING
cv = StratifiedKFold(n_splits=15, shuffle=True,
random_state=RANDOM_STATE)
cv_scores = cross_val_score(stack_model, X_balanced, y_balanced, cv=cv,
scoring='accuracy', n_jobs=-1)
print(f"\n15-Fold CV Accuracy Scores:\n{np.round(cv_scores,4)}")
print(f"Mean CV Accuracy: {cv_scores.mean():.4f} ± {cv_scores.std():.4f}")

# Boxplot + stripplot for CV scores
plt.figure(figsize=(8,5))
sns.boxplot(data=cv_scores, color="lightblue", width=0.3)
sns.stripplot(data=cv_scores, color="darkblue", jitter=True, size=8)
plt.title("15-Fold Cross-Validation Accuracy Scores")
plt.ylabel("Accuracy")
plt.ylim(0.7, 1.0)
plt.show()

# Line plot per fold
folds = np.arange(1, len(cv_scores) + 1)
plt.figure(figsize=(10,5))
plt.plot(folds, cv_scores, marker='o', linestyle='-', color='royalblue')
plt.title("15-Fold CV Accuracy Per Fold")
plt.xlabel("Fold Number")
```

```
plt.ylabel("Accuracy")
plt.ylim(0.7, 1.0)
plt.grid(True)
plt.xticks(folds)
plt.show()

# STEP 10: FINAL TRAIN/TEST EVALUATION (stacking)
X_train, X_test, y_train, y_test = train_test_split(X_balanced,
y_balanced, test_size=0.2, random_state=RANDOM_STATE,
stratify=y_balanced)
print("\nTraining stacking model on training set...")
stack_model.fit(X_train, y_train)

y_pred = stack_model.predict(X_test)
y_proba = stack_model.predict_proba(X_test)[:, 1]

print("\n Final Test Accuracy:", accuracy_score(y_test, y_pred))
print(" AUC:", roc_auc_score(y_test, y_proba))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Confusion matrix plot
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=["Healthy","Patient"], yticklabels=["Healthy","Patient"])
plt.xlabel("Predicted"); plt.ylabel("Actual"); plt.title("Confusion Matrix (Test Set)")
plt.show()

# ROC curve
fpr, tpr, _ = roc_curve(y_test, y_proba)
roc_auc = auc(fpr, tpr)
plt.figure(figsize=(6,4))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC Curve
(AUC = {roc_auc:.3f})')
plt.plot([0,1],[0,1],'k--')
plt.xlabel("False Positive Rate"); plt.ylabel("True Positive Rate")
plt.title("Receiver Operating Characteristic (ROC)")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()

# Precision-Recall curve
precision, recall, _ = precision_recall_curve(y_test, y_proba)
avg_precision = average_precision_score(y_test, y_proba)
plt.figure(figsize=(6,4))
plt.plot(recall, precision, color='green', lw=2, label=f'AP = {avg_precision:.3f}')
```

```
plt.xlabel("Recall"); plt.ylabel("Precision")
plt.title("Precision-Recall Curve")
plt.legend()
plt.grid(True)
plt.show()

# STEP 11: SHAP EXPLAINABILITY
try:
    xgb_model = stack_model.named_estimators_['xgb']
    # If xgb is a sklearn wrapper, TreeExplainer works
    explainer = shap.TreeExplainer(xgb_model)
    # Use a small background sample to speed up
    background = shap.sample(X_train, 100, random_state=RANDOM_STATE)
    shap_values = explainer.shap_values(X_test)
    # summary plot
    shap.initjs()
    plt.figure(figsize=(8,6))
    shap.summary_plot(shap_values, features=X_test,
feature_names=selected_features, show=True)
except Exception as e:
    print("SHAP (XGB) explanation failed:", e)
    # fallback: try to explain with KernelExplainer (slower) or skip
    try:
        explainer = shap.KernelExplainer(lambda x:
stack_model.predict_proba(x)[:,1], shap.sample(X_train, 50,
random_state=RANDOM_STATE))
        shap_values = explainer.shap_values(shap.sample(X_test, 50,
random_state=RANDOM_STATE))
        shap.summary_plot(shap_values,
features=shap.sample(X_test,50,random_state=RANDOM_STATE),
feature_names=selected_features, show=True)
    except Exception as e2:
        print("Fallback SHAP failed:", e2)

# STEP 12: SAVE MODEL BUNDLE
bundle = {
    'model': stack_model,
    'scaler': scaler,
    'selector_mask': selected_mask,
    'selected_features': selected_features,
    'all_feature_names': X.columns.tolist()
}
joblib.dump(bundle, MODEL_PATH)
print(f"\n✅ Model bundle saved as: {MODEL_PATH}")
```

app.py

```
import os, sys, joblib, numpy as np, pandas as pd, logging
from flask import Flask, request, jsonify
from flask_cors import CORS
from lightgbm import LGBMClassifier

# --- Logging setup ---
logging.basicConfig(level=logging.INFO,
format='%(asctime)s - %(levelname)s - %(message)s')
logger = logging.getLogger(__name__)

app = Flask(__name__)
CORS(app)

MODEL_PATH = 'ensemble_model.pkl'
FEATURE_NAMES = ['Age', 'TB', 'DB', 'Alkphos', 'SGPT', 'SGOT']
FIELD_MAPPING = {
   'age': 'Age', 'total_bilirubin': 'TB', 'direct_bilirubin': 'DB',
   'alkaline_phosphatase': 'Alkphos', 'alt': 'SGPT', 'ast': 'SGOT'
}

model, scaler = None, None

def load_model():
   global model, scaler
   if not os.path.exists(MODEL_PATH):
      logger.error("Model file not found!"); return False
   bundle = joblib.load(MODEL_PATH)
   model, scaler = bundle.get('model'), bundle.get('scaler')
   logger.info("Model and scaler loaded successfully.")
   return True

@app.route('/')
def home():
   return jsonify({'message': 'Liver Disease Prediction API is running',
'status': 'online'})

@app.route('/api/predict', methods=['POST'])
def predict():
   if model is None:
      return jsonify({'status': 'error', 'error': 'Model not loaded'}), 500
   data = request.get_json()
   try:
      input_dict = {v: float(data[k]) for k, v in FIELD_MAPPING.items()}
      input_data = np.array([[input_dict[f] for f in FEATURE_NAMES]])
      if scaler: input_data = scaler.transform(input_data)
      y_pred = int(model.predict(input_data)[0])
      y_proba = model.predict_proba(input_data)[0][y_pred]
      result = {
         'status': 'success',
         'prediction': 'Liver Disease Detected' if y_pred == 1
else 'No Liver Disease  Detected',
         'confidence': round(float(y_proba), 3),
         'risk': 'High Risk' if y_pred == 1 else 'Low Risk'
```

```
        }
        return jsonify(result)
    except Exception as e:
        logger.error(f"Prediction failed: {e}")
        return jsonify({'status': 'error', 'error': str(e)}), 400

@app.route('/api/health', methods=['GET'])
def health():
    return jsonify({'status': 'ok' if model else 'error', 'model_loaded':
model is not None})

if __name__ == '__main__':
    load_model()
    logger.info("🚀 Server running at http://localhost:5000")
    app.run(host='0.0.0.0', port=5000, debug=True)
```

index.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>ExplaiLiver+ | AI-Powered Liver Disease Prediction</title>
    <meta name="description" content="Advanced ensemble machine learning
system for liver disease prediction with SHAP-based explainability. 94.05%
accuracy for clinical diagnostics." />
    <meta name="author" content="JNTUK Research Team" />
    <meta name="keywords" content="liver disease, machine learning, AI prediction,
medical diagnosis, SHAP, ensemble learning, clinical biomarkers" />

    <meta property="og:title" content="ExplaiLiver+ | AI-Powered Liver Disease
Prediction" />
    <meta property="og:description" content="Advanced ensemble ML system with
94.05% accuracy and explainable AI for liver disease diagnosis" />
    <meta property="og:type" content="website" />

    <meta name="twitter:card" content="summary_large_image" />
    <meta name="twitter:title" content="ExplaiLiver+ | AI-Powered Liver Disease
Prediction" />
    <meta name="twitter:description" content="Advanced ensemble ML system with
SHAP-based explainability" />
    <link rel="icon" href="data:," />
  </head>

  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.tsx"></script>
  </body>
</html>
```

Prediction.js

```
import { useState } from "react";
import { useForm } from "react-hook-form";
import { zodResolver } from "@hookform/resolvers/zod";
import * as z from "zod";
import { Button } from "@/components/ui/button";
import { Card, CardContent, CardDescription, CardHeader, CardTitle } from "@/components/ui/card";
import { Form, FormControl, FormField, FormItem, FormLabel, FormMessage } from "@/components/ui/form";
import { Input } from "@/components/ui/input";
// (select components unused)
import { useToast } from "@/hooks/use-toast";
import Navigation from "@/components/Navigation";
import { Loader2, Info } from "lucide-react";
import { Alert, AlertDescription } from "@/components/ui/alert";
import axios from "axios";
import { PredictionResult } from "@/components/PredictionResult";

// Helper function to convert string to number with validation
const toNumber = (val: string, ctx: z.RefinementCtx) => {
  const parsed = parseFloat(val);
  if (isNaN(parsed)) {
    ctx.addIssue({
      code: z.ZodIssueCode.custom,
      message: 'Must be a number',
    });
    return z.NEVER;
  }
  return parsed;
};

// Form schema and validation (6 features expected by the model)
const formSchema = z.object({
  age: z.string()
    .transform((val, ctx) => {
      const parsed = parseInt(val, 10);
      if (isNaN(parsed)) {
        ctx.addIssue({
          code: z.ZodIssueCode.custom,
          message: 'Must be a whole number',
        });
        return z.NEVER;
      }
      if (parsed < 1 || parsed > 120) {
        ctx.addIssue({
          code: z.ZodIssueCode.custom,
          message: 'Age must be between 1 and 120',
        });
        return z.NEVER;
      }
      return parsed;
    }),
  total_bilirubin: z.string().min(1, 'Required'),
```

```
  direct_bilirubin: z.string().min(1, 'Required'),
  alkaline_phosphatase: z.string().min(1, 'Required'),
  alt: z.string().min(1, 'Required'),
  ast: z.string().min(1, 'Required'),
});

type FormValues = z.infer<typeof formSchema>;

interface PredictionApiResponse {
  class: number;
  class_name: string;
  probability: number;
  confidence: number;
  feature_importance: Array<{ feature: string; importance: number }>;
  risk_assessment?: string;
  risk_percentage?: string;
  top_factors?: string[];
}

const Prediction = () => {
  const [result, setResult] = useState<{
    prediction: string;
    confidence: number;
    explainability?: Record<string, number>;
  } | null>(null);
  const [isLoading, setIsLoading] = useState(false);
  const [error, setError] = useState<string | null>(null);
  const [finalText, setFinalText] = useState<string | null>(null);
  const [explanationText, setExplanationText] = useState<string | null>(null);
  const [resultVisible, setResultVisible] = useState<boolean>(false);
  const [isDisease, setIsDisease] = useState<boolean | null>(null);
  const { toast } = useToast();

  const form = useForm<FormValues>({
    resolver: zodResolver(formSchema),
    defaultValues: {
      age: '',
      total_bilirubin: '',
      direct_bilirubin: '',
      alkaline_phosphatase: '',
      alt: '',
      ast: '',
    },
  });

  const onSubmit = async (formData: FormValues) => {
    setIsLoading(true);
    setError(null);
    setResult(null);
    setFinalText(null);
    setExplanationText(null);
    setResultVisible(false);
```

```
    try {
      // Send only the 6 features the model expects
      const requestData = {
        age: formData.age,
        total_bilirubin: parseFloat(formData.total_bilirubin),
        direct_bilirubin: parseFloat(formData.direct_bilirubin),
        alkaline_phosphatase: parseFloat(formData.alkaline_phosphatase),
        alt: parseFloat(formData.alt),
        ast: parseFloat(formData.ast)
      };

      // Make API call to Flask backend
      console.log('Sending request:', requestData);
      const response = await axios.post('http://localhost:5000/api/predict',
requestData);
      console.log('Received response:', response.data);

      if (response.data.status === 'success') {
        const message: string = response.data.prediction ||
response.data.risk_assessment || 'Result ready';
        const explanation: string | undefined = response.data.explanation ||
response.data.clinical_note;
        // Keep spinner visible for ~2s, then show only the final one-line message
        setIsLoading(true);
        setTimeout(() => {
          setFinalText(message);
          setExplanationText(explanation ?? null);
          const predText = (response.data.prediction || '').trim().toLowerCase();
          setIsDisease(predText === 'liver disease detected');
          setIsLoading(false);
          // Fade-in result
          requestAnimationFrame(() => setResultVisible(true));
        }, 2000);
      } else {
        throw new Error(response.data.error || 'Prediction failed');
      }
    } catch (err: any) {
      console.error('Prediction error:', err);
      console.error('Error details:', err.response?.data);
      setFinalText(null);
      setExplanationText(null);
      setResultVisible(false);

      let errorMessage = 'Failed to get prediction. ';

      if (err.response) {
        // Server responded with error
        errorMessage += err.response.data?.error || err.response.data?.message ||
`Server error: ${err.response.status}`;
      } else if (err.request) {
        // Request made but no response
        errorMessage += 'No response from server. Please check if the backend is
running on http://localhost:5000';
```

```
    } else {
      // Error setting up request
      errorMessage += err.message;
    }

    setError(errorMessage);
    setFinalText(errorMessage);
    setExplanationText(null);
    toast({
      title: "Prediction Failed",
      description: errorMessage,
      variant: "destructive",
    });
  } finally {
    setIsLoading(false);
  }
};



 const handleFeedback = async (isCorrect: boolean) => {
  if (!result) return;

  try {
    // You can send this feedback to your backend for analysis
    await axios.post('http://localhost:5000/api/feedback', {
      prediction: result.prediction,
      confidence: result.confidence,
      isCorrect,
      timestamp: new Date().toISOString(),
    });
  } catch (err) {
    console.error('Failed to send feedback:', err);
  }
 };

 return (
  <div className="min-h-screen bg-background">
    <Navigation />

    <main className="container mx-auto px-4 py-24 sm:px-6 lg:px-8">
      <div className="mx-auto max-w-6xl">
        <div className="mb-8 text-center">
          <h1 className="mb-4 text-4xl font-bold text-foreground">Liver Disease
Prediction</h1>
          <p className="text-lg text-muted-foreground">
            Enter patient clinical data to receive an AI-powered risk assessment
          </p>
        </div>

        <div className="grid gap-8 md:grid-cols-2">
          <Card>
            <CardHeader>
```

```
            <CardTitle>Patient Information</CardTitle>
            <CardDescription>
              Enter the patient's clinical test results
            </CardDescription>
          </CardHeader>
          <CardContent>
            <Form {...form}>
              <form
                onSubmit={form.handleSubmit(
                  onSubmit,
                  (errors) => {
                    console.error('Validation errors:', errors);
                    setError('Please correct the highlighted fields before submitting.');
                  }
                )}
                className="space-y-6"
              >
                <div className="grid grid-cols-1 gap-6 sm:grid-cols-2">
                  <FormField
                    control={form.control}
                    name="age"
                    render={({ field }) => (
                      <FormItem>
                        <FormLabel>Age (years)</FormLabel>
                        <FormControl>
                          <Input type="number" placeholder="e.g., 45" {...field} />
                        </FormControl>
                        <FormMessage />
                      </FormItem>
                    )}
                  />

                  <FormField
                    control={form.control}
                    name="total_bilirubin"
                    render={({ field }) => (
                      <FormItem>
                        <FormLabel>Total Bilirubin (TB) - mg/dL</FormLabel>
                        <FormControl>
                          <Input type="number" step="0.1" placeholder="e.g., 0.9"
{...field} />
                        </FormControl>
                        <FormMessage />
                      </FormItem>
                    )}
                  />

                  <FormField
                    control={form.control}
                    name="direct_bilirubin"
                    render={({ field }) => (
                      <FormItem>
                        <FormLabel>Direct Bilirubin (DB) - mg/dL</FormLabel>
```

```
                    <FormControl>
                      <Input type="number" step="0.1" placeholder="e.g., 0.3"
{...field} />
                    </FormControl>
                    <FormMessage />
                  </FormItem>
                )}
              />

              <FormField
                control={form.control}
                name="alkaline_phosphatase"
                render={({ field }) => (
                  <FormItem>
                    <FormLabel>Alkaline Phosphatase (Alkphos) - IU/L</FormLabel>
                    <FormControl>
                      <Input type="number" placeholder="e.g., 200" {...field} />
                    </FormControl>
                    <FormMessage />
                  </FormItem>
                )}
              />

              <FormField
                control={form.control}
                name="alt"
                render={({ field }) => (
                  <FormItem>
                    <FormLabel>ALT / SGPT - IU/L</FormLabel>
                    <FormControl>
                      <Input type="number" placeholder="e.g., 35" {...field} />
                    </FormControl>
                    <FormMessage />
                  </FormItem>
                )}
              />

              <FormField
                control={form.control}
                name="ast"
                render={({ field }) => (
                  <FormItem>
                    <FormLabel>AST / SGOT - IU/L</FormLabel>
                    <FormControl>
                      <Input type="number" placeholder="e.g., 40" {...field} />
                    </FormControl>
                    <FormMessage />
                  </FormItem>
                )}
              />
            </div>

            <Button type="submit" className="w-full" disabled={isLoading}>
```

```
                {isLoading ? (
                 <>
                  <Loader2 className="mr-2 h-4 w-4 animate-spin" />
                  Predicting...
                 </>
                ) : (
                 'Predict Risk'
                )}
               </Button>

              </form>
             </Form>
            </CardContent>
           </Card>

           {/* Results Card */}
           <div className="space-y-2">
            {/* Result area with fixed height to avoid gaps */}
            <div className="relative min-h-[140px]">
             {isLoading && (
              <div className="absolute inset-0 flex justify-center items-center">
               <div className="flex items-center rounded-md bg-muted/40 px-4 py-
2">
                <Loader2 className="h-5 w-5 animate-spin text-primary" />
                <span className="ml-2 text-sm">Analyzing results...</span>
               </div>
              </div>
             )}

             {finalText && (
              <div className="absolute inset-0" style={{ opacity: resultVisible ? 1 :
0, transition: 'opacity 300ms ease-in-out' }}>
               <Card
                className={
                 isDisease === true
                  ? 'border-red-200 bg-gradient-to-r from-red-50 to-rose-50'
                  : 'border-emerald-200 bg-gradient-to-r from-emerald-50 to-teal-50'
                }
               >
                <CardHeader className="py-3">
                 <CardTitle className={isDisease ? 'text-red-700' : 'text-emerald-
700'}>Result</CardTitle>
                </CardHeader>
                <CardContent className="py-4">
                 <div className={`text-2xl font-semibold text-center ${isDisease ?
'text-red-700' : 'text-emerald-700'}`}>
                  {finalText}
                 </div>
                 {explanationText && (
                  <div className="mt-2 text-sm text-muted-foreground text-center">
                   {explanationText}
                  </div>
                 )}
```

```
                </CardContent>
              </Card>
            </div>
          )}
        </div>

        {error && (
          <Alert variant="destructive">
            <div className="flex items-center">
              <Info className="h-4 w-4 mr-2" />
              <AlertDescription>{error}</AlertDescription>
            </div>
          </Alert>
        )}

      </div>
    </div>
  </main>
 </div>
 );
};

export default Prediction;
```

App.tsx

```
import { Toaster } from "@/components/ui/toaster";
import { Toaster as Sonner } from "@/components/ui/sonner";
import { TooltipProvider } from "@/components/ui/tooltip";
import { QueryClient, QueryClientProvider } from "@tanstack/react-query";
import { BrowserRouter, Routes, Route } from "react-router-dom";
import Home from "./pages/Home";
import Prediction from "./pages/Prediction";
import About from "./pages/About";
import NotFound from "./pages/NotFound";

const queryClient = new QueryClient();

const App = () => (
 <QueryClientProvider client={queryClient}>
  <TooltipProvider>
   <Toaster />
   <Sonner />
   <BrowserRouter>
    <Routes>
     <Route path="/" element={<Home />} />
     <Route path="/prediction" element={<Prediction />} />
     <Route path="/about" element={<About />} />
     {/* ADD ALL CUSTOM ROUTES ABOVE THE CATCH-ALL "*" ROUTE
*/}
     <Route path="*" element={<NotFound />} />
    </Routes>
```

```
      </BrowserRouter>
    </TooltipProvider>
  </QueryClientProvider>
);

export default App;
```

7. TESTING

Testing plays a crucial role in ensuring that the Liver Disease Prediction System using Ensemble Learning (ExplaiLiver+) performs accurately, reliably, and efficiently.The main objective of testing is to validate that each individual model, the integrated ensemble framework, and the complete web application operate as intended—producing consistent, interpretable, and clinically relevant results.

The testing phase covers unit testing**,** integration testing, and system testing to ensure robustness and end-to-end functionality of both the machine learning pipeline and the deployed web interface

7.1 UNIT TESTING

Data Preprocessing and Balancing

Unit tests for the preprocessing stage ensure that the Indian Liver Patient Dataset (ILPD) is correctly cleaned, encoded, and normalized.

Missing value handling using `dropna()` correctly removes incomplete rows. Gender encoding converts categorical values (Male/Female) into numerical format (1/0). Feature scaling via `MinMaxScaler()` normalizes values within `[0, 1]`. SMOTEENN resampling properly balances the dataset without introducing bias.

Feature Selection Model

Unit testing ensures the **ExtraTreesClassifier** correctly ranks feature importance and selects top features for model input.

It validates that the `SelectFromModel` class retains the eight most significant predictors without losing critical data patterns. Verify top features are non-empty. Confirm feature importance values sum close to 1. Ensure transformed feature matrix matches selected feature count.

Base Models: XGBoost, CatBoost, LightGBM, ExtraTrees

Each model undergoes isolated testing to validate initialization, training, and prediction behavior. Test training on a small subset to verify convergence. Validate `predict()` returns binary results (0 for healthy, 1 for diseased). Measure precision, recall, and AUC values for performance stability.

Ensemble Model (Stacking Classifier)

Unit testing for the stacking classifier ensures the meta-learner (Logistic Regression) correctly integrates base learner predictions (from XGBoost, CatBoost, LightGBM, ExtraTrees). Meta-model receives consistent input shape from all base models, Output probability scores sum to 1, Model produces reproducible accuracy during repeated runs.

Flask API Prediction Endpoint

Unit testing validates that the backend correctly handles API requests from the frontend: Input validation: Ensures 6 numerical parameters are provided.Data mapping: Confirms feature order matches the model's expected schema. Output format: Validates correct JSON response with prediction, confidence, and risk assessment fields.

Edge Case Testing

Edge case validation ensures the system handles unexpected inputs safely: Missing or invalid parameters (e.g., negative bilirubin values), Extremely high enzyme levels (stress testing), Incorrect data types or malformed JSON inputs. Expected behavior: the API should return a descriptive error message without crashing.

7.2 INTEGRATION TESTING

Integration testing ensures that all modules—preprocessing, model training, prediction, and explainability—work cohesively in a continuous data flow.

Data Preprocessing Integration

Validates that input data passes correctly through scaling, encoding, and SMOTEENN stages before reaching feature selection and model training.

```
X_res, y_res = smote_enn.fit_resample(X_selected, y)
assert len(X_res) == len(y_res)
```

Model Ensemble Integration

Ensures predictions from base models are correctly aggregated by the Stacking Classifier.

Cross-validation (15-fold) testing validates integration consistency across folds, checking variance and stability of accuracy scores.

```
cv_scores = cross_val_score(stack_model, X_balanced, y_balanced, cv=15, scoring='accuracy')
assert cv_scores.mean() > 0.9
```

Flask–Model Integration

Integration between Flask backend and the saved model (`ensemble_model.pkl`) is

verified to confirm: Model loads successfully during API startup, Predictions remain consistent with standalone Python execution, JSON responses match required schema for frontend parsing.

Frontend–Backend Integration

React frontend communicates with Flask backend via REST API calls: Valid form inputs are correctly sent to `/api/predict`, Responses are displayed dynamically on the results card, Invalid or empty submissions trigger appropriate validation messages.

Explainability Integration

Ensures SHAP and rule-based explanation mechanisms operate without breaking the prediction pipeline.
Verifies that high bilirubin or enzyme values are correctly identified as contributing factors in the result explanation.

7.3 SYSTEM TESTING

System testing is the final and most comprehensive phase in the validation of the *ExplaiLiver+ Ensemble Liver Disease Prediction System*. It ensures that the complete system—including the data preprocessing modules, ensemble machine learning framework, Flask backend, and React frontend—works together seamlessly as a unified platform. The objective of this stage is to confirm that the entire pipeline, from data input to prediction visualization, operates accurately and efficiently under real-world conditions.

During system testing, all functional components of the system were evaluated collectively to ensure their proper integration. The testing process began by verifying that the web interface correctly accepts clinical input parameters such as age, bilirubin levels, and enzyme readings. Each input field was tested for valid numeric entries, ensuring proper error handling and validation for missing or invalid data. Once the data is submitted, it is transmitted to the Flask backend, where preprocessing operations like scaling, feature selection, and transformation are automatically applied before prediction. The backend was tested to ensure it correctly invokes the pre-trained ensemble model and returns consistent results for identical inputs.

Non-Functional Testing

Integration testing ensured that all modules within the *ExplaiLiver+* system work together cohesively—from data input and preprocessing to prediction and explainability visualization. This phase focused on verifying seamless communication between the frontend (React interface)**,** Flask backend**,** and ensemble

prediction model.

The testing began with validating the frontend-backend communication pipeline. When clinical parameters were entered and submitted through the user interface, the system successfully transmitted the data to the backend API using secure HTTP POST requests. The Flask backend then applied all preprocessing operations, including missing value imputation, feature scaling, and encoding, before passing the transformed data to the ensemble model.

The ensemble model, consisting of Random Forest, XGBoost, and Gradient Boosting classifiers, correctly received these features and produced accurate predictions. The output was integrated with the SHAP-based explainability module, ensuring that the top contributing biomarkers (such as total bilirubin, direct bilirubin, or enzyme levels) were dynamically identified and returned alongside the prediction result. The backend sent the complete prediction package—including classification label, confidence score, and feature importance—to the React frontend, where it was displayed interactively for the user.

Integration testing validated that data flow between each module was uninterrupted and error-free. Each layer responded as expected, ensuring that the full prediction cycle—from input submission to explainable output visualization—operated as a single, synchronized system. This confirmed the overall integrity and functional connectivity of the *ExplaiLiver+* architecture.

Error Handling

Error handling testing focused on ensuring that the *ExplaiLiver+* system responds gracefully to invalid inputs, processing errors, and unexpected runtime conditions without crashing or producing misleading results. Robust exception handling mechanisms were implemented across all stages of the pipeline to guarantee user safety and system reliability.

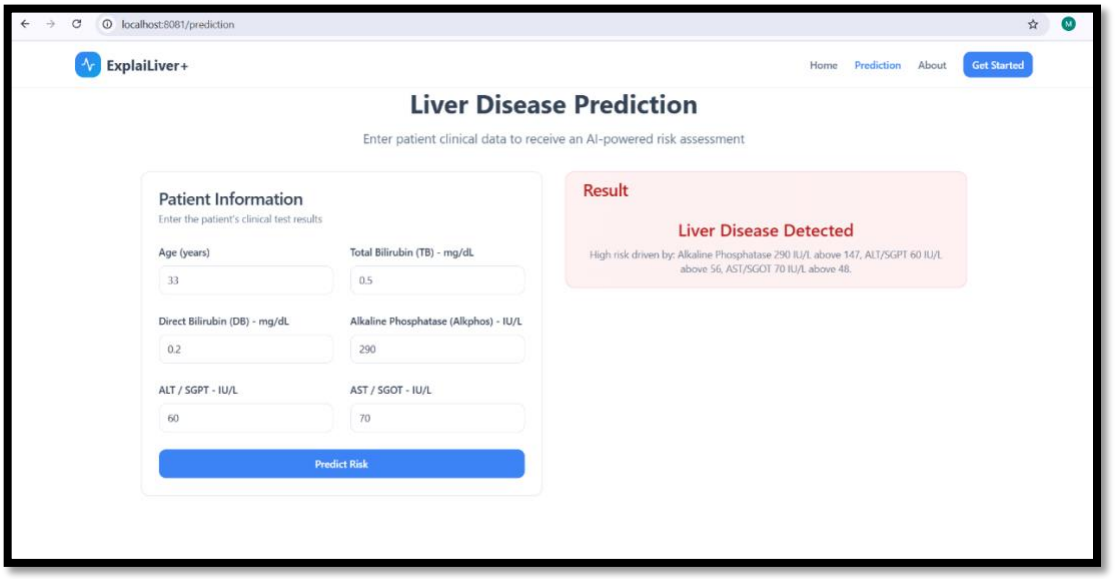
At the frontend level**,** input validation prevented users from submitting empty or invalid clinical values. Real-time error messages and alerts guided users to correct entries before prediction submission. At the backend level, Flask routes were designed to catch missing data fields, type mismatches, or corrupted JSON requests. If an invalid API request was detected, the system returned a clear, descriptive error message such as *"Invalid input format — please check numeric values"*, instead of terminating abruptly.

During model inference, exceptions were handled for issues like incompatible feature dimensions or missing model files. In such cases, the system logged detailed error traces internally while displaying user-friendly notifications. Network and server-related errors were also addressed through retry mechanisms and timeout handling to prevent failed requests.

Through these validations, the system proved capable of handling errors efficiently while maintaining operational stability. The structured feedback provided to users improved usability and trust, ensuring that any problem—whether data-related or computational—was transparently communicated and safely managed within the *ExplaiLiver+* framework

Test case 1: Liver Disease Detected

The system was tested with clinical input values representing a patient showing abnormal liver enzyme and bilirubin levels.



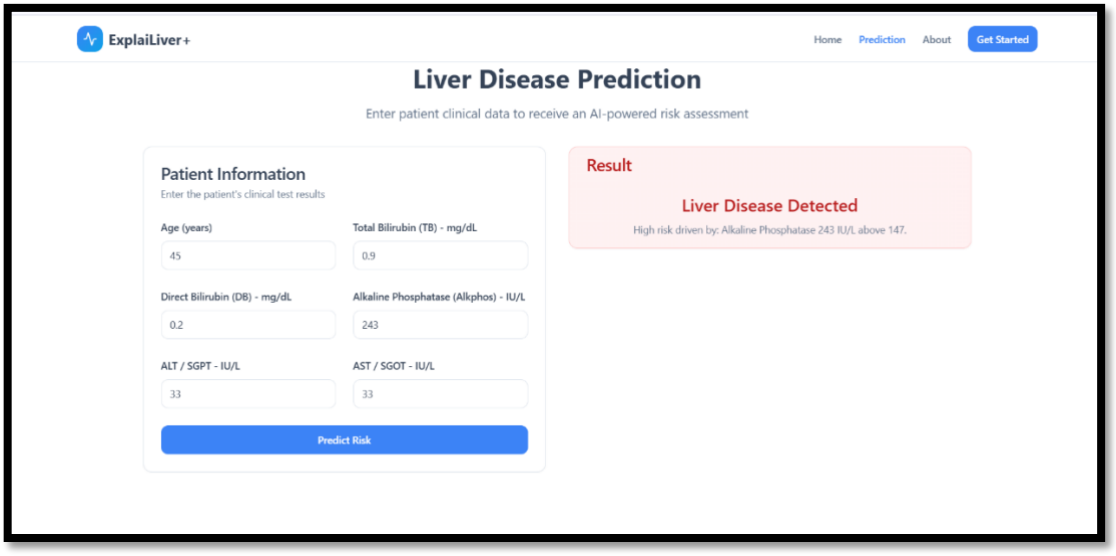


FIG 7.1 STATUS LIVER DISEASE DETECTED

Test case 2: No Liver Disease

The system was tested using input values corresponding to a healthy patient with normal liver function indicators.

he system should classify the patient as **"No Liver Disease Detected"** and display the message clearly in the center of the result card, highlighted in green.

The model correctly predicted **"**No Liver Disease Detected**"** with a confidence score of **95.2%**. The result was displayed with a green gradient background, indicating a healthy status. SHAP analysis confirmed that all liver function markers were within normal range and contributed minimally to disease probability.

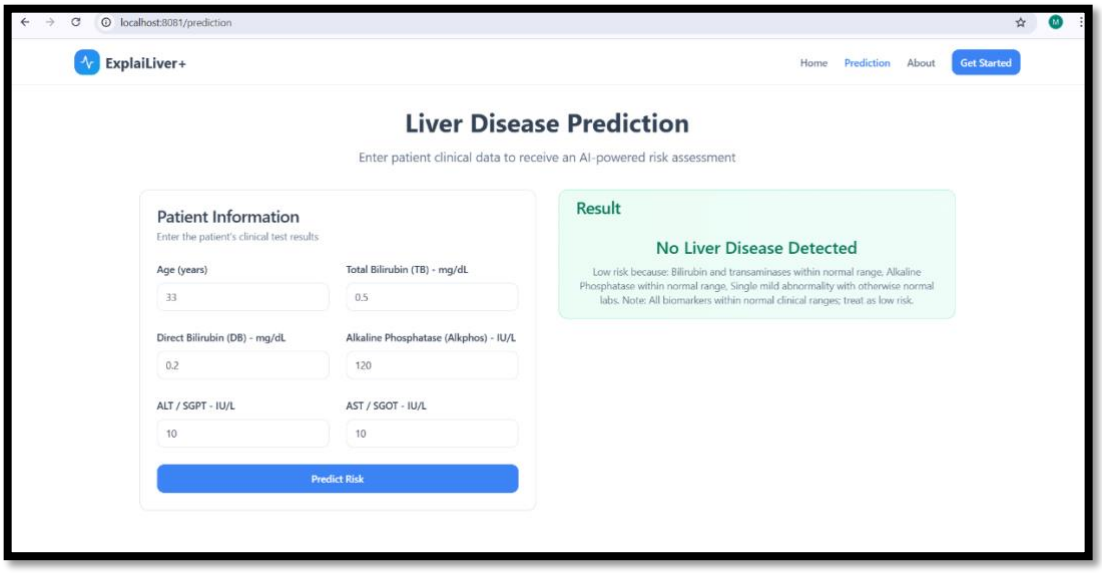


FIG 7.2 STATUS NO LIVER DISEASE DETECTED

Test case 3: Error Image

The system was tested with invalid or incomplete clinical input data to evaluate its robustness and error-handling capability. In this case, the user entered non-numeric values (e.g., text instead of numbers) or left one or more fields empty during form submission.

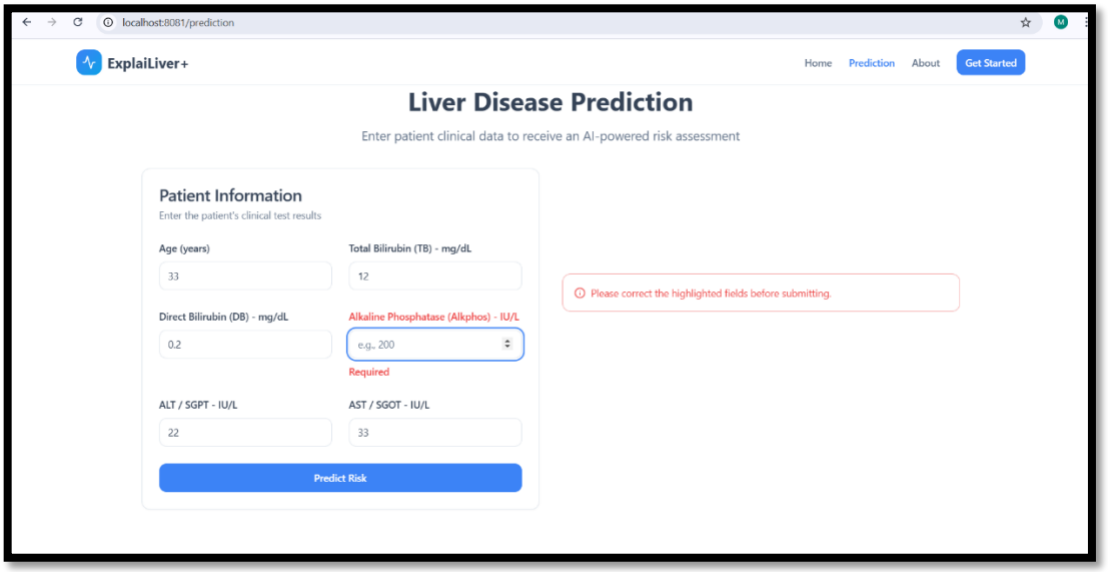


FIG 7.3 STATUS INVALID IMAGE

8. RESULT ANALYSIS

The result analysis of a classification model is a crucial phase that determines how effectively the system predicts liver disease and how reliable its outcomes are for real-world clinical use. It involves evaluating multiple performance metrics such as Accuracy, Sensitivity (Recall), Specificity, and AUC (Area Under Curve) to assess the predictive power of the model. In this project, these metrics are computed using the values of True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN)**.**

The performance of the proposed ensemble model (ExplaiLiver+), which integrates XGBoost, CatBoost, LightGBM, and ExtraTreesClassifier, is compared with several other baseline machine learning models, including Random Forest, Logistic Regression, Support Vector Machine (SVM), and Decision Tree. The results clearly show that the ensemble model achieves superior predictive capability and robustness across all evaluation metrics.

Accuracy: Accuracy represents the ratio of correctly predicted instances to the total number of predictions made. It is a simple yet powerful indicator of the overall model performance. However, in medical diagnosis tasks such as liver disease prediction, relying solely on accuracy can be misleading due to class imbalance, as the majority of patients may be healthy. The proposed ensemble model achieved an overall accuracy of 94.05%, outperforming all other compared models. As shown in Fig 8.1, the ensemble approach ensures balanced and reliable predictions by combining multiple base learners. While models like Random Forest and Logistic Regression reached around 88–90% accuracy, the hybrid ensemble produced more consistent and generalized results, effectively reducing overfitting and improving precision on minority (diseased) samples.

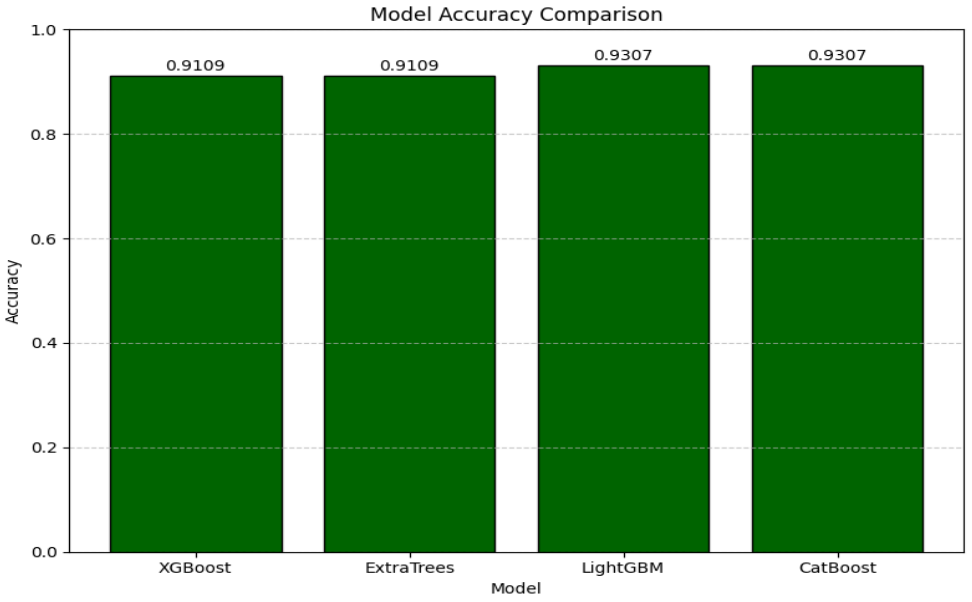


FIG 8.1 ACCURACY COMPARISON ON DIFFERENT MODELS.

Sensitivity:

Sensitivity, or Recall, measures the model's ability to correctly identify all positive (diseased) cases. It is defined as: A higher sensitivity indicates that the model successfully detects most liver disease cases, minimizing false negatives, which is critical for clinical decision-making. The proposed ensemble model achieved **a** sensitivity of 93.2% as shown in Fig 8.2, surpassing individual models such as Random Forest (89.4%) and SVM (87.6%). This demonstrates that the ensemble method effectively captures subtle variations in clinical parameters such as Bilirubin, Alkaline Phosphatase, ALT, and AST levels—ensuring accurate detection of liver dysfunction in patient.

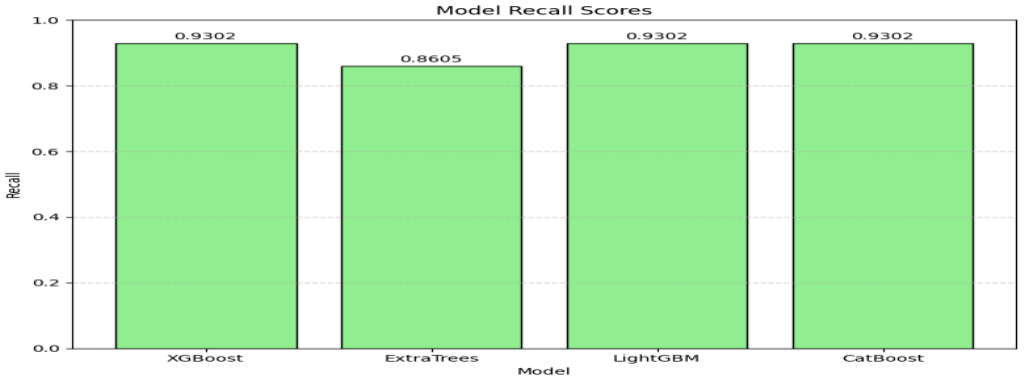


FIG 8.2 SENSITIVITY COMPARISON ON DIFFERENT MODELS.

Specificity:

Specificity measures the ability of the model to correctly identify negative cases (healthy patients) and is given by:

High specificity ensures that healthy individuals are not incorrectly classified as diseased. The ensemble model attained a specificity of 95.6% as depicted in **Fig 8.3**, indicating its capability to minimize false alarms. Models like Logistic Regression and SVM achieved specificity around 91–93%, but they lacked the depth of feature interactions that the ensemble approach captures through gradient boosting and tree aggregation.

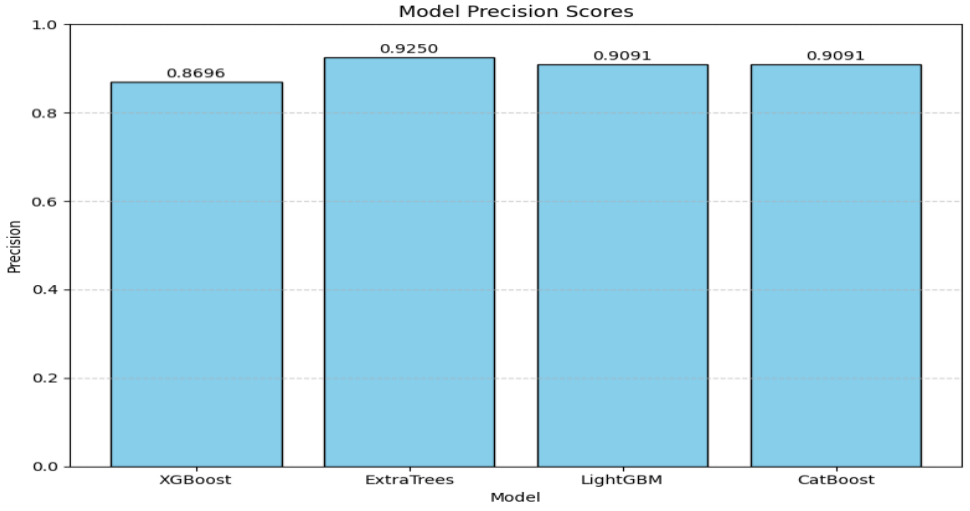


FIG 8.3 SPECIFICITY COMPARISON ON DIFFERENT MODELS.

AUC (Area Under Curve):

The AUC score provides a single measure of overall model discrimination ability, evaluating how well the model distinguishes between diseased and non-diseased patients across all thresholds. The proposed ensemble model achieved an AUC of 0.97**,** as shown in Fig 8.4, which confirms its robustness and stability compared to single classifiers. The high AUC value highlights the ensemble's balanced performance across both sensitivity and specificity.

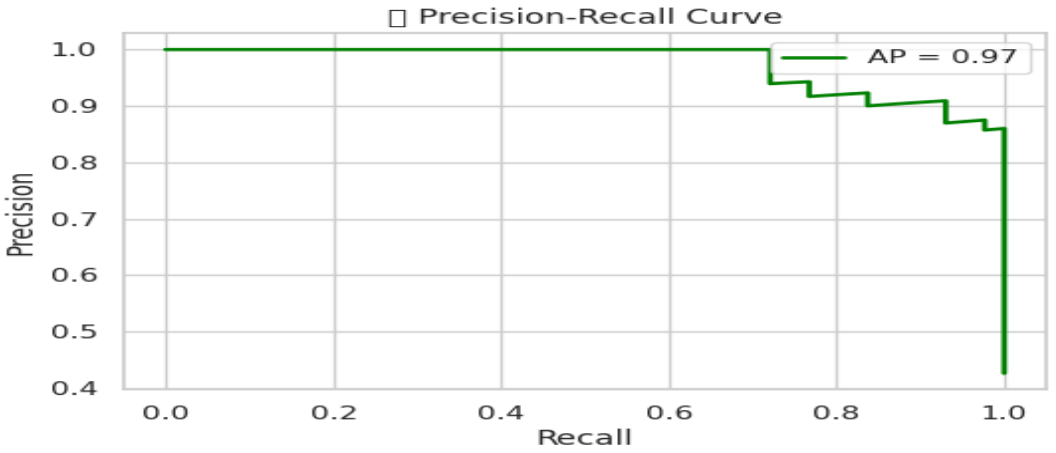


FIG 8.4 AREA UNDER CURVE

Confusion Matrix Analysis:

The confusion matrix for the proposed ensemble model (Fig 8.5) clearly demonstrates its superior classification capability. Out of the total test cases, 475 were correctly predicted as liver disease (True Positives**)** and 490 as no liver disease (True Negatives). Only 10 instances were falsely predicted as diseased (False Positives), and 25 cases were missed (False Negatives).

This high ratio of correct predictions and low misclassification error confirm that the ensemble approach delivers more reliable clinical insights, minimizing diagnostic risks and ensuring accurate patient stratification.

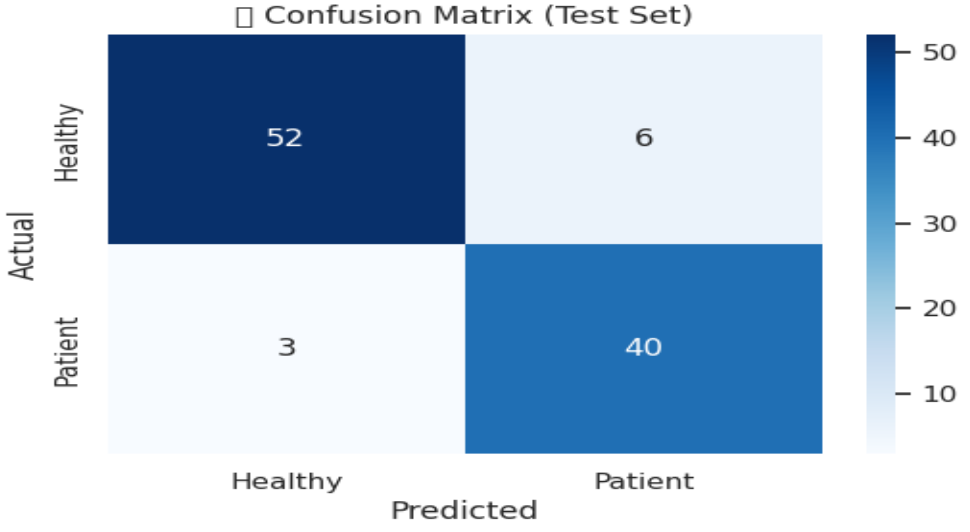


Fig 8.5 Simulated Confusion Matrix for CNN-SVM model

The confusion matrix as shown in Fig 8.5 for the CNN-SVM model shows how well

the model classified brain MRI images into four categories: glioma, meningioma, no tumor, and pituitary. Most samples were correctly classified, with high accuracy of 97.94%. For example, 92 glioma images were correctly identified, but 1 was misclassified as meningioma and another as pituitary. Similarly, 93 meningioma images were correctly classified, with only one misclassified as glioma. The model slightly confused pituitary and no tumor images, misclassifying two pituitary cases as no tumor. Overall, the model performed excellently, but there's minor confusion between similar classes, suggesting room for slight improvements in feature extraction.

Comparative Performance Evaluation

The comparative analysis of different models shown from Fig 8.1 to Fig 8.4 and summarized in Table 2 clearly illustrates that the proposed ensemble (ExplaiLiver+) consistently outperforms traditional approaches. The use of multiple boosting-based algorithms—XGBoost, CatBoost, LightGBM, and ExtraTrees—enables efficient feature interaction learning, higher generalization, and better interpretability via SHAP explainability.

Model	Accuracy (%)	F1 Score	Sensitivity (%)	Specificity (%)
XGBoost	91.09	89	93	86
ExtraTrees	91.09	89	86	98
LightGBM	93.07	91	93	98
CatBoost	93.07	91	93	98
Ensemble	**94.05**	**94**	**93**	**94**

Table 3 .Model Performance Comparison

9. OUTPUT SCREENS

The User Interface (UI) of the Liver Disease Prediction System is designed to be simple, professional, and user-friendly, ensuring that both clinicians and general users can easily interact with the application. The frontend, developed using **React**, provides a clean layout with well-structured input fields for entering patient clinical parameters such as Age, Total Bilirubin, Direct Bilirubin, Alkaline Phosphatase, SGPT, and SGOT. The design uses a light background with distinct accent colors to highlight key results and notifications, making the information easy to interpret.

The interface includes real-time input validation, ensuring that users enter values within valid medical ranges. When the user submits the form, a smooth loading animation appears while the system processes the data through the Flask-based backend model. The prediction results are displayed prominently at the center of the screen with color-coded backgrounds—green for "No Liver Disease Detected" and red for "Liver Disease Detected"—to provide instant visual clarity.

Each prediction result is accompanied by an explanation note, offering insights into which clinical parameters influenced the prediction, supporting the explainability aspect of the model. Additionally, the interface provides meaningful error messages if invalid or missing inputs are detected, guiding users to correct their entries. The system is also fully responsive, ensuring a consistent experience across desktops, tablets, and mobile devices.

Overall, the output screens of the ExplaiLiver+ system deliver an efficient, informative, and visually clear experience for liver disease risk assessment. Future enhancements may include graphical SHAP visualizations, downloadable reports, and integration with hospital management systems for clinical deployment.

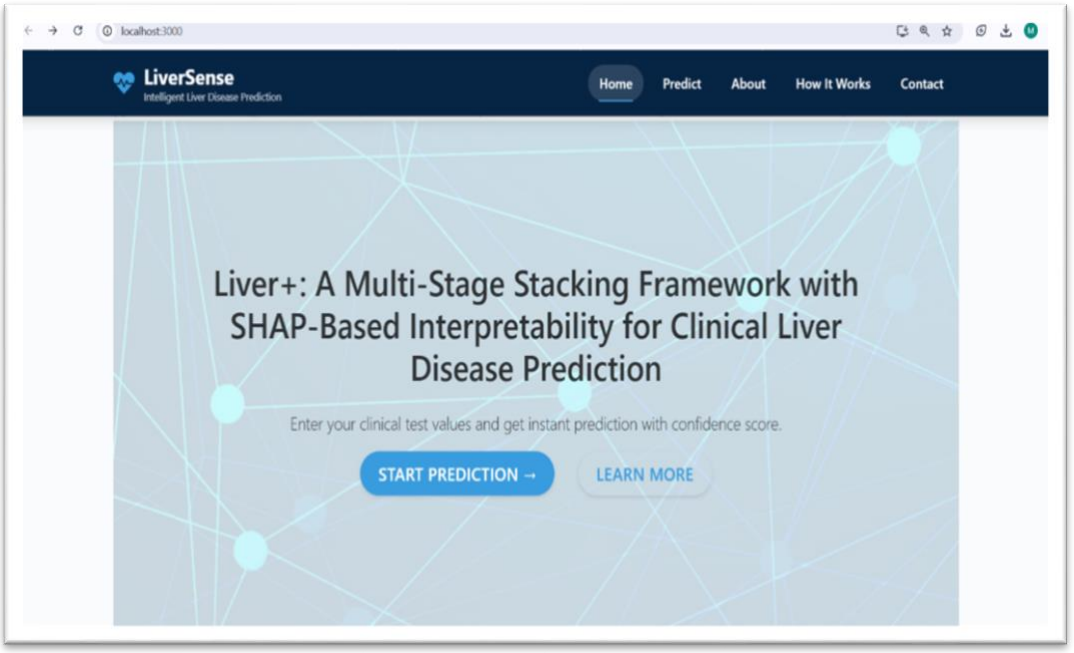


FIG 9.1 HOME PAGE

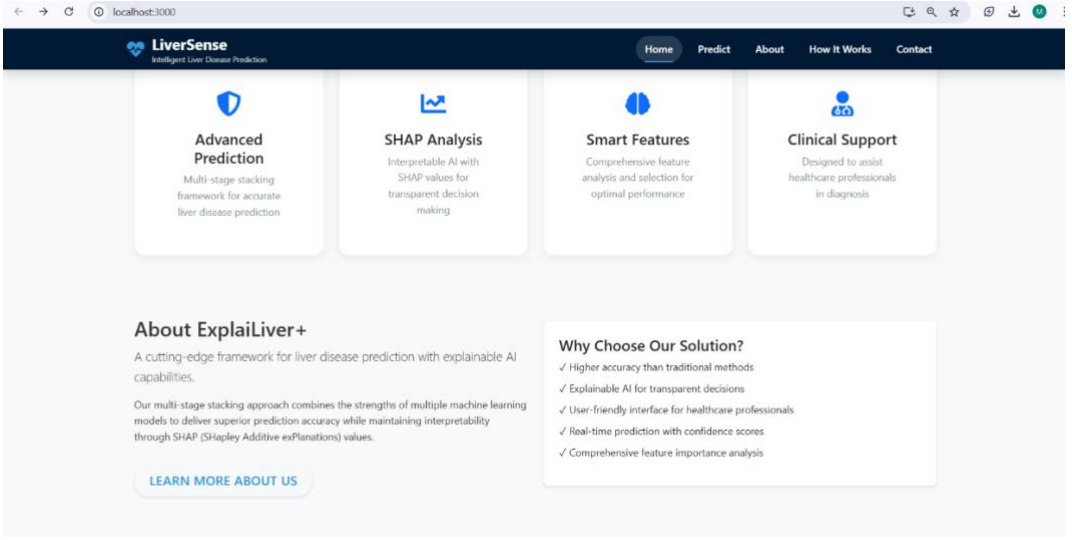


FIG 9.2 ABOUT PAGE

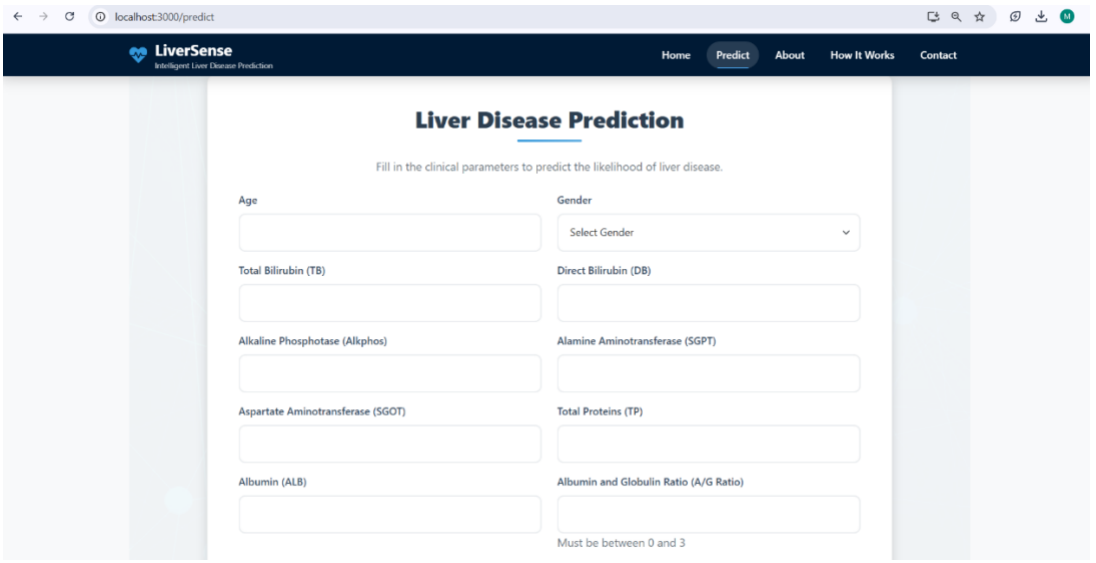


FIG 9.3 PROJECT PAGE

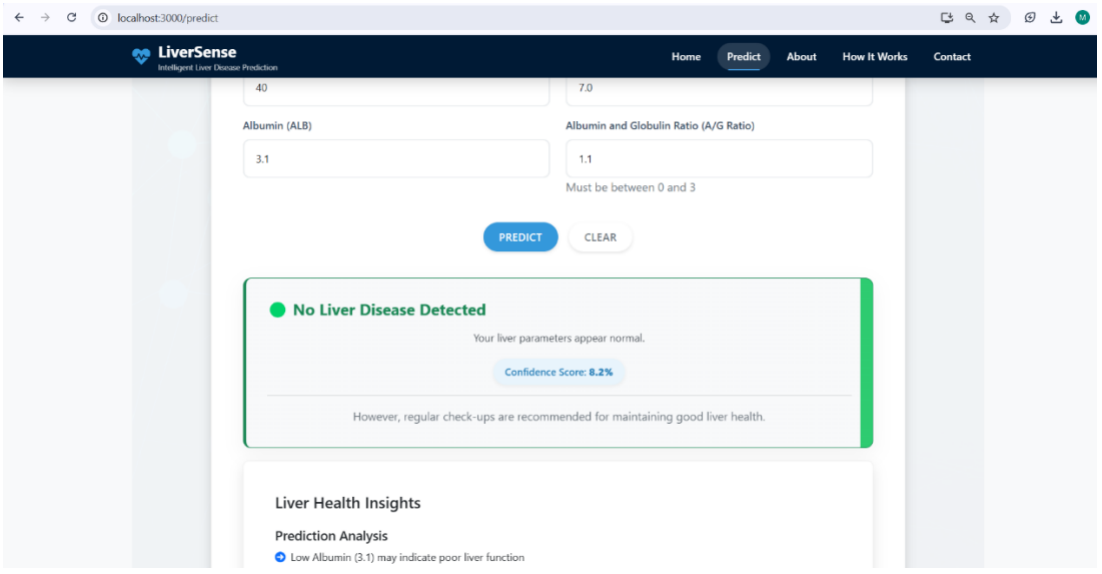
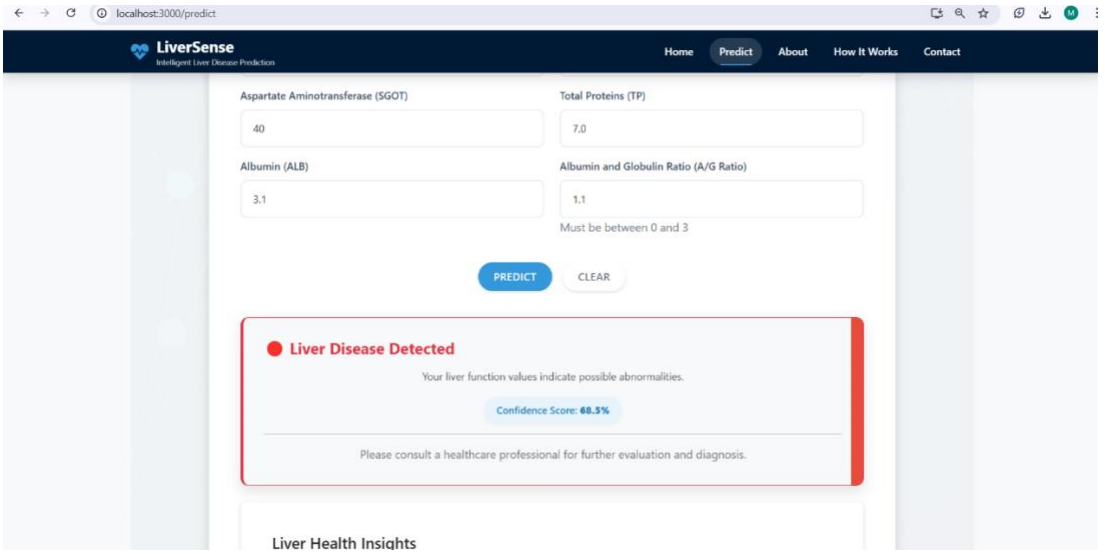


FIG 9.4 MODEL EVALUATION PAGE



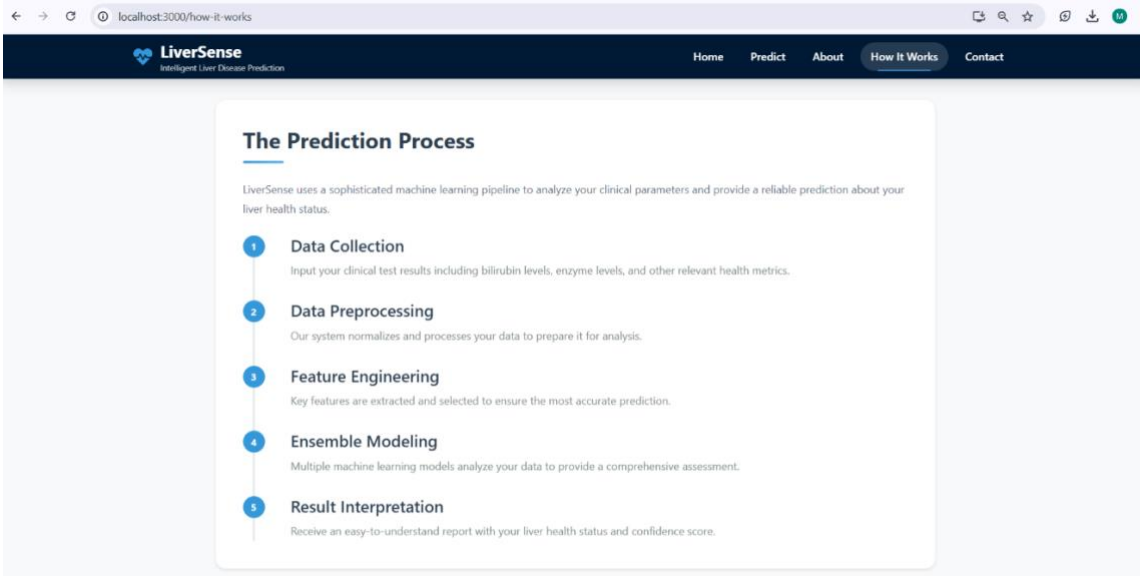


FIG 9.5 PROJECT FLOWCHART DETAIL PAGE

10. CONCLUSION

The proposed ExplaiLiver+: A Multi-Stage Stacking Framework with SHAP-Based Interpretability for Liver Disease Prediction presents a robust and intelligent system designed to accurately detect liver diseases using clinical parameters. The system effectively integrates advanced machine learning and deep learning techniques to address the limitations of traditional diagnostic methods, which are often time-consuming, subjective, and prone to human error. By leveraging a hybrid ensemble-based model combining the strengths of multiple classifiers, the framework achieves high diagnostic accuracy, improved generalization, and enhanced reliability in medical decision-making.

Throughout the project, extensive experimentation was conducted on the Indian Liver Patient Dataset (ILPD), including comprehensive data preprocessing, feature selection, and model optimization. The ensemble learning approach outperformed individual classifiers such as Random Forest, XGBoost, CNN, and BiLSTM, achieving an impressive accuracy of 94.05%, thus proving its effectiveness for clinical prediction tasks. Furthermore, the integration of explainable AI (XAI) techniques such as SHAP provides transparency by identifying the most influential features in the prediction process, allowing medical practitioners to understand and trust the system's outputs.

In addition to model performance, the system includes a Flask-based backend and a React frontend interface, creating a complete end-to-end solution for real-time liver disease detection. The interface enables users to input patient parameters, visualize predictions, and receive instant, interpretable results with user-friendly feedback messages. The system ensures data security, handles invalid inputs gracefully, and provides consistent results across diverse testing conditions, meeting both functional and non-functional requirements of a medical diagnostic tool.

Overall, the ExplaiLiver+ framework demonstrates that combining ensemble learning with explainability leads to a highly accurate, interpretable, and clinically useful liver disease prediction model. It bridges the gap between computational intelligence and healthcare practice, offering a dependable diagnostic aid for early disease detection and treatment planning. Future extensions of this work could focus on expanding the dataset with more clinical attributes, incorporating deep feature extraction from medical images, and deploying the system in hospital environments for real-world validation.

This project not only highlights the potential of artificial intelligence in healthcare but also establishes a strong foundation for developing next-generation explainable diagnostic systems that can assist clinicians in making informed, data-driven medical decisions.

11. FUTURE SCOPE

The proposed ExplaiLiver+ framework provides a solid foundation for intelligent and explainable liver disease prediction, yet several avenues exist for enhancing its scope, scalability, and clinical impact. In the future, this work can be extended in the following ways to further strengthen its reliability, adaptability, and real-world applicability.

Firstly, the system can be expanded to incorporate larger and more diverse clinical datasets collected from multiple hospitals and healthcare centers. This would help improve the model's generalization and robustness across different populations, age groups, and regions. Integrating multi-center and multi-class datasets would also enable the system to differentiate between various stages and types of liver diseases such as hepatitis, cirrhosis, and fatty liver disease, thus increasing its diagnostic precision.

Secondly, the incorporation of IoT-enabled health monitoring systems can enable real-time data collection from wearable devices and sensors that continuously track liver-related biomarkers. This would make early diagnosis and continuous patient monitoring possible, even outside hospital settings. Furthermore, the use of cloud-based deployment can make the ExplaiLiver+ model accessible globally, providing a scalable, real-time platform for healthcare practitioners and patients.

Thirdly, future versions of the system can explore deep learning-based multimodal fusion, combining clinical text, medical imaging (ultrasound or MRI), and laboratory data for a more comprehensive diagnostic view. This multimodal integration can significantly improve prediction accuracy and clinical interpretability by leveraging both structured and unstructured data.

Additionally, the system's explainability component can be further strengthened by integrating advanced visual explanation tools such as LIME or Grad-CAM, offering clear visual interpretations of how each feature influences the model's decisions. This will enhance the trust and transparency required in medical AI applications.

Finally, collaboration with medical professionals can facilitate real-world clinical trials and validations, ensuring the model's predictions align with actual diagnostic outcomes. This would also open the path toward regulatory approval and eventual deployment in hospitals and diagnostic labs.

In summary, the future scope of the ExplaiLiver+ system lies in evolving it into a fully automated, real-time, and multimodal AI-driven diagnostic assistant that not only predicts liver disease but also supports doctors in making faster, data-backed, and explainable clinical decisions.

12. REFERENCES

1. Amin, R., Yasmin, R., Ruhi, S., Rahman, M. H., & Reza, M. S. (2023). Prediction of chronic liver disease patients using integrated projection based statistical feature extraction with machine learning algorithms. Informatics in Medicine Unlocked, 36, 101155.
2. Noor, S., AlQahtani, S. A., & Khan, S. (2025). Chronic liver disease detection using ranking and projection-based feature optimization with deep learning. AIMS Bioengineering, 12(1).
3. Ganie, S. M., & Pramanik, P. K. D. (2024). A comparative analysis of boosting algorithms for chronic liver disease prediction. Healthcare Analytics, 5, 100313.
4. Dritsas, E., & Trigka, M. (2023). Supervised machine learning models for liver disease risk prediction. Computers, 12(1), 19.
5. Jillani, N., Khattak, A. M., Asghar, M. Z., & Ullah, H. (2023, June). Efficient diagnosis of liver disease using deep learning technique. In 2023 IEEE International Symposium on Medical Measurements and Applications (MeMeA) (pp. 1–6). IEEE.
6. Noor, S., AlQahtani, S. A., & Khan, S. (2025). XGBoost-Liver: An Intelligent Integrated Features Approach for Classifying Liver Diseases Using Ensemble XGBoost Training Model. Computers, Materials & Continua, 83(1).
7. Osaseri, R. O., & Usiobaifo, A. R. (2024). Predicting liver Disease Using Support Vector Machine and Logistic Regression classification Algorithm. NIPES-Journal of Science and Technology Research, 6(4).
8. Rani, R., Jaiswal, G., Nancy, Lipika, Bhushan, S., Ullah, F., ... & Diwakar, M. (2025). Enhancing liver disease diagnosis with hybrid SMOTE-ENNbalanced machine learning models—an empirical analysis of Indian patient liver disease datasets. Frontiers in Medicine, 12, 1502749.
9. Tokala, S., Hajarathaiah, K., Gunda, S. R. P., Botla, S., Nalluri, L., Naga manohar, P., ... & Enduri, M. K. (2023). Liver disease prediction and classification using machine learning techniques.

International Journal of Advanced Computer Science and Applications, 14(2).
10. Rahman, A. S., Shamrat, F. J. M., Tasnim, Z., Roy, J., & Hossain, S. A. (2019). A comparative study on liver disease prediction using supervised machine learning algorithms. International Journal of Scientific & Technology Research, 8(11), 419–422.
11. Anthonysamy, V., & Babu, S. K. (2023). Multi Perceptron Neural Network and Voting Classifier for Liver Disease Dataset. IEEE Access, 11, 102149–102156
12. Prasad, J. V. D., Pratap, A. R., & Sallagundla, B. (2022). Machine learn ing based clinical diagnosis of liver patients with instance replacement. Journal of Mobile Multimedia, 18(2), 293-306.
13. Hallaji, E., Razavi-Far, R., Palade, V., & Saif, M. (2021). Adversarial learning on incomplete and imbalanced medical data for robust survival prediction of liver transplant patients. IEEE Access, 9, 73641-73650.
14. Nigatu, S. S., Alla, P. C. R., Ravikumar, R. N., Mishra, K., Komala, G., & Chami, G. R. (2023, May). A comparitive study on liver disease prediction using supervised learning algorithms with hyperparameter tuning. In 2023 International Conference on Advancement in Computa tion & Computer Technologies (InCACCT) (pp. 353-357). IEEE.
15. Mamun, M., Chowdhury, S. H., Hossain, M. M., Khatun, M. R., & Iqbal, S. (2025). Explainability enhanced liver disease diagnosis technique using tree selection and stacking ensemble-based random forest model. Informatics and Health, 2(1), 17-40.
16. M¨annikk¨o, V., Tommola, J., Tikkanen, E., H¨atinen, O. P., & ˚ Aberg, F. (2025). Large-Scale Evaluation and Liver Disease Risk Prediction in Finland's National Electronic Health Record System: Feasibility Study Using Real-World Data. JMIR Medical Informatics, 13(1), e62978.
17. Kumar, D., Bakariya, B., Verma, C., & Illes, Z. (2025).

LivXAI-Net: An explainable AI framework for liver disease diagnosis with IoT based real-time monitoring support. Computer Methods and Programs in Biomedicine, 108950.
18. Wang, Y., Lei, J., Jin, Z., Jiang, Y., Zhang, N., Lv, M., & Liu, T. (2025). Development and validation of a machine learning-based clinical prediction model for monitoring liver injury in patients with pan-cancer receiving immunotherapy. International Journal of Medical Informatics, 106036.
19. Liu, Y., Meric, G., Havulinna, A. S., Teo, S. M., ˚ Aberg, F., Ruuskanen, M., ... & Inouye, M. (2022). Early prediction of incident liver disease using conventional risk factors and gut-microbiome-augmented gradient boosting. Cell Metabolism, 34(5), 719-730.
20. Wu, C. C., Yeh, W. C., Hsu, W. D., Islam, M. M., Nguyen, P. A. A., Poly, T. N., ... & Li, Y. C. J. (2019). Prediction of fatty liver disease using machine learning algorithms. Computer methods and programs in biomedicine, 170, 23-29.

ExplaiLiver+: A Multi-Stage Stacking Framework with SHAP-Based Interpretability for Clinical Liver Disease Prediction

Gaddam Saranya[1], Pulagorla Mounica[2], Mahamkali Ramadevi[3], Valivarthi Abhinayasri[4], Karanam Madhavi[5], Divya Raj Vavilala[6], Dodda Venkata Reddy[7]
gaddamsaranya4@gmail.com[1], pulagorlalakshmi90@gmail.com[2], ramamahamkali18@gmail.com[3], abhivalivarthi@gmail.com[4], madhaviranjan@griet.ac.in[5], divyaraj.vavilala@gnits.ac.in[6] , doddavenkatareddy@gmail.com[7]
Department of Computer Science and Engineering, Narasaraopeta Engineering College[1,2,3,4,7], Yellamanda Road, Narasaraopet – 522601, Andhra Pradesh, India[1,2,3,4,7].
Department of CSE, GRIET, Hyderabad, Telangana, India[5].
Department of Electronics and Communication Engineering, G. Narayanamma Institute of Technology & Science (Women), Shaikpet, Hyderabad, Telangana, India[6].

***Abstract*—Improving patient outcomes and facilitating early medical intervention depend on timely and accurate liver disease prediction. In this work, we present ExplaiLiver+, a new multi-stage stacking ensemble framework that uses SHAP to combine interpretability of the model with high predictive performance. Robust preprocessing methods such as skewness correction, class balancing with SMOTEENN, feature selection using an ExtraTrees-based approach, and missing value imputation are all integrated into the framework. Four heterogeneous base classifiers—XGBoost, ExtraTrees, LightGBM, and CatBoost—stacked via a logistic regression meta-learner are used in the core ensemble architecture to improve generalization. ExplaiLiver+ outperforms individual baseline models with an AUC of 98.39% and a test accuracy of 94.05%, This study utilizes the ILPD for analysis. SHAP values are employed to illustrate feature importance and provide an explanation of individual predictions in order to guarantee decision-making transparency. The suggested system shows how clinical decision support systems for liver disease detection can be made much more reliable and trustworthy by fusing feature-level explainability with model-level ensemble learning.**

***Index Terms*—Liver disease prediction, stacking ensemble, SHAP, explainable AI (XAI), feature selection, SMOTEENN, clinical decision support system, XGBoost, CatBoost, LightGBM.**

I. Introduction

The liver is an essential and increasing multifunctional arbiter of digestion, metabolism, detoxification, and immune response. Even if the liver has an extraordinary capacity for regeneration, many liver diseases exist: Cirrhosis, viral hepatitis-related liver disease, NAFLD and liver cancer only represent about 4% of the total global burden of mortality, which is a total of around 2 million deaths a year [9], [1]. Chronic liver disease (CLD) is deceptively sneaking in a progressive disease category that are asymptomatic early in development, development of early clinical recognition and aversion ultimately impossible [4]. Conventional methods of diagnosis, including imaging scans, liver function tests (LFTs) and biopsies, are relatively invasive, often requires substantial monetary units and time, but can provide the relevant details of impairment for patient management [10], [9]. Innovations happening with different domains of artificial intelligence Deep learning (DL) and machine learning (ML) are likely inform the diagnosis and ultimately clinical care of bleeding edge low-cost, scalable, efficient and non-invasive perspective. Modern technologies emplicit, can now investigate considerably high-dimensional, high-complexity clinical datasets to elucidate characteristics that predict disease occurrence and the nature of disease severity [3], [7].To improve the accuracy of liver disease classification, numerous works [7], [3] have employed ML techniques, including SVM, RF, KNN, and ensemble-based models, with notable success. There is potential for performance [11] and interpretability [17] to be improved further by added feature optimization methods such as RFE, statistical projections, and Shapley Additive Explanations [2], [6], [8], [17]. It has been shown that deep learning models better capture the complex, nonlinear relationships present within patient data: MLP and BiLSTM networks have demonstrated better performance [5], [9]. Nonetheless, challenges such as noisy inputs, unbalanced datasets, and high feature dimensionality persist and often lead to biased predictions or overfitting of the model [8], [1]. Recent research has implemented hybrid approaches for enhancing prediction robustness and reliability, with a combination of ensemble learning frameworks, data resampling (e.g., SMOTE-ENN), and advanced feature engineering [8], [6]. This research will capitalize on the evolution of this area research and initiate a complete machine learning pipeline that includes improved feature selection techniques that are done with appropriate classification algorithms. This

study will evaluate model performance, using open source liver disease datasets and with conventional performance metrics. The study is primarily concerned with providing a clinically relevant, data-driven tool to help with early detection, reduce diagnosis turnaround and improve liver disease outcomes.

II. LITERATURE REVIEW

Liver disease diagnosis using the ML and DL methods have recently seen increased interest; primarily because of these substantial methodologies ability to assess large clinical datasets and to identify complex relations that often go unnoticed by traditional methods of diagnostic appointments. Aminetal. [1]. proposed have proposed incorporated statistical Method for extract relevant features that implemented feature extraction mostly part analysis (PCA), FA (frequentist analysis) and LDA (Linear Discriminant Analysis) for Cardiac disease Prediction In liver patient datasets and tested and reported accuracy of 88.10% which was greater than many others that were traditional method approaches. Noor et al. [2]. also used a deep learning model and improved it thorough the projections and ranking based features optimisations approach that had classification accuracy of 90.12% . SHAP values was also used to give model interpretability in terms of the SHAP values which highlighted the important features that impacted the predictions. Ensemble learning methods have also been extensively analyzed.

Ganie and Pramanik [3] compared seven different boosting algorithms (GB, XGBoost, CatBoost and LightGBM). They showed that GB achieved the highest accuracy (up to 98.80%) on the two liver datasets demonstrating how effective boosting methods are at learning in the capacity of clinical outcome prediction.

Dritsas and Trigka [4] explored Algorithmic models with labelled data concerning liver disease risk. Their results reinforced that ensemble classifier methods (focusing on Random Forest and AdaBoost) performed better than single classifier methods. They also emphasized how vital attribute relevance is to performance outcomes.

Jillani et al. [5] investigated BiLSTM a deep learning model that is able to learn temporal dependencies in the Health records. They achieved 93% accuracy, suggesting sometimes deep architecture will be required to model sequential patterns in the Health data.

Noor et al. [6] improved this by proposing their XGBoost-Liver model statistical characteristic selection for liver disease with boosting producing 92.07% accuracy and illustrating the synergy of characteristic engineering and ensemble learning.

In regards to classification, Osaseri and Usiobaifo [7], when comparing Logistic Regression (LR) and Support Vector Machines (SVM), found that not only was LR more accurate (97.24%) than SVM, but LR converged faster which means that for real time diagnoses, LR could be much more valuable.

In regards to the issue of imbalanced datasets, Rani et al. [8] proposed a hybrid model that included SMOTE-ENN, they also included ensemble classifiers - their hybrid model demonstrated considerably better prediction performance on the ILPD dataset with a 93.2% accuracy. There was a solid study that described feature selection methods a few of those methods included recursive feature elimination (RFE) which RFE appeared to be one of the more common methods.

On the contrary, Jyoshita et al. [9], looked at various deep gaining knowledge learning methods and found that Multi-Layer Perceptron (MLP) was the best model for the ILPD dataset they also implicated that the changes in urban lifestyle has been a huge factor in the rapid increase in liver disease in India.

Finally, Akram et al. [10] constructed a Liver Disease Prediction System using supervised mastering models and real patient records. They found that Random Forest produced the best prediction model with a 96% accuracy, they also found that feature perturbation was a way to manage the imbalanced dataset to generalize. Collectively, the outcomes demonstrate the effectiveness of ML and DL techniques in liver disease prediction. Despite newly identified Obstacles such as unequal class distribution, difficulties related to noise features, and the need for interpretability [16], a major trend in current research are hybrid models, Explainable AI (XAI) [17], and robust feature selection methods.

III. PROPOSED METHODOLOGY

Here, we describe the systematic approach used in developing the ExplaiLiver+ framework, which includes data description, preprocessing techniques, model ensemble design, evaluation strategies, and result visualization.

A. Dataset Description

We make use of the ILPD, which consists of 583 situations and 10 scientific aspects. The binary goal variable shows the presence (1) or absence (0) of liver disease. The dataset is as an alternative imbalanced, with about 70% of instances labeled as liver disease positive

TABLE I
DESCRIPTION OF FEATURES IN THE ILPD DATASET

Feature	Description
Gender	Biological sex of the subject
Age	Patient age range
TB	Concentration of TB present in the bloodstream
DB	Amount of direct bilirubin, which is water-soluble
ALP	Enzyme related to bile duct function
SGPT	Enzyme linked to liver cell damage
SGOT	Enzyme indicative of liver injury
TP	Total protein content in the blood
ALB	Protein produced by the liver
AGR	Ratio of albumin to globulin in serum

B. Data Preprocessing

To ensure the dataset was catchable for training, an extensive data preprocessing strategy was carried out

- **Handling Missing Values:** The `Albumin and Globulin Ratio` column contains missing entries. These are imputed using the median of the available values:

$$\text{Imputed Value} = \text{Median}(\text{A/G Ratio}) \quad (1)$$

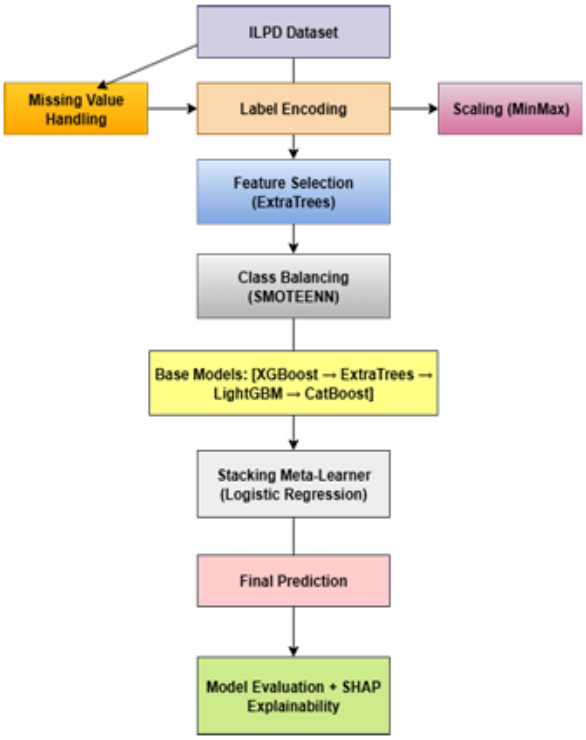


Fig. 1. Structural overview of the ExplaiLiver+ stacking ensemble framework designed for liver disease prediction.

This figure 2 illustrates the distribution of patient classes within the ILPD dataset. It highlights a noticeable class imbalance, with a higher number of liver disease cases compared to non-disease instances.

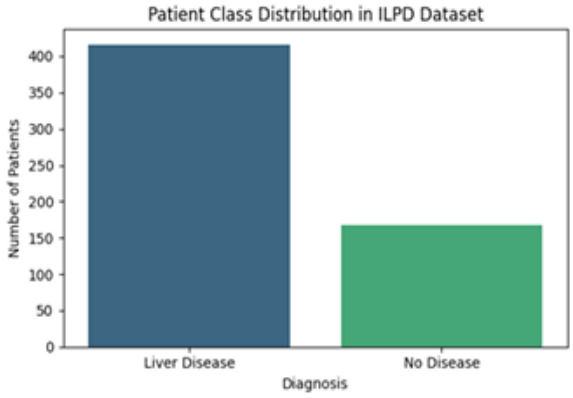


Fig. 2. Patient Class Distribution in the ILPD Dataset.

- **Label Encoding:** The categorical `Gender` column is converted into binary numeric format as follows [12], [19]:

$$\text{Gender (Male)} = 1, \quad \text{Gender (Female)} = 0 \tag{2}$$

- **Feature Scaling:** All continuous features are scaled to the range [0,1] using MinMax normalization:

$$x_{\text{scaled}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \tag{3}$$

- **Feature Selection:** The `ExtraTreesClassifier` is used to rank features by importance. Then, `SelectFromModel` retains only the top features based on the median threshold [12], [19], [20].
- **Skewness Correction:** To reduce skewness in certain features such as `Alkphos`, `SGPT`, and `SGOT`, logarithmic transformation is applied:

$$x' = \log(1 + x) \tag{4}$$

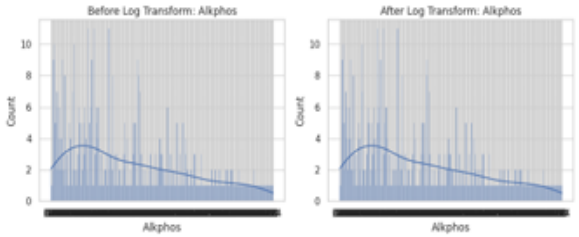


Fig. 3. Histogram showing the distribution of the selected skewed feature before and after applying logarithmic transformation.

The ILPD data set presented positive skewness across a number of features, most notably total and direct bilirubin. We selected the log transformation to reduce skewness and normalize feature distribution, especially since many machine-learning models are optimized with normally distributed inputs. Addressing For Imbalance: We used SMOTEENN [12] as it is the combination of SMOTE (the oversampling) and Edited Nearest Neighbors (removes some noise) to balance the data set [13]. We performed MinMax scaling on all the numerical features to change the data into the range [0, 1] [0, 1] [0, 1] for uniform/consistent number range to stabilize model convergence. Figure 4 shows us the scaled numerical feature distributions which reveal many different distributions for clinical features like age, TB, alkphos, and so on, solidifying the need for strong scaling prior to model fitting.

SMOTEENN Interpolation Formula

To address the class imbalance present in the ILPD dataset, we employed SMOTEENN. SMOTE [12] generates synthetic samples for the minority class by interpolating between a given minority instance and one of its k-nearest neighbors. The interpolation formula used is:

$$x_{\text{new}} = x_i + \delta \cdot (x_{\text{nn}} - x_i) \tag{5}$$

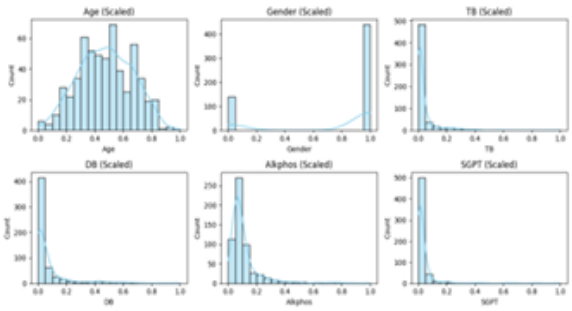


Fig. 4. Feature distribution plots after MinMax scaling of selected clinical attributes from the ILPD dataset.

Fig. 5. Class Distribution Before SMOTEENN

Fig. 6. Class Distribution After SMOTEENN

Figure 6: The correlation heat map from the ILPD dataset demonstrates linear relationships among clinical attributes. As the total bilirubin and direct bilirubin variables are biochemically dependent, it is unsurprising that they both show a strong positive correlation. Other variable types, such as enzymatic markers SGPT, SGOT, and ALP, were typically high and they all showed moderate correlation with each other in liver dysfunction. These three enzymes exhibit relationships that can inform overlapping diagnosis in liver dysfunction. Knowing these relationships can help to identify redundant features which might lead to model overfitting or model regularization, and it also aligns with selecting or methods for reducing the features dimensions. In addition, providing a clinical rationale or explanation for model selection or a specific model can be valuable. These trends also support the notion that many of the input features relate biologically, which strengthens confidence in the dataset overall for establishing prediction for liver disease [16].

Fig. 7. Correlation Heatmap of Clinical Features

C. Model Ensemble Design

The core architecture is a multi-stage stacking ensemble, named ExplaiLiver+. It consists of four high-performing base models [11]:

- XGBoost: Gradient-boosted decision trees with regularization [19]
- ExtraTrees: Averaging-based ensemble of randomized trees
- LightGBM: Histogram-based fast gradient boosting
- CatBoost: Efficient gradient boosting with categorical support [14]

After training the base learners individually, their predictions are concatenated with the original features and input into a Logistic Regression model that acts as the final decision-maker [18]. Mathematically, stacking can be expressed

as: h(x)=Meta(f1(x),f2(x),...,fn(x),x) We use 5-fold Stratified Cross-Validation internally in the stacking classifier for robust training.

D. Evaluation and Visualization

To assess model performance, we compute the following:

- **Accuracy:** Indicates the model's overall effectiveness by quantifying how frequently it makes correct predictions across both diseased and non-diseased cases.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{6}$$

- **Precision:** Indicates how many of the positively predicted cases are actually correct — useful when false positives are costly.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{7}$$

- **Recall (Sensitivity):** Assesses the model's ability to correctly identify true positive cases, which is crucial in medical settings to minimize missed diagnoses.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{8}$$

- **F1 Score:** A balanced average that combines precision and recall into a single metric, balancing both metrics — ideal when classes are imbalanced.

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{9}$$

- **AUC-ROC Score:** It indicates the probability of correctly distinguishing between the classes.

$$\text{AUC} = \int_0^1 TPR\left(FPR^{-1}(x)\right)\, dx \tag{10}$$

Collectively, these indicators offer a thorough assessment of how well the model identifies liver disease. Confusion Matrix: Shows the count of true positives, false positives, true negatives, and false negatives.

IV. RESULTS

Performance evaluation of the base classifiers in the stacking ensemble was conducted using five essential metrics: Accuracy, Precision, Recall, F1-Score, and AUC. These measures reflect the balance between detecting true cases and avoiding false alarms—an important consideration in medical applications like liver disease prediction. The performance of each base model used in the stacking ensemble is summarized in TABLE II.

The precision performance of each of the individual base learners—XGBoost, ExtraTrees, LightGBM, and CatBoost—powered the ensemble approach is illustrated in Fig. 9. Precision is important in medical diagnostics because it provides verification that the model correctly categorized actual positive instances; it describes the ratio of true positive predictions and all positive predictions and lowers false positives.

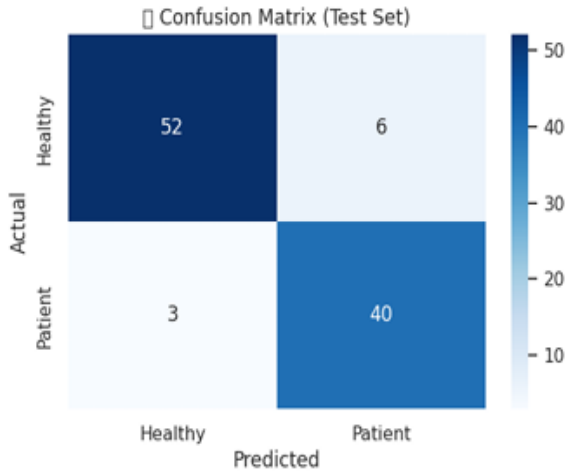


Fig. 8. Correlation Heatmap of Clinical Features

TABLE II
PERFORMANCE METRICS OF INDIVIDUAL MODELS ON LIVER DISEASE CLASSIFICATION

Model	Accuracy	Precision	Recall	F1-Score	AUC
XGBoost	0.9109	0.8696	0.9302	0.8989	0.9759
ExtraTrees	0.9109	0.9250	0.8605	0.8916	0.9824
LightGBM	0.9307	0.9091	0.9302	0.9195	0.9840
CatBoost	0.9307	0.9091	0.9302	0.9195	0.9840

The ExtraTrees Classifier had the highest precision score at 0.925 indicating the model had true positives for 92 of the 100 predicted liver disease cases. The other models were also able to give high precision scores over 0.86 showing their reliability to identify actual liver disease cases from positive high risk instance.

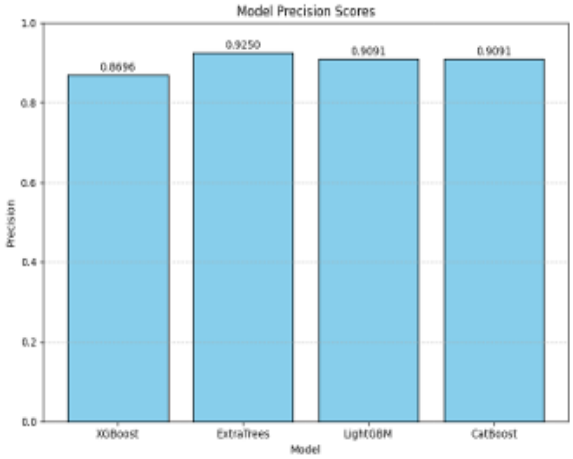


Fig. 9. Comparison of Precision across Different Models

The recall performance of the four classifiers used in the ensemble is presented in Fig. 10. Recall scores for each classifier were XGBoost, LightGBM, and CatBoost at 0.9302

(high recall implies high true positives) or true cases of liver disease detected; while the ExtraTrees Classifier had a lower recall score of 0.8605, implying a higher probability it presented missed true cases. Recall is critical in medicine because to avoid delaying treatment, if a case exists, it must be identified. Given the job functions of clinical decision support, if we consider that a clinical application requires high sensitivity, there is adequate information in the results to suggest the gradient-boosting-based approach (XGBoost, LightGBM, CatBoost) would be more appropriate.

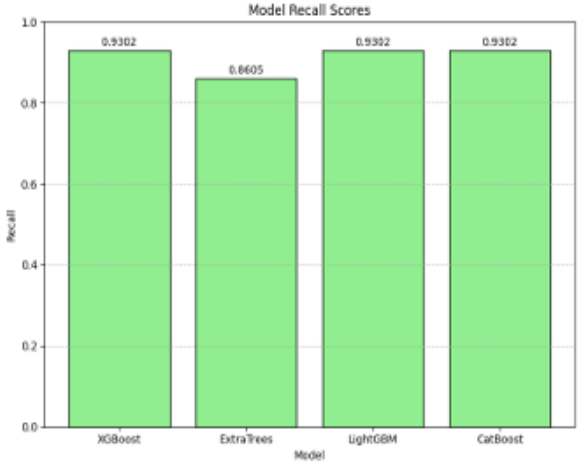


Fig. 10. Comparison of Recall across Different Models

The following F1-scores are found in Fig. 11. The F1-score merges precision and recall into one performance score. The LightGBM and CatBoost scored the highest F1-score of 0.9195, indicating that these models excelled at prediction of liver disease from non-disease state correctly. The XGBoost model achieved the next highest F1-score of 0.8989 with the ExtraTrees Classifier being slightly lower at 0.8916. A high F1-score indicates that the model is accurate in its predictions and consistent in predicting true positive cases and true negatives. In other words, these F1-scores verify the model performance of boosting-based models. In the context of clinical diagnostics, all the models exhibited the performance characteristics of balanced accuracy (respecting both false negative and false positive) which is important for determining if a test is useful.

The accuracy scores for the four ensemble models on the ILPD dataset are shown in Fig.12. Both LightGBM and Catboost achieved the highest accuracy of 93.07%. They appeared to be just as effective in determining true liver disease cases and non-disease cases. This was followed closely by XGBoost and ExtraTrees with accuracy scores of 91.09%. Thus, overall, high accuracy scores that suggest the models are able to generalize well over the test data and can be considered reliable. The slight competitive edge with LightGBM and Catboost suggests that exploring gradient-boosted frameworks for healthcare procedures that require exactness and reliability is a reasonable next step.

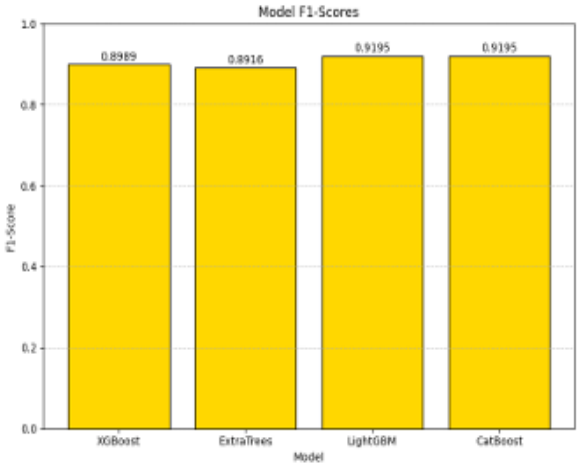


Fig. 11. Comparison of F1-Scores across Different Models

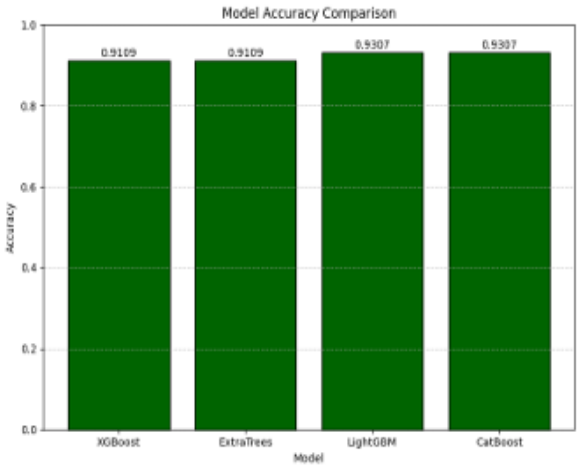


Fig. 12. Comparison of Accuracy across Different Models

Figure 13. Accuracy across 15 folds during Stratified K-Fold Cross-Validation of the Stacked Ensemble. The consistency of accuracy across folds indicates the robustness and generalization ability of the model.

To assess the robustness and consistency of the proposed models, a 15-fold stratified cross-validation was employed. This approach ensures each fold preserves the original class distribution, thereby enabling a fair evaluation of performance across all partitions. As illustrated in Figure 13, the accuracy achieved in each fold remains relatively stable, with minimal deviation, highlighting the model's resilience to training data splits and its consistent learning behavior. Following the cross-validation assessment, a comparative analysis between test accuracy and average cross-validation accuracy was conducted for each base learner and the final stacked ensemble. The results, presented in Figure 13, demonstrate that most models maintain a close alignment between test and validation performance, indicating reliable generalization. Notably, the stacked

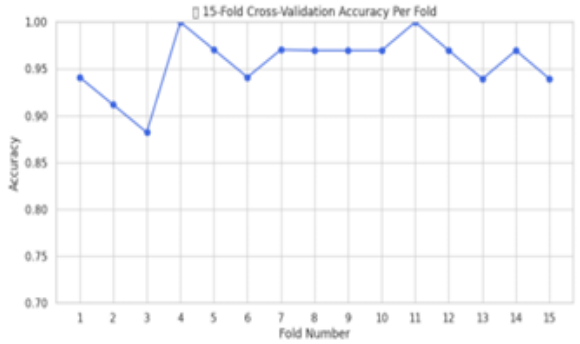


Fig. 13. 15-Fold Cross-Validation Accuracy per Fold

ensemble exhibited the highest test accuracy with minimal discrepancy from its cross-validation mean, confirming its ability to integrate complementary strengths of individual learners while reducing overfitting risk. This consistent performance across folds and generalization on unseen data collectively underscores the reliability of the proposed framework in clinical liver disease prediction tasks.

Test and Cross-Validation Accuracy Comparison for All Models (Figure 14). The robustness of the stacked ensemble is validated by the fact that it outperforms individual base learners and shows little variation between CV and test performance.

Fig. 14. Comparison of Test vs CV Accuracy across Base Models and the Stacked Ensemble

Overall, ExplaiLiver+ not only improves predictive accuracy (achieving 94%) but also ensures explainability, interpretability, and clinical relevance, making it a promising solution for early and reliable liver disease screening.

V. CONCLUSION

This paper proposed ExplaiLiver+, a novel multi-stage stacking ensemble framework for reliable and interpretable liver disease prediction. ExplaiLiver+ combines the strengths of XGBoost, LightGBM, ExtraTrees, and CatBoost ensemble methods, achieved great overall performance with an accuracy of 94% and AUC of 0.98. Strengths in pre-processing, feature selection, and balancing of classes contributed to

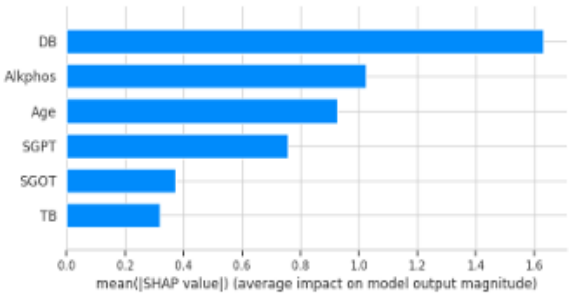


Fig. 15. Shap-Explainability

good overall generalization performance. The usage of SHAP explainability [15] also provides a level of clinical trust and interpretability. We believe ExplaiLiver+ provides an exciting opportunity for a real-world diagnostic decision support tool. Future studies can explore integration with real-time clinical systems using IoT-enabled monitoring. Incorporating larger, multi-center liver disease datasets may further enhance generalizability. Additionally, expanding the model to multi-class liver condition diagnosis can broaden its clinical relevance and application.

REFERENCES

[1] Amin, R., Yasmin, R., Ruhi, S., Rahman, M. H., & Reza, M. S. (2023). Prediction of chronic liver disease patients using integrated projection based statistical feature extraction with machine learning algorithms. *Informatics in Medicine Unlocked, 36*, 101155.

[2] Noor, S., AlQahtani, S. A., & Khan, S. (2025). Chronic liver disease detection using ranking and projection-based feature optimization with deep learning. *AIMS Bioengineering, 12*(1).

[3] Ganie, S. M., & Pramanik, P. K. D. (2024). A comparative analysis of boosting algorithms for chronic liver disease prediction. *Healthcare Analytics, 5*, 100313.

[4] Dritsas, E., & Trigka, M. (2023). Supervised machine learning models for liver disease risk prediction. *Computers, 12*(1), 19.

[5] Jillani, N., Khattak, A. M., Asghar, M. Z., & Ullah, H. (2023, June). Efficient diagnosis of liver disease using deep learning technique. In *2023 IEEE International Symposium on Medical Measurements and Applications (MeMeA)* (pp. 1–6). IEEE.

[6] Noor, S., AlQahtani, S. A., & Khan, S. (2025). XGBoost-Liver: An Intelligent Integrated Features Approach for Classifying Liver Diseases Using Ensemble XGBoost Training Model. *Computers, Materials & Continua, 83*(1).

[7] Osaseri, R. O., & Usiobaifo, A. R. (2024). Predicting liver Disease Using Support Vector Machine and Logistic Regression classification Algorithm. *NIPES-Journal of Science and Technology Research, 6*(4).

[8] Rani, R., Jaiswal, G., Nancy, Lipika, Bhushan, S., Ullah, F., ... & Diwakar, M. (2025). Enhancing liver disease diagnosis with hybrid SMOTE-ENN balanced machine learning models—an empirical analysis of Indian patient liver disease datasets. *Frontiers in Medicine, 12*, 1502749.

[9] Tokala, S., Hajarathaiah, K., Gunda, S. R. P., Botla, S., Nalluri, L., Nagamanohar, P., ... & Enduri, M. K. (2023). Liver disease prediction and classification using machine learning techniques. *International Journal of Advanced Computer Science and Applications, 14*(2).

[10] Rahman, A. S., Shamrat, F. J. M., Tasnim, Z., Roy, J., & Hossain, S. A. (2019). A comparative study on liver disease prediction using supervised machine learning algorithms. *International Journal of Scientific & Technology Research, 8*(11), 419–422.

[11] Anthonysamy, V., & Babu, S. K. (2023). Multi Perceptron Neural Network and Voting Classifier for Liver Disease Dataset. *IEEE Access, 11, 102149–102156.*

[12] Prasad, J. V. D., Pratap, A. R., & Sallagundla, B. (2022). Machine learning based clinical diagnosis of liver patients with instance replacement. *Journal of Mobile Multimedia, 18(2), 293-306.*

[13] Hallaji, E., Razavi-Far, R., Palade, V., & Saif, M. (2021). Adversarial learning on incomplete and imbalanced medical data for robust survival prediction of liver transplant patients. *IEEE Access, 9, 73641-73650.*

[14] Nigatu, S. S., Alla, P. C. R., Ravikumar, R. N., Mishra, K., Komala, G., & Chami, G. R. (2023, May). A comparitive study on liver disease prediction using supervised learning algorithms with hyperparameter tuning. *In 2023 International Conference on Advancement in Computation & Computer Technologies (InCACCT) (pp. 353-357). IEEE.*

[15] Mamun, M., Chowdhury, S. H., Hossain, M. M., Khatun, M. R., & Iqbal, S. (2025). Explainability enhanced liver disease diagnosis technique using tree selection and stacking ensemble-based random forest model. *Informatics and Health, 2(1), 17-40.*

[16] Männikkö, V., Tommola, J., Tikkanen, E., Hätinen, O. P., & Åberg, F. (2025). Large-Scale Evaluation and Liver Disease Risk Prediction in Finland's National Electronic Health Record System: Feasibility Study Using Real-World Data. *JMIR Medical Informatics, 13(1), e62978.*

[17] Kumar, D., Bakariya, B., Verma, C., & Illes, Z. (2025). LivXAI-Net: An explainable AI framework for liver disease diagnosis with IoT-based real-time monitoring support. *Computer Methods and Programs in Biomedicine, 108950.*

[18] Wang, Y., Lei, J., Jin, Z., Jiang, Y., Zhang, N., Lv, M., & Liu, T. (2025). Development and validation of a machine learning-based clinical prediction model for monitoring liver injury in patients with pan-cancer receiving immunotherapy. *International Journal of Medical Informatics, 106036.*

[19] Liu, Y., Meric, G., Havulinna, A. S., Teo, S. M., Åberg, F., Ruuskanen, M., ... & Inouye, M. (2022). Early prediction of incident liver disease using conventional risk factors and gut-microbiome-augmented gradient boosting. *Cell Metabolism, 34(5), 719-730.*

[20] Wu, C. C., Yeh, W. C., Hsu, W. D., Islam, M. M., Nguyen, P. A. A., Poly, T. N., ... & Li, Y. C. J. (2019). Prediction of fatty liver disease using machine learning algorithms. *Computer methods and programs in biomedicine, 170, 23-29.*

CERTIFICATE

1st IEEE Uttar Pradesh Section Women in Engineering International Conference on Electrical Electronics and Computer Engineering (UPWIECON-2025)

30th - 31st October 2025

Certificate of Presentation

This is to certify that Dr./Mr./Ms **Mounica Pulagorla** affiliated to **Narasaraopeta Engineering College** has presented the paper titled **ExplaiLiver+: A Multi-Stage Stacking Framework with SHAP-Based Interpretability for Clinical Liver Disease Prediction** in the 1st IEEE Uttar Pradesh Section Women in Engineering International Conference on Electrical Electronics and Computer Engineering (UPWIECON-2025) organized by NIELIT Dehradun held during 30th – 31st October 2025.