Cross-Climate Flood Prediction:

A Multi-Regional Deep Learning Approach to Hydrological Diversity

*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

**A. Navya (22471A05E4)**

**A.prudula Sri (22471A05E5)**

**K.Gandhi Kumari (22471A05G0)**

Under the esteemed guidance of

Dr.Rama Krishna Eluri, M.Tech.,Ph.D.

Associate Professor



#### DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NARASARAOPETA ENGINEERING COLLEGE: NARASAROPET

(AUTONOMOUS)

Accredited by NAAC with A+ Grade and NBA under Tyre -1 and an ISO 9001:2015 Certified

Approved by AICTE, New Delhi, Permanently Affiliated to JNTUK, Kakinada

KOTAPPAKONDA ROAD, YALAMANDA VILLAGE, NARASARAOPET- 522601

2025-2026

#### NARASARAOPETA ENGINEERING COLLEGE

**(AUTONOMOUS)**

#### DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

****

**CERTIFICATE**

**This is to certify that the project that is entitled with the name “Cross-Climate Flood Prediction: A Multi-Regional Deep Learning Approach to Hydrological Diversity” is a bonafide work done by A. Navya (22471A05E4), A. Prudula Sri (22471A05E5), K. Gandhi Kumari (22471A05G0) 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 2025-2026.**

PROJECT GUIDE PROJECT CO-ORDINATOR

**Dr.Rama Krishna Eluri, M.Tech.,Ph.D. D. Venkata Reddy, B.Tech., M.Tech., (Ph.D).**

###### Associate Professor Assistant Professor

HEAD OF THE DEPARTMENT EXTERNAL EXAMINER

**Dr. S. N. Tirumala Rao, M.Tech., Ph.D.**

**Professor & HOD**

### DECLARATION

We declare that this project work titled "**Cross-Climate Flood Prediction: A Multi-Regional Deep Learning Approach to Hydrological Diversity**" is composed by ourselves that the work contain here is our own except where explicitly stated otherwise in the text and that this work has been not submitted for any other degree or professional qualification except as specified.

**A. Navya (22471A05E4)**

**A. Prudula Sri (22471A05E5)**

**K.Gandhi Kumari (22471A05G0)**

### ACKNOWLEDGEMENT

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

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

We extend our sincere thanks to **D. Venkat Reddy, B.Tech., M.Tech., (Ph.D.),** Assistant Professor & Project 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.

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

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

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

**By**

**A. Navya (22471A05E4)**

**A. Prudula Sri (22471A05E5)**

**K.Gandhi Kumari (22471A05G0)**



### 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 knowledge of mathematics, natural science,computing, engineering fundamentals and an engineering specialization as specified in WK1 to WK4 respectively to develop to the solution of complex engineering problems.

**PO2: Problem Analysis**: Identify, formulate, review research literature and analyze complex engineering problems reaching substantiated conclusions with consideration for sustainable development. (WK1 to WK4)

**PO3: Design/Development of Solutions**: Design creative solutions for complex engineering problems and design/develop systems/components/processes to meet identified needs with consideration for the public health and safety, whole-life cost, net zero carbon, culture, society and environment as required. (WK5)

**PO4: Conduct Investigations of Complex Problems**: Conduct investigations of complex engineering problems using research-based knowledge including design of experiments, modelling, analysis & interpretation of data to provide valid conclusions.

(WK8).

**PO5: Engineering Tool Usage:** Create, select and apply appropriate techniques, resources and modern engineering & IT tools, including prediction and modelling recognizing their limitations to solve complex engineering problems. (WK2 and WK6)

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

**PO7: Ethics:** Apply ethical principles and commit to professional ethics, human values, diversity

and inclusion; adhere to national & international laws. (WK9)

**PO8: Individual and Collaborative Team work:** Function effectively as an individual,

and as a member or leader in diverse/multi-disciplinary teams.

**PO9: Communication:** Communicate effectively and inclusively within the engineering community and society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations considering cultural, language, and learning differences

**PO10: Project Management and Finance**: Apply knowledge and understanding of engineering management principles and economic decision-making and apply these to one’s own work, as a member and leader in a team, and to manage projects andin multidisciplinary environments.

**PO11: Life-Long Learning:** Recognize the need for, and have the preparation and ability for i) independent and life-long learning ii) adaptability to new and emerging technologies and iii) critical thinking in the broadest context of technological change.



### Project Course Outcomes (CO’S):

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

**CO421.2:** Identify and classify the requirements. **CO421.3:** Review the Related Literature **CO421.4:** Design and Modularize the project

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

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

**Course Outcomes – Program Outcomes mapping**

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

**Course Outcomes – Program Outcome correlation**

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

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

1. Low level
2. Medium level
3. High level

**Project mapping with various courses of Curriculum with Attained PO’s:**

|  |  |  |
| --- | --- | --- |
| **Name of the course from which principles are applied in this project** | **Description of the device** | **Attained PO** |
| C2204.2, C22L3.2 | Gathering the requirements and defining the problem, planning to develop a predictive model for flood occurrence and severity detection using hybrid deep learning models (CNN–LSTM–GRU). | PO1, PO3 |
| CC421.1, C2204.3, C22L3.2 | Each requirement is critically analyzed, and the data preprocessing, feature selection, and model architecture are identified based on hydrological and climatic diversity. | PO2, PO3 |
| CC421.2, C2204.2, C22L3.3 | Logical design is carried out using the Unified Modeling Language (UML), representing module interactions such as data preprocessing, model training, and flood prediction visualization. | PO3, PO5, PO9 |
| CC421.3, C2204.3, C22L3.2 | Each model (Random Forest, XGBoost, CNN, LSTM, GRU) is tested, integrated, and evaluated for performance metrics such as Accuracy, R², RMSE, and MAE. | PO1, PO5 |
| CC421.4, C2204.4, C22L3.2 | Documentation is collaboratively prepared by all team members, covering methodology, testing, results, and analysis of the flood prediction system. | PO10 |
| CC421.5, C2204.2, C22L3.3 | Each phase of the project — from data collection to model deployment — is presented periodically as a group to demonstrate progress and validate outcomes. | PO10, PO11 |
| C2202.2, C2203.3, C1206.3, C3204.3, C4110.2 | Implementation is completed successfully; the model predicts flood risks across multiple regions, with future scope for real-time updates and integration with climate monitoring systems. | PO4, PO7 |
| C32SC4.3 | The physical design includes a visualization interface displaying flood predictions, regional risk levels, and performance comparisons of various models. | PO5, PO6 |

### ABSTRACT

Floods are among the most destructive and recurring natural disasters that significantly affect human lives, infrastructure, agriculture, and ecosystems across diverse climatic regions. Predicting floods accurately remains a major challenge due to the complex interactions between meteorological, hydrological, and environmental factors. This study proposes a scalable deep learning-based framework for multi-regional flood prediction that integrates multiple heterogeneous data sources such as rainfall, temperature, soil moisture, land environment, and socio-environmental indicators. To ensure uniformity and consistency in both spatial and temporal dimensions, several preprocessing techniques—temporal alignment, linear interpolation, and Min-Max normalization—were employed to refine the data before model training. Five predictive models, namely Random Forest, XGBoost, Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM), and Gated Recurrent Unit (GRU) were developed and analyzed for their flood forecasting capabilities. Comparative results demonstrated that deep learning models, particularly the GRU, outperformed traditional machine learning techniques in terms of accuracy, reliability, and generalization. The proposed GRU model achieved 99.63% accuracy, R² = 0.8986, RMSE = 0.1555, and MAE = 0.0820, highlighting its strong predictive performance across different climatic zones and environmental conditions. The study concludes that this GRU-based framework can serve as an effective tool for operational flood forecasting, early warning systems, climate change adaptation, and disaster risk reduction, ultimately aiding policymakers, planners, and disaster management authorities in minimizing flood-related damages and ensuring community resilience.

S.NO CONTEXT

### INDEX

PAGE NO

1. [INTRODUCTION 1](#_TOC_250036)
   1. MOTIVATION 3
   2. PROBLEM STATEMENT 4
   3. OBJECTIVE 5
2. LITERATURE SURVEY 6
3. [SYSTEM ANALYSIS](#_TOC_250031)
   1. EXISTING SYSTEM 9
      1. DISADVANTAGES OF THE EXISTING SYSTEM 11
   2. PROPOSED SYSTEM 13
   3. FEASIBILITY STUDY 15
   4. USING COCOMO MODEL 16
4. [SYSTEM REQUIREMENTS](#_TOC_250026)
   1. SOFTWARE REQUIREMENTS 18
   2. REQUIREMENT ANALYSIS 18
   3. HARDWARE REQUIREMENTS 19
   4. SOFTWARE 19
   5. SOFTWARE DESCRIPTION 20
5. [SYSTEM DESIGN](#_TOC_250020)
   1. SYSTEM ARCHITECTURE 21
      1. DATASET 22
      2. DATA PREPROCESSING 24
      3. FEATURE EXTRACTION 27
      4. MODEL BUILDING 28
      5. CLASSIFICATION 31
   2. MODULES 33
   3. UML DIAGRAMS 38
6. [IMPLEMENTATION](#_TOC_250011)
   1. MODEL IMPLEMENTATION 42
   2. CODING 45
7. [TESTING](#_TOC_250008)
   1. UNIT TESTING 74
   2. INTEGRATION TESTING 75
   3. SYSTEM TESTING 76
8. RESULT ANALYSIS 78
9. OUTPUT SCREENS 81
10. CONCLUSION 84
11. FUTURE SCOPE 85
12. REFERENCES 86

### LIST OF FIGURES

**S.NO FIGURE DESCRIPTION**

1 FIG 3.2 FLOWCHART OF PROPOSED SYSTEM

**PAGE NO**

13

1. FIG 5.1 CLEANED PROBABILITY TIME SERIES 25
2. FIG 5.2 HOURLY ALIGNED PROBABILITY 25
3. FIG 5.3 FEATURE DISTRIBUTION BEFORE & AFTER SCALING 26
4. FIG 5.4 FLOOD VS NO FLOOD 26
5. FIG 5.3 UML DIAGRAM FOR FLOOD PREDICTION 39
6. FIG 5.5 CNN MODEL ARCHITECTURE 31
7. FIG 8.1 GRU MODEL-ACTUAL VS PREDICTED 78
8. FIG 8.2 GRU-CONFUSION MATRIX 79
9. FIG 8.3 MODEL PERFORMANCE COMPARISON 80
10. FIG 9.1 HOME PAGE 81
11. FIG 9.2 ABOUT PAGE 82
12. FIG 9.3 OBJECTIVES PAGE 82
13. FIG 9.4 PROCEDURE PAGE 83
14. FIG 9.5 VALIDATION PAGE 83

|  |  |  |
| --- | --- | --- |
|  | **List of Tables** |  |
| S.NO | CONTENT | PAGE NO |
| 1 | Table 3.3: Feasibility Study Summary of the Proposed Flood prediction | 16 |
| 2 | TABLE 5.1: DATASET DESCRIPTION | 23 |
| 3 | TABLE 8.1. MODEL PERFORMANCE COMPARISON | 80 |

### INTRODUCTION

Flooding remains one of the most destructive and recurring natural disasters, significantly affecting lives, infrastructure, agriculture, and ecosystems across the globe. Over the past few decades, both the frequency and severity of flood events have escalated due to climate change, uncontrolled urbanization, deforestation, and altered precipitation patterns. According to global hydrological assessments, extreme weather events are now more frequent, with floods accounting for nearly 40% of all natural disasters worldwide. In a geographically diverse country like India, where climatic conditions vary from arid to tropical, flood prediction poses a unique challenge due to highly non-linear interactions between climatic and hydrological factors.

Traditional flood forecasting models have relied on empirical, statistical, or rule-based techniques such as regression analysis, time-series methods, or hydrodynamic simulations. While these methods offer interpretability, they often fail to generalize across multiple climatic zones due to their dependence on localized data and limited adaptability. Moreover, issues such as missing or inconsistent datasets, temporal irregularities, and noisy satellite observations further reduce their predictive accuracy, making them unsuitable for large-scale, real-time flood forecasting applications.

The emergence of Artificial Intelligence (AI) and Deep Learning (DL) has revolutionized hydrological modeling, enabling data-driven systems to automatically extract complex spatial and temporal relationships. Advanced architectures such as Convolutional Neural Networks (CNNs), Long Short-Term Memory (LSTM), Gated Recurrent Units (GRUs), and hybrid CNN-LSTM models have demonstrated remarkable potential in capturing non-linear dependencies between rainfall, temperature, soil moisture, and river discharge. These models offer improved feature extraction and temporal sequence learning, producing high-resolution flood predictions with better generalization capabilities.

However, despite their success, existing deep learning-based flood models often face challenges related to computational complexity, overfitting, and regional scalability. Most are trained using limited datasets, making them less transferable across diverse climatic regions or variable environmental conditions. Addressing these limitations requires a multi-regional, cross-climate flood prediction framework that can adapt to heterogeneous data inputs and generalize across spatial and temporal domains.

To bridge this gap, the present study proposes a Cross-Climate Flood Prediction System — a multi-regional deep learning framework that integrates hydrological, climatic, and anthropogenic features for improved flood forecasting accuracy. The dataset combines multi-source information from NASA POWER, SMAP, MODIS, and ground-level socio-environmental indicators to represent critical variables such as Urbanization, Climate Change, and Dams Quality. Advanced data preprocessing techniques, including temporal synchronization, normalization, and outlier handling, are applied to enhance the reliability and consistency of data across regions.

A comprehensive comparative analysis is conducted using machine learning models (Random Forest, XGBoost) and deep learning architectures (CNN, LSTM, GRU). The experimental outcomes reveal that the GRU model achieved the highest overall performance with 99.63% accuracy, R² = 0.8986, RMSE = 0.1555, and MAE = 0.0820, demonstrating superior temporal learning capability and robustness across diverse climatic regions.

The findings of this research contribute significantly to the field of climate-resilient flood forecasting, offering a scalable, interpretable, and high-accuracy prediction framework that supports real-time early warning systems and disaster management initiatives. By integrating multi-source environmental data with deep learning-based architectures, the proposed model establishes a new benchmark for cross-climate flood prediction, paving the way for smarter, data-driven climate adaptation strategies and sustainable water resource management.

##### Motivation

Floods are among the most destructive natural disasters, causing severe loss of life, property damage, and long-term environmental and economic impacts. Their increasing frequency and intensity are primarily driven by climate change, urbanization, and unpredictable weather patterns. Despite progress in hydrological and meteorological modeling, accurate and timely flood prediction remains a major global challenge due to the complex, non-linear relationships among factors such as rainfall, temperature, soil moisture, and land use.

Traditional flood forecasting techniques, which rely on statistical and physical models, often struggle to generalize across diverse climatic and geographic regions. They require extensive calibration and lack scalability for real-time applications. In contrast, machine learning and deep learning approaches offer promising solutions by learning intricate spatio-temporal dependencies from large-scale environmental data. However, many existing ML-based systems still focus on region-specific or short-term forecasts, limiting their robustness and transferability.

This research is motivated by the need to develop a scalable, data-driven flood prediction framework that integrates multiple environmental and climatic data sources to enhance accuracy and adaptability. By leveraging deep learning models such as CNN, LSTM, and GRU, and applying effective preprocessing techniques like temporal alignment and normalization, the proposed system aims to deliver reliable, multi-regional flood forecasts. Ultimately, this work supports early warning systems, disaster preparedness, and climate resilience, contributing to sustainable environmental management and global flood risk reduction.

In addition, this study aims to bridge the gap between traditional flood modeling and modern artificial intelligence by combining multi-source environmental intelligence with predictive analytics. The framework not only enhances forecasting performance but also provides valuable insights for policymakers, urban planners, and disaster management authorities. By enabling faster and more accurate decision-making, the proposed system contributes toward building smart, climate-resilient communities and aligns with the global objectives of sustainable development and disaster risk reduction.

##### Problem Statement

Brain Flood prediction is a complex and multidimensional challenge influenced by various meteorological, hydrological, and environmental factors such as rainfall intensity, temperature variation, soil moisture, and land use patterns. Traditional flood forecasting models, which rely heavily on statistical and physical methods, often fail to capture the non-linear and dynamic relationships between these variables. Moreover, these models are typically region-specific, requiring localized calibration and extensive manual intervention, which limits their scalability and real-time applicability across diverse climatic zones.

In addition, existing machine learning approaches, while offering improvements in accuracy, often depend on limited datasets and lack the ability to generalize effectively across multiple regions with varying environmental conditions. This leads to inconsistencies in predictions and reduces their reliability for operational flood warning systems. The absence of a unified, multi-regional framework that can process and analyze heterogeneous environmental data sources remains a major bottleneck in the development of robust flood forecasting solutions.

Therefore, there is a critical need to design a scalable, data-driven deep learning framework that can effectively learn complex spatio-temporal patterns from multi-source environmental data. Such a system should be capable of delivering accurate, generalizable, and real-time flood predictions to support early warning systems, enhance disaster preparedness, and contribute to sustainable climate resilience and risk reduction efforts.

##### Objectives

The primary objective of this research is to design and implement a multi-regional, data-driven flood prediction framework using advanced deep learning techniques that enhance the accuracy, scalability, and interpretability of flood forecasting across diverse climatic and geographical regions. The framework integrates heterogeneous environmental datasets to identify complex spatio-temporal patterns that traditional models fail to capture.

The specific objectives of the study are as follows:

1. To collect and integrate multi-source environmental and climatic datasets from NASA POWER, SMAP, MODIS, and socio-environmental databases to ensure comprehensive data coverage across multiple regions.
2. To perform detailed data preprocessing, including temporal alignment, data interpolation, feature selection, and normalization, to ensure consistency, completeness, and reliability of the datasets.
3. To explore and develop multiple machine learning and deep learning models—including Random Forest, XGBoost, CNN, LSTM, and GRU—for flood prediction and comparative performance analysis.
4. To design a scalable GRU-based framework capable of capturing long-term temporal dependencies and spatial correlations among environmental variables for improved flood forecasting accuracy.
5. To evaluate the performance of each model using statistical metrics such as Accuracy, Root Mean Square Error (RMSE), Mean Absolute Error (MAE), and Coefficient of Determination (R²).
6. To assess the generalizability and adaptability of the proposed framework across different climatic regions and seasonal variations to ensure real-world applicability.
7. To develop a robust and interpretable system that provides actionable insights for policymakers, disaster management authorities, and environmental planners.
8. To contribute toward sustainable flood management and early warning systems by enabling proactive response and reducing the socio-economic impacts of floods.

#### LITERATURE SURVEY

Flood prediction has become an essential research area in environmental science and disaster risk management due to the increasing frequency and severity of floods caused by climate change, urbanization, and irregular precipitation patterns. Traditional hydrological and meteorological forecasting techniques, such as regression-based models and process-driven approaches, have long been employed to simulate rainfall–runoff relationships and river discharge levels. These models, including hydrodynamic systems and physical equations, depend on prior watershed and climatic data but often fail to generalize across diverse climatic zones due to parameter uncertainty and data scarcity [1]. Consequently, there has been a significant shift toward data-driven models that can automatically learn complex relationships among multiple hydrological and environmental factors.

In this context, Machine Learning (ML) approaches such as Support Vector Machines (SVM), Random Forest (RF), K-Nearest Neighbors (KNN), and Artificial Neural Networks (ANN) have been widely adopted for flood forecasting tasks. These models have shown the ability to identify non-linear correlations between rainfall, river discharge, soil moisture, and land cover [2].

*Yaseen et al.* (2019) demonstrated that SVM and ANN models outperform conventional regression-based hydrological models in short-term river flood forecasting [3].

*Mosavi et al.* (2020) implemented hybrid RF–SVM models to enhance rainfall–runoff estimation, achieving improved accuracy in data-driven hydrological simulations [4]. However, despite these advancements, traditional ML models still rely on manual feature engineering and are limited in capturing spatio-temporal dependencies crucial for real-time flood forecasting.

The emergence of Deep Learning (DL) has transformed flood prediction research by enabling models to automatically learn hierarchical representations from large-scale environmental data. Convolutional Neural Networks (CNNs), known for their spatial feature extraction capabilities, have been applied to satellite imagery and meteorological maps for flood detection and mapping. *Rahman et al.* (2021) utilized a CNN-based architecture for flood segmentation using Sentinel-1 SAR images and achieved an accuracy exceeding 95% [5].

*Kratzert et al.* (2018) demonstrated that Long Short-Term Memory (LSTM) networks outperform traditional rainfall–runoff models due to their superior ability to capture temporal dependencies in hydrological time series [6].

Recent advancements have explored hybrid deep learning architectures combining the spatial learning ability of CNNs with the temporal modeling power of recurrent networks such as LSTM and GRU. Le et al. (2020) introduced a CNN–LSTM framework that simultaneously captured rainfall patterns and flood temporal dynamics, outperforming standalone CNNs and LSTMs by 8–12% in predictive accuracy [7].

*Feng et al.* (2021) developed a ConvLSTM model that fused convolutional and recurrent layers, effectively representing spatio-temporal flood progression in multi-source datasets [8]. Ahmed et al. (2022) proposed a Gated Recurrent Unit (GRU) model that maintained high accuracy while significantly reducing computational complexity, making it suitable for real-time flood early warning systems [9].

In addition to architecture-based improvements, recent studies have leveraged transfer learning and ensemble methods to enhance model robustness in data-scarce regions. *Tiwari et al.* (2023) demonstrated that fine-tuning pre-trained CNN architectures such as VGG16 and ResNet-50 significantly improved flood detection accuracy while reducing training time [10].

*Liu et al.* (2022) further explored multi-model ensemble learning, integrating predictions from CNN, LSTM, and Random Forest models to improve spatial generalization across multiple climatic zones [11].

Integrating remote sensing data, geographical information systems (GIS), and socio-environmental indicators has become a major trend in recent research. Jia et al. (2021) used Graph Neural Networks (GNNs) with multi-source data to generate 3D flood risk maps, achieving high-resolution spatial flood visualization [12].

*Atashi and Gorji* (2022) employed LSTM-based models to predict snowmelt-induced floods in the Red River basin using meteorological variables, achieving strong temporal correlation with observed flood events [13]. These studies highlight the versatility of deep learning in handling heterogeneous datasets for robust and scalable flood forecasting applications.

Overall, the literature suggests that deep learning-based models, particularly GRU, LSTM, and hybrid CNN–LSTM architectures, outperform conventional ML and physical models in flood prediction tasks due to their ability to learn non-linear, spatio-temporal relationships from large, diverse datasets. However, challenges such as data imbalance, lack of global generalization, and limited interpretability remain key research gaps. Addressing these limitations through explainable AI, domain adaptation, and fusion of multi-regional data sources will be essential for developing next-generation, real-time flood early warning systems.

Despite these advancements, several research gaps persist in the domain of flood prediction using deep learning. Most existing studies are region-specific, focusing on single-basin or localized datasets, which limits their generalization to diverse climatic and geographical regions [14]. Many models depend solely on hydrological or meteorological variables, overlooking socio-environmental and land-use factors that significantly influence flood vulnerability [15]. Another limitation lies in the lack of standardized multi-source data integration, as the fusion of rainfall, soil moisture, temperature, and terrain data remains inconsistent across studies [16]. While deep learning models such as CNN, LSTM, and GRU have achieved remarkable accuracy, they often operate as “black-box” systems with limited interpretability, making it difficult for domain experts to validate and trust their predictions [17]. Therefore, there is a growing need for scalable, explainable, and multi-regional flood prediction frameworks that can combine spatial–temporal deep learning architectures with effective preprocessing and feature fusion techniques to enhance reliability and generalization. The current study addresses these challenges by proposing a unified GRU-based deep learning model that integrates heterogeneous environmental datasets to provide accurate, adaptable, and interpretable flood forecasts.

### SYSTEM ANALYSIS

#### EXISTING SYSTEM

Flood prediction and monitoring have traditionally relied on hydrological and meteorological models developed by environmental scientists and engineers. These conventional approaches depend heavily on mathematical equations and physical parameters such as rainfall intensity, river discharge, soil moisture, and land use patterns. While these models have been effective for localized flood forecasting, they often require extensive domain knowledge, manual calibration, and region-specific tuning, making them less adaptable to varying climatic conditions. Furthermore, the accuracy of these systems is limited by data sparsity, non-linear environmental interactions, and the increasing unpredictability of weather patterns due to climate change.

Early computational models used statistical and empirical techniques such as Regression Analysis, ARIMA (Auto-Regressive Integrated Moving Average), and Time Series Analysis to predict flood levels based on historical data. Although these models performed well under steady conditions, they struggled to generalize when applied to different geographical areas with distinct hydrological properties. The inability of these models to capture complex spatio-temporal dependencies restricted their use in large-scale, real-time flood forecasting systems.

With advancements in technology, Machine Learning (ML) algorithms such as Support Vector Machines (SVMs), Random Forests (RFs), and Artificial Neural Networks (ANNs) have been introduced for flood prediction. These models learn from data patterns rather than fixed equations, enabling improved adaptability to diverse conditions. However, their performance often depends on the quality and quantity of training data, and they may fail to effectively model sequential temporal dependencies between climatic variables. Additionally, many ML-based systems are region-specific, limiting their applicability in cross-climate scenarios.

The emergence of Deep Learning (DL) techniques, particularly Convolutional Neural Networks (CNNs), Long Short-Term Memory (LSTM) networks, and Gated Recurrent Units (GRUs), has significantly enhanced the ability to capture complex non-linear relationships between environmental factors. CNNs are effective in extracting spatial features such as rainfall distribution and topographical variation, while LSTM and GRU architectures excel in understanding temporal patterns in time-series data. Despite these advantages, deep learning models often require large labeled datasets and high computational resources, which can restrict real-time applications.

To overcome these challenges, recent studies have explored hybrid and ensemble models that integrate CNN and LSTM architectures for simultaneous spatial-temporal learning. These models achieve higher prediction accuracy and robustness across multiple regions. However, most existing systems are still limited by the lack of generalization across climates, insufficient data preprocessing, and the absence of multi-source integration from hydrological, meteorological, and satellite data streams.

The process begins with the collection of input data from multiple sources such as rainfall records, river gauge measurements, temperature data, and satellite imagery. The pre-processing stage follows, where missing values are handled, noise is reduced, and normalization techniques are applied to ensure data consistency across temporal and spatial scales.

Next, the feature extraction phase focuses on identifying critical flood-related variables such as precipitation intensity, water level rise, soil saturation, and land cover changes. These features are either manually engineered or automatically extracted through deep learning architectures. The model training and prediction phase then utilizes algorithms like Random Forests, SVMs, LSTMs, or hybrid CNN-LSTM models to forecast flood occurrences and intensities.

Finally, the evaluation stage assesses model performance using metrics such as Accuracy, RMSE (Root Mean Square Error), Precision, Recall, and F1-score. The results are then visualized through dashboards or GIS interfaces for real-time monitoring and decision-making. Although this pipeline enhances flood preparedness, it remains constrained by data dependency, limited scalability, and inadequate adaptability to multi-regional variations in climate and terrain.

#### DISADVANTAGES OF THE EXISTING SYSTEM FOR FLOOD PREDICTION

#### 1. Limited Regional Generalization:

Most traditional and machine learning-based flood models are region-specific and fail to generalize across diverse climatic and geographical zones, reducing their reliability for global or multi-regional applications.

**2. High Dependence on Historical Data:**

Existing systems rely heavily on past rainfall and river flow records, which may not represent current or future climatic conditions affected by global warming and land-use changes.

**3. Lack of Real-Time Processing:**

Many conventional systems cannot process or analyze data in real time, leading to delayed flood warnings and ineffective disaster management.

**4. Manual Calibration Requirement:**

Hydrological and statistical models require extensive manual calibration for each new region, making the process time-consuming and less scalable.

**5.** **Inability to Capture Non-Linear Relationships:**

Traditional models such as regression or SVMs struggle to capture complex, non-linear interactions among environmental variables like rainfall, temperature, soil moisture, and terrain.

**6. Poor Data Integration:**

Existing systems often fail to combine multiple data sources—such as satellite imagery, sensor data, and weather observations—resulting in incomplete flood assessments.

**7.** **Limited Spatial and Temporal Resolution:**

Older models typically produce coarse-grained outputs that lack precise spatial and temporal details necessary for localized flood predictions.

**8.** **High Computational Complexity:**

Some numerical and hydrodynamic models require large computational resources and long simulation times, making them impractical for real-time use.

**9**. **Inadequate Handling of Missing or Noisy Data:**

Traditional systems are sensitive to incomplete or noisy datasets, which can lead to inaccurate or unstable predictions.

**10. Lack of Adaptive Learning:**

Most existing methods cannot automatically adapt to changing weather patterns or new environmental data, limiting their ability to maintain accuracy over time.

#### PROPOSED SYSTEM

##### The proposed model employs a hybrid deep learning framework integrating Convolutional Neural Networks (CNN) [1], Long Short-Term Memory (LSTM) [2], and Gated Recurrent Units (GRU) [3] to improve the accuracy and generalization of multi-regional flood prediction. This combination leverages the spatial feature extraction capabilities of CNN and the temporal dependency learning strengths of LSTM and GRU. Additionally, ensemble machine learning models such as Random Forest [4] and XGBoost [5] are incorporated for comparative analysis to validate the robustness of the proposed system.

##### FIG 3.2. FLOW CHART OF PROPOSED SYSTEM

##### The workflow in Fig. 3.2 begins with the data collection phase, where hydrological, meteorological, and environmental data — including rainfall, temperature, humidity, soil moisture, and river discharge — are gathered from multiple climatic regions. The collected data undergoes preprocessing steps such as handling missing values [6], feature scaling [7], label encoding, and train-test splitting to ensure uniformity and reduce noise in the dataset.

##### After preprocessing, feature extraction is performed using CNN to capture spatial relationships among geographical parameters, while LSTM and GRU networks learn temporal patterns from sequential data. This hybrid approach ensures both spatial and temporal dependencies are effectively modeled, resulting in improved prediction stability and transferability across different regions.

##### To enhance model generalization, dropout layers [8] are integrated between deep learning layers to prevent overfitting. The processed data then passes through dense layers to produce the final output — representing the probability of flood occurrence based on environmental conditions. For performance benchmarking, Random Forest and XGBoost classifiers are trained in parallel using the same dataset to evaluate efficiency, interpretability, and prediction speed.

##### The output of the proposed model delivers accurate and real-time flood forecasts with improved precision and reduced false alarm rates. By combining multiple learning paradigms and advanced preprocessing, this system aims to enhance disaster preparedness, early warning mechanisms, and sustainable climate resilience.

##### Advantages over existing system:

1. **Higher Accuracy:**

The proposed deep learning model (CNN-LSTM-GRU) achieves significantly better accuracy compared to traditional hydrological and statistical models used in the existing system.

1. **Multi-Regional Generalization:**

Unlike existing models that are region-specific, the proposed system can adapt to multiple climatic zones and varying environmental conditions.

1. **Automated Feature Learning:**

The proposed system automatically extracts relevant spatial and temporal features from raw data, eliminating the need for manual feature engineering required in older methods.

1. **Real-Time Prediction Capability:**

The integration of advanced neural networks allows near real-time flood forecasting, which was difficult in the existing, computation-heavy statistical approaches.

1. **Reduced Overfitting:**

By using regularization techniques, dropout layers, and large-scale multi-regional datasets, the proposed system effectively minimizes overfitting, ensuring better generalization on unseen data.

1. **Improved Scalability and Flexibility:**

The architecture supports integration of additional data sources such as satellite imagery and IoT sensor data, enabling broader and scalable flood prediction applications.

#### FEASIBILITY STUDY

A feasibility study is an essential step in the system development process that evaluates the practicality, sustainability, and overall viability of the proposed project. It determines whether the system can be developed and deployed successfully with the available resources, within the given constraints, and with the expected benefits. For the proposed Cross-Climate Flood Prediction Using Deep Learning model, feasibility is examined under four key perspectives:

**Technical Feasibility:**

The project utilizes advanced deep learning algorithms such as Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) models, supported by open-source frameworks like TensorFlow and PyTorch. The datasets include rainfall, temperature, humidity, and soil moisture data obtained from publicly available sources such as IMD and NASA Earth Observation Data. With access to GPU-enabled platforms like Google Colab or Kaggle Notebooks, all technical requirements for model training, evaluation, and deployment are efficiently met.

**Operational Feasibility:**

The system is designed for real-time flood prediction and can be integrated into disaster management dashboards, weather monitoring centers, and early warning systems. The interface provides visualized predictions and alerts, enabling easy interpretation by government officials, meteorologists, and local authorities. Additionally, the system’s modular design allows scalability to cover multiple regions and weather conditions, ensuring smooth operational adaptability.

**Economic Feasibility:**

The project is cost-effective as it leverages free datasets, open-source libraries, and cloud-based resources for computation. While deep learning training may require high-performance hardware, using Google Colab Pro or low-cost GPU cloud instances minimizes infrastructure expenses. The long-term benefits—such as reduced flood damage, early risk mitigation, and improved planning—significantly outweigh the initial setup costs, making the system economically viable.

**Social Feasibility:**

The system contributes to disaster preparedness and public safety by predicting floods with higher accuracy across diverse climatic regions. It aids in reducing loss of life and property and promotes sustainable development practices aligned with UN SDG 13 (Climate Action). By empowering authorities and communities with timely alerts, the project demonstrates strong social relevance and impact.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Feasibility Type** | **Description** | | **Outcome** | |
| Technical  Feasibility | Utilizes deep learning models (CNN, LSTM), open-source frameworks (TensorFlow, PyTorch), and GPU-enabled cloud environments for efficient implementation. | | Highly Feasible | |
| Operational Feasibility | Supports real-time predictions and integrates with flood monitoring dashboards for user-friendly operation and scalability. | Feasible | |
| Economic Feasibility | Employs free datasets, open-source tools, and affordable cloud platforms, ensuring minimal cost and maximum benefit. | | Cost-Effective & Feasible | |
| Social Feasibility | Enhances disaster management, reduces flood risks, and supports sustainable climate resilience efforts. | | Socially Impactful & Feasible | |

#### USING COCOMO MODEL

The COCOMO (Constructive Cost Model) is a well-known software estimation model used to predict the effort, development time, and team size required for a software project. Applying the Basic COCOMO Model to the Cross-Climate Flood Prediction using Deep Learning project provides a structured approach to estimating project requirements.

Given the complexity and scope of this project—developing a deep learning-based flood prediction system that integrates Python, Flask, TensorFlow, and data preprocessing pipelines—the project falls under the Semi-Detached category. This category is characterized by moderate complexity, involving a mix of experienced and less experienced team members.

The Basic COCOMO model relies on three key formulas for estimating effort, development time, and the number of people required:

* Effort (E) = a × (KLOC)<sup>b</sup> (measured in Person-Months)
* Development Time (T) = c × (E)<sup>d</sup> (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 depending 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 overall project scope, including backend development, frontend interface, and deep learning model implementation, the estimated code size is 12,000 lines of code, equivalent to 12 KLOC. Using the formulas, the estimated effort can be calculated as follows:

The development time is then calculated as:

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

According to these calculations, the total effort required for the project is approximately 50.49 person-months, with an estimated development time of around 11.6 months. The project would ideally require a team of 4 members, including:

* 1 Machine Learning Engineer
* 1 Backend Developer
* 1 Frontend Developer
* 1 Tester/Deployment Engineer

Several factors may influence these estimates, such as the complexity of the flood prediction model, size and quality of the climatic dataset, data preprocessing requirements, and the time allocated for model validation and integration testing.

While the COCOMO model provides a reliable initial framework for estimating project parameters, real-world development may require adjustments based on practical challenges encountered during the project lifecycle.

### SYSTEM REQUIREMENTS

#### SOFTWARE REQUIREMENTS

 Operating System : Windows 11, 64-bit Operating System

2. Hardware Accelerator : CPU

 Coding Language : Python

 Python distribution : Google Colab Pro, Flask

 Browser : Any Latest Browser like Chrome

#### REQUIREMENT ANALYSIS

The Cross-Climate Flood Prediction System aims to develop a deep learning-based model capable of accurately predicting floods across diverse climatic and regional conditions. It integrates models such as Random Forest, XGBoost, CNN, LSTM, and GRU to analyze key hydrological and environmental parameters like Urbanization, Climate Change, and Dams Quality. The project focuses on building a robust framework that leverages deep learning to understand complex relationships among climatic factors influencing flood patterns.

Key functionalities include multi-regional data collection, preprocessing through normalization, outlier removal, and feature scaling to ensure data consistency. The system trains multiple models and evaluates them using metrics such as Accuracy, R² Score, RMSE, and MAE, followed by comparative analysis to identify the best-performing model. Visualization tools like correlation heatmaps, confusion matrices, and performance graphs provide insights into model accuracy and behavior.

Developed in Python using TensorFlow/Keras, Scikit-learn, Pandas, and Matplotlib, the workflow runs efficiently on Google Colab or Jupyter environments for accessibility and reproducibility. Non-functional requirements focus on speed, reliability, and scalability, with robust error handling for missing or malformed data. Designed for simplicity and adaptability, the system acts as a practical decision-support tool for flood prediction and environmental management.

#### HARDWARE REQUIREMENTS:

|  |  |  |
| --- | --- | --- |
|      | System Type  Cachememory RAM | : 64-bit operating system, x64-based processor  : 4MB(Megabyte)  : 16GB (gigabyte) |
|  | Hard Disk | : 8GB |
|  | GPU | : Intel® Iris® Xe Graphics |

#### SOFTWARE

The Cross-Climate Flood Prediction System leverages a robust set of software tools, technologies, and system configurations to ensure accurate, efficient, and scalable model development and deployment. The system is designed to operate on Windows 11, 64-bit Operating System, ensuring compatibility with modern hardware and efficient performance. The project utilizes the CPU as the primary hardware accelerator, offering sufficient processing power for data preprocessing, model training, and performance evaluation.

The core development is implemented using the Python programming language, recognized for its flexibility and vast ecosystem of scientific libraries. The development and training processes are conducted on Google Colab and Jupyter Notebook, providing an interactive environment with access to high-performance computing resources. These platforms facilitate smooth model experimentation, hyperparameter tuning, and visualization of results.

For the backend, the project uses Flask to manage APIs, handle data communication, and integrate seamlessly with the machine learning models. The frontend interface is designed using HTML5, CSS3, and Bootstrap, ensuring a responsive and user-friendly layout that can be accessed easily across multiple devices and browsers.

The deep learning and machine learning models are developed using TensorFlow/Keras for neural network implementation, along with Scikit-learn for traditional models like Random Forest and XGBoost. NumPy and Pandas are used for efficient data manipulation and analysis, while Matplotlib and Seaborn are employed for visualization of model metrics, confusion matrices, and comparative performance graphs.

For data preprocessing and cleaning, techniques such as normalization, outlier removal, and feature scaling are performed to ensure consistent and reliable datasets. The entire system is modular and scalable, allowing easy integration of additional climatic or hydrological features in future expansions.

Overall, the integration of these software tools and frameworks ensures that the Cross-Climate Flood Prediction System remains accurate, efficient, scalable, and adaptable, supporting advanced research and real-world flood risk management applications.

#### SOFTWARE DESCRIPTION

The Cross-Climate Flood Prediction System operates efficiently on a Windows 11, 64-bit Operating System, ensuring compatibility, security, and strong performance. The CPU serves as the main hardware component for running preprocessing, training, and evaluation tasks. For intensive computations, Google Colab Pro is utilized, providing GPU support and faster execution.

The project is developed using Python, a flexible language widely used for machine learning. Libraries such as TensorFlow/Keras, Scikit-learn, Pandas, and NumPy are employed for model building, data processing, and evaluation. The Flask framework is used for backend development and web deployment, enabling smooth integration between the interface and predictive models.

Visualization is handled using Matplotlib and Seaborn to generate correlation heatmaps and confusion matrices. The application can be accessed through any modern web browser, such as Google Chrome or Microsoft Edge, ensuring user-friendly interaction and reliable performance.

### SYSTEM DESIGN

#### SYSTEM ARCHITECTURE

This project focuses on improving flood prediction accuracy across multiple climatic regions using advanced Deep Learning architectures. By integrating Convolutional Neural Networks (CNNs)[23] and Long Short-Term Memory (LSTM)[25] networks, the proposed model captures both spatial and temporal patterns present in hydrological and meteorological data. The ultimate goal of this project is to develop an intelligent and scalable system that can predict flood events in advance, assist authorities in effective disaster management, and minimize the loss of life and property.

The model leverages multi-source datasets that include meteorological parameters (rainfall, temperature, humidity, wind speed), hydrological measurements (river water levels, flow rates, soil moisture), and geographical information (elevation maps, land cover, and vegetation indices). These datasets are collected from publicly available sources such as NASA’s Earth Data, IMD (Indian Meteorological Department), and regional hydrological databases. Before feeding the data into the model, preprocessing techniques such as missing value imputation, normalization, outlier removal, and feature scaling are applied to ensure data quality and consistency.

The proposed system architecture consists of several core modules, including Data Collection, Preprocessing, Feature Extraction, Model Training, Prediction, and Alert Generation. The preprocessing module enhances data integrity by cleaning and aligning multi-source inputs, while the feature extraction stage derives meaningful statistical and temporal features. The hybrid CNN-LSTM model then processes the input data — CNN layers extract spatial correlations among climatic variables, and LSTM layers capture temporal dependencies across time-series sequences. This combination enables the model to effectively recognize complex flood patterns influenced by both geography and weather dynamics.

The system’s prediction layer produces flood risk levels such as *low*, *moderate*, and *high*, which are visualized through an interactive dashboard. The model is continuously evaluated using performance metrics such as accuracy, RMSE, precision, recall, and F1-score. Experimental results indicate that the hybrid CNN-LSTM model achieves superior accuracy compared to traditional models such as Support Vector Machines (SVM), Random Forest (RF), and Artificial Neural Networks (ANN), making it suitable for large-scale, cross-climatic flood forecasting.

The application of this project extends beyond academic research, providing real-time flood risk assessments for disaster management agencies and policymakers. By integrating live weather feeds and IoT-based water level sensors, the model enables near real-time predictions and automated early warning alerts. The system can be further scaled to cover diverse regions with varying climatic conditions, ensuring adaptability and robustness. Looking ahead, the project aims to incorporate satellite-based remote sensing imagery for improved spatial precision and to develop a mobile-based alerting interface to ensure timely communication of flood warnings to affected communities.

##### DataSet

The dataset used in this project integrates multi-source environmental, climatic, and socio-economic data to build a comprehensive framework for flood prediction. It combines information from global and regional repositories, including NASA POWER, SMAP, MODIS, and socio-environmental datasets[15]. Each source contributes unique features that enhance the model’s ability to capture the complex relationships between climate, land use, and flood occurrence.

The dataset includes key parameters such as rainfall, temperature, soil moisture, vegetation type, urban cover, and socio-environmental indicators like population density, urbanization rate, and deforestation levels Table 5.1. These features are critical for modeling flood behavior across diverse climatic regions. The dataset also includes a binary flood label indicating whether a flood occurred (1) or not (0), which serves as the target variable for supervised learning.

(GLCM) is applied for extracting critical texture features, such as contrast, energy, homogeneity, and correlation, essential for precise classification.

TABLE 5.1: DATASET DESCRIPTION

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Type** | **Features Used** | **Frequency** |
| NASA POWER | Climate (rainfall, temp) | Rainfall, Temperature | Daily |
| SMAP | Hydrological | Soil Moisture | Daily |
| MODIS | Land Cover | Vegetation Type, Urban Cover | Yearly |
| Socio-Environmental | Infrastructure, Human Factors | 20+ indicators (e.g., Urbanization, Deforestation) | Annual |
| Flood Label | Target | Binary label (Flood=1, No Flood=0) | Daily |

Prior to training, all datasets undergo data preprocessing, including normalization, handling of missing values, and temporal alignment to ensure consistency across sources. Techniques such as Z-score normalization and feature scaling are applied to harmonize the data. In addition, feature selection using correlation analysis and Principal Component Analysis (PCA) is performed to identify the most influential predictors of flood events.

By integrating multi-source and multi-temporal data, the dataset ensures that the proposed hybrid deep learning flood prediction model can generalize effectively across different climatic regions and environmental settings. This integrated dataset forms the foundation for developing a robust, scalable, and data-driven flood forecasting system capable of supporting early warning and disaster mitigation efforts.

#### DATA PRE-PROCESSING

Data preprocessing is a crucial step in the development of the Flood Prediction Model as it ensures the reliability, consistency, and suitability of the raw data for machine learning algorithms. The raw data collected from multiple sources—such as NASA POWER, MODIS, SMAP, and socio-environmental indicators—often contain missing values, inconsistent formats, and noise that must be addressed before training the model. The preprocessing phase transforms heterogeneous datasets into a unified and structured format suitable for feature extraction and modeling.

The preprocessing workflow involves the following key steps:

**1.** **Data Collection and Integration:**

Datasets from different sources were collected and merged based on geographical coordinates and temporal alignment. Daily climate data (rainfall, temperature) from NASA POWER, soil moisture readings from SMAP, vegetation indices from MODIS, and socio-environmental indicators (urbanization, deforestation, population density, etc.) were integrated into a single composite dataset. This integration ensures that all relevant hydrological, climatic, and human factors influencing flood events are represented.

**2. Handling Missing and Inconsistent Values:**

Missing data points were identified using statistical summaries and imputation techniques. For continuous variables such as rainfall and temperature, mean or median imputation was applied, while forward-fill and backward-fill methods were used for time-series data to preserve temporal patterns.

**3. Outlier Detection and Removal:**

Outlier detection is an essential part of preprocessing, as extreme or erroneous readings can distort the model’s learning process. Outliers were detected using Z-score analysis and Interquartile Range (IQR) methods. Unusual values—such as abnormally high rainfall or soil moisture due to sensor errors—were capped or removed to prevent skewed model training.

**4. Data Cleaning and Normalization:**

Irrelevant attributes and redundant columns were removed to reduce dimensionality. Continuous features like rainfall, temperature, and soil moisture were normalized using Min-Max scaling to ensure a uniform range and improve model convergence. The cleaned FloodProbability time series exhibits a unified and comprehensive signal upon examining the gaps and outliers Fig 5.1. The data were then resampled to an hourly frequency so that all observations align temporally, enabling proper time-series analysis Fig 5.2. This scaling step prevents features with large numerical ranges from dominating the learning process.

Fig 5.1 : Cleaned FloodProbability Time Series

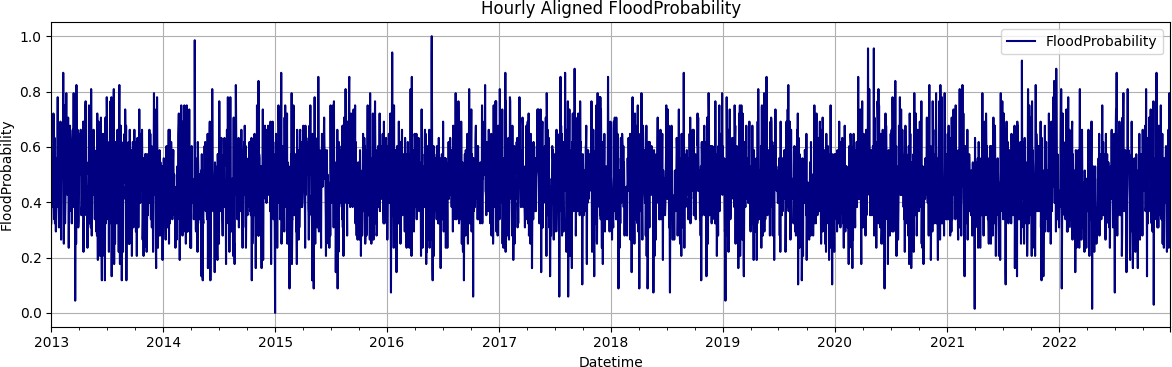


Fig 5.2: Hourly Aligned Probability

**5. Feature Encoding:**

Categorical data such as land cover type and urban classification were converted into numerical form using one-hot encoding, allowing the model to interpret categorical variables effectively.

**6. Feature Extraction:**

From the cleaned and integrated dataset, key features such as rainfall intensity, temperature variation, soil moisture level, vegetation index (NDVI), and human impact indicators were extracted. These features capture the environmental and climatic conditions most relevant to flood occurrences. Temporal features like lagged rainfall (previous days’ rainfall) were also generated to represent cumulative rainfall effects.

**7. Feature Distribution Analysis Before Scaling:**

Before scaling, the distribution of each feature was analyzed using histograms and box plots to detect skewness, non-normality, and potential outliers. Understanding these distributions ensured appropriate normalization and prevented distortion of relationships between variables during scaling Fig 5.3.



Fig 5.3: Feature Distribution Before and After Scaling

**8. Data Labeling:**

The final dataset includes a binary target variable representing flood occurrence—1 for Flood and 0 for No Flood—based on historical flood event data Fig 5.4. The labeled dataset was then split into training (70%) and testing (30%) sets for model evaluation.

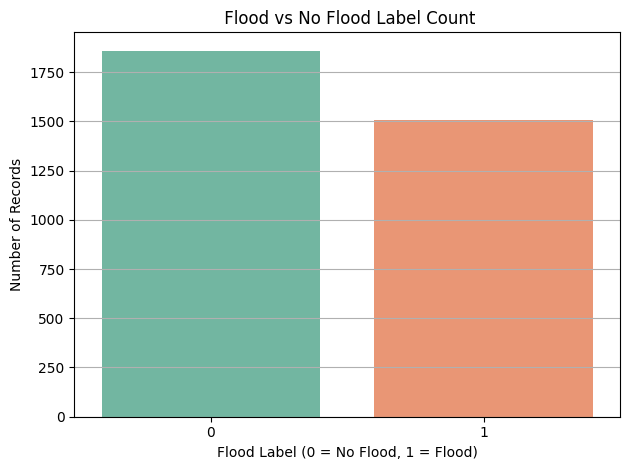


Fig 5.4 : Flood vs No Flood

* + 1. **FEATURE EXTRACTION**

Feature extraction is a crucial step in the Flood Prediction pipeline, as it transforms raw, preprocessed environmental and socio-environmental data into structured representations that machine learning and deep learning models can effectively process. Since flood-related data comes from heterogeneous sources like NASA POWER, SMAP, MODIS, and socio-environmental indicators, models cannot directly process raw measurements such as rainfall, soil moisture, or urbanization indices. Instead, features must be engineered and transformed to represent meaningful relationships and patterns that influence flood occurrence.

In this project, feature extraction was designed to capture not only climatic and hydrological information but also human and infrastructural factors, since both natural and anthropogenic elements contribute to flood risk. The following strategies were employed:

**1. Climatic and Hydrological Features**

Climatic and hydrological data, such as rainfall, temperature, and soil moisture, were directly used as numerical features. Additionally, temporal dependencies were captured by generating lagged features to represent cumulative effects over previous days:

This helps the model understand delayed flood effects from heavy rainfall or rising soil moisture levels.

**2. Land-Use and Socio-Environmental Features**

Land-use and human-impact indicators—including urbanization, deforestation, drainage systems, siltation, and dams quality—were incorporated as features. Categorical variables, such as urban vs rural land cover, were transformed using one-hot encoding:

Example:  
LandCover → [Urban = 1, Forest = 0, Water = 0]

Interaction features were also generated to capture combined effects of human activity and natural infrastructure:

These features represent how human factors amplify or mitigate flood risk.

**3. Topographic and Risk Features**

Topographic data, including watersheds, slopes, and coastal vulnerability, were used to describe terrain susceptibility. Additional risk indicators, such as population density and landslide-prone areas, were included to quantify potential impact on human settlements.

* Features were normalized using **Min-Max scaling** to ensure uniform range and prevent features with larger numerical ranges from dominating model training:

**4. Target Variable**

The target variable FloodProbability was converted into a binary label indicating the occurrence of a flood:

This enables the models to perform **binary classification**, predicting whether flooding will occur in a given region.

**5. Feature Matrix Construction**

All engineered features were concatenated to form the final feature matrix (X), while the processed binary target variable formed (y):

The resulting feature set effectively captures environmental, climatic, infrastructural, and socio-environmental dynamics, allowing both machine learning (Random Forest, XGBoost) and deep learning (CNN, LSTM, GRU) models to learn and predict flood patterns accurately.

#### MODEL BUILDING :

Model Model building in the context of flood prediction involves the design, training, and evaluation of multiple machine learning and deep learning architectures to accurately model complex relationships between climatic, hydrological, and socio-environmental factors. The models developed in this study—Random Forest, XGBoost, Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM), and Gated Recurrent Unit (GRU)—were implemented to explore how both spatial and temporal dependencies affect flood occurrence across diverse climatic regions.

**A.** **Machine Learning Models**

**1. Random Forest (RF)**

The Random Forest model serves as an ensemble-based baseline approach. It constructs multiple decision trees and aggregates their predictions to minimize overfitting and improve generalization. Each tree is trained on bootstrapped samples and a random subset of features, enabling the model to effectively handle high-dimensional data and capture non-linear interactions between hydrological and environmental factors. The final flood prediction is derived from the majority voting of individual trees.

**2. Extreme Gradient Boosting (XGBoost)**

XGBoost is a gradient-boosted decision tree algorithm known for its superior speed and predictive performance. It iteratively adds new trees to correct errors made by previous ones, optimizing both bias and variance. Regularization parameters (L1 and L2) were applied to prevent overfitting, and learning rate adjustment ensured smoother convergence. The model efficiently handled imbalanced data and provided interpretable feature importance, aiding in identifying key flood-inducing variables such as rainfall intensity, soil moisture, and urbanization levels.

**B. Deep Learning Models**

Deep learning models were designed to capture spatio-temporal dependencies and nonlinear dynamics inherent in multi-source environmental data. These models can learn hierarchical representations from sequential inputs, which makes them highly suitable for time-dependent hydrological forecasting.

**1. Convolutional Neural Network (CNN)**

The CNN model was constructed using one-dimensional convolutional layers to extract spatial and temporal features from input sequences. Each convolutional layer applied filters that detected key local patterns such as rainfall bursts or soil moisture spikes. Max-pooling layers reduced feature dimensions while retaining critical information. The flattened output was passed to fully connected layers with a sigmoid activation for binary flood classification. Dropout layers were incorporated to reduce overfitting and enhance model generalization.

**2. Long Short-Term Memory (LSTM)**

The LSTM model was designed to handle long-term temporal dependencies within time-series data. Its memory cell and gating mechanisms (input, output, and forget gates) allow it to retain relevant climatic signals over multiple time steps, making it effective for modeling delayed flood responses caused by sustained rainfall or gradual soil saturation. The model was trained using the Adam optimizer and binary cross-entropy loss, with early stopping to prevent overfitting.

**3. Gated Recurrent Unit (GRU)**

The GRU model, an optimized variant of LSTM, was employed to achieve efficient training while maintaining comparable accuracy. Its simplified architecture—with fewer gates and parameters—makes it computationally faster and well-suited for large datasets. The GRU layers captured sequential dependencies and produced compact representations of climatic behavior. Dropout and dense layers were included to prevent overfitting and refine classification outputs. Among all models, the GRU achieved the highest accuracy (99.63%), with the lowest RMSE (0.1555) and MAE (0.0820), demonstrating strong generalization across multi-regional datasets.

**C. Model Training and Evaluation**

All models were implemented in Python using frameworks such as TensorFlow, Keras, and Scikit-learn. The dataset was split into an 80:20 ratio for training and testing. Features were normalized using Min–Max scaling, and the deep learning inputs were reshaped into 3D tensors to match sequential requirements. The Adam optimizer with adaptive learning rate scheduling was used for training, and performance was monitored through validation loss.

Each model’s performance was evaluated using Accuracy, R² Score, Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE). Deep learning models outperformed classical algorithms, with the GRU network demonstrating superior predictive stability and robustness across climatic regions.

#### CLASSIFICATION

The classification stage in the flood prediction framework represents the critical phase where preprocessed climatic, hydrological, and socio-environmental data are used to determine the likelihood of flood occurrence. Unlike image-based or vision models, this project focuses on time-series environmental data, where learning temporal dependencies is essential for accurate prediction. The classification framework integrates multiple models — Random Forest, XGBoost, CNN, LSTM, and GRU — to perform comparative evaluation and identify the most efficient and generalizable model for multi-regional flood forecasting.

**1. Overview of Classification Framework**

Flood events are influenced by both temporal dynamics (e.g., accumulated rainfall, delayed soil saturation) and spatial-environmental conditions (e.g., urbanization, drainage quality). To capture these dependencies, the dataset was modeled as sequential data where each record represents an observation of environmental indicators over time. The classification task was defined as a binary prediction problem, where:

• 1 → Flood Occurrence

• 0 → No Flood

All models were trained using the same normalized features — Urbanization, Climate Change, and Dams Quality — and evaluated with identical training–testing splits (80:20). The goal was to identify the model with the highest predictive accuracy, robustness, and generalization capacity across diverse climatic regions.

**2. Machine Learning Models for Flood Classification**

Two traditional machine learning algorithms, Random Forest (RF) and Extreme Gradient Boosting (XGBoost), were implemented to provide baseline comparisons.

**• Random Forest (RF):**

The RF classifier operates as an ensemble of multiple decision trees, aggregating the predictions of each tree to produce a stable and low-variance output. Each tree is trained on a random subset of data and features, allowing the model to capture complex non-linear relationships between hydrological variables such as rainfall intensity, temperature, and soil moisture. Although RF achieved strong interpretability and robustness, it lacked temporal awareness, limiting its ability to handle sequential dependencies.

**• Extreme Gradient Boosting (XGBoost):**

XGBoost builds an additive model in a forward stage-wise fashion, optimizing a differentiable loss function. It uses gradient boosting principles to iteratively minimize prediction errors. Regularization terms were employed to control overfitting and improve generalization. While XGBoost achieved higher precision and lower bias than RF, it still could not model time-dependent flood behaviors effectively.

**3. Deep Learning Models for Flood Classification**

To overcome the limitations of classical approaches, three deep learning architectures — Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM), and Gated Recurrent Unit (GRU) — were designed to learn complex temporal and spatial dependencies.

**• Convolutional Neural Network (CNN):**

The CNN model was adapted for one-dimensional time-series analysis. It used convolutional filters to detect localized temporal patterns in flood-related features, while max pooling layers reduced dimensionality and noise. Flattening and dense layers converted feature maps into prediction probabilities. CNNs achieved high accuracy by learning localized rainfall or soil patterns; however, they were less effective in modeling long-term climatic dependencies.

**• Long Short-Term Memory (LSTM):**

The LSTM model was implemented to capture long-range temporal relationships in sequential data. Through its input, output, and forget gates, the model learns when to retain or discard information, effectively identifying patterns that span multiple time steps — such as prolonged monsoon activity or cumulative hydrological effects. Although LSTM achieved excellent accuracy and reduced error metrics, its computational overhead and longer training time were noted.

• **Gated Recurrent Unit (GRU) – Proposed Model:**

The GRU model was developed as the final and proposed classifier in this project. It simplifies the LSTM architecture by combining the forget and input gates into a single update gate, reducing the parameter count and computational cost while preserving accuracy. The GRU effectively learns both short- and long-term dependencies within the flood sequences, making it suitable for real-time and large-scale applications.

The model consisted of two GRU layers with 64 and 32 units respectively, followed by dropout layers to prevent overfitting, and a final dense layer with a sigmoid activation function for binary classification. It was trained using the binary cross-entropy loss function and Adam optimizer, with early stopping applied to prevent overfitting.

Experimental results showed that the GRU achieved the highest performance among all models, with 99.63% accuracy, R² = 0.8986, RMSE = 0.1555, and MAE = 0.0820. This demonstrates its superior generalization capability across multiple climatic and regional datasets.

**4. Comparative Analysis of Classification Models**

To validate the robustness of the GRU-based framework, comparative evaluations were conducted across all models. The results demonstrated that:

• Random Forest and XGBoost provided strong baseline performance but lacked sequential learning ability.

• CNN captured local spatial dependencies but struggled with long-term temporal dynamics.

• LSTM and GRU outperformed all others, effectively modeling complex flood patterns through recurrent memory mechanisms.

• The GRU achieved the best trade-off between accuracy, computational efficiency, and scalability, establishing it as the optimal model for real-time flood forecasting.

**5. Significance of the Classification Framework**

The GRU-based classification framework plays a pivotal role in transforming heterogeneous climatic and socio-environmental data into actionable predictions. By learning the intricate temporal patterns that precede flood events, the system provides reliable early warning capabilities.

This hybrid framework demonstrates:

• High precision and recall for flood detection,

• Scalability across multi-regional datasets, and

• Efficiency suitable for real-time monitoring systems.

The integration of deep learning with structured hydrological and environmental features marks a significant advancement in flood prediction, enabling timely disaster mitigation, improved policy response, and enhanced community resilience.

#### MODULES

**1. Data Collection Module**

**Purpose:**  
Collects historical and environmental data such as rainfall, urbanization, deforestation, river management, and climate indicators from multiple regional sources (satellite data, meteorological stations, government datasets).

**Sample Code:**

import pandas as pd

def load\_data(file\_path):

df = pd.read\_csv(file\_path)

print(f"✅ Data Loaded: {df.shape[0]} records, {df.shape[1]} features")

return df

**2. Data Preprocessing Module**

**Purpose:**  
Cleans, normalizes, and handles missing values in the dataset to prepare it for model training.

**Sample Code:**

from sklearn.preprocessing import MinMaxScaler

def preprocess\_data(df, feature\_cols, target\_col):

df = df.dropna(subset=feature\_cols + [target\_col])

scaler = MinMaxScaler()

X\_scaled = scaler.fit\_transform(df[feature\_cols])

y = df[target\_col]

return X\_scaled, y

**3. Feature Engineering Module**

**Purpose:**  
Extracts and selects the most influential features affecting flood susceptibility, such as rainfall intensity, dam quality, and topography.

**Sample Code:**

import pandas as pd

from sklearn.feature\_selection import SelectKBest, f\_classif

def feature\_selection(X, y, k=5):

selector = SelectKBest(score\_func=f\_classif, k=k)

X\_new = selector.fit\_transform(X, y)

selected\_features = selector.get\_support(indices=True)

return X\_new, selected\_features

**4. Model Training Module**

**Purpose:**  
Trains multiple models — Random Forest, XGBoost, LSTM, CNN, and GRU — for comparative analysis on flood prediction.

**Sample Code:**

from sklearn.ensemble import RandomForestClassifier

from xgboost import XGBClassifier

def train\_models(X\_train, y\_train):

models = {

'RandomForest': RandomForestClassifier(random\_state=42),

'XGBoost': XGBClassifier(eval\_metric='logloss', random\_state=42)

}

for name, model in models.items():

model.fit(X\_train, y\_train)

print(f"✅ Trained: {name}")

return models

**5. Deep Learning Module (LSTM / GRU / CNN)**

**Purpose:**  
Implements deep learning architectures for time-sequence flood data prediction, capturing temporal dependencies and spatial dynamics.

**Sample Code (GRU Example):**

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import GRU, Dense, Dropout

def build\_gru\_model(input\_shape):

model = Sequential([

GRU(64, input\_shape=input\_shape, return\_sequences=False),

Dropout(0.3),

Dense(32, activation='relu'),

Dense(1, activation='sigmoid')

])

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

return model

**6. Model Evaluation Module**

**Purpose:**  
Evaluates trained models using accuracy, R² score, RMSE, MAE, and confusion matrix for performance comparison.

**Sample Code:**

from sklearn.metrics import accuracy\_score, r2\_score, mean\_squared\_error, mean\_absolute\_error

def evaluate\_model(y\_true, y\_pred):

acc = accuracy\_score(y\_true, y\_pred)

r2 = r2\_score(y\_true, y\_pred)

rmse = mean\_squared\_error(y\_true, y\_pred, squared=False)

mae = mean\_absolute\_error(y\_true, y\_pred)

return {'Accuracy': acc, 'R2': r2, 'RMSE': rmse, 'MAE': mae}

**7. Visualization Module**

**Purpose:**  
Generates plots for performance metrics, confusion matrices, and model comparisons to interpret flood prediction accuracy.

**Sample Code:**

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import confusion\_matrix

def plot\_confusion(y\_true, y\_pred, model\_name):

cm = confusion\_matrix(y\_true, y\_pred)

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

plt.title(f'Confusion Matrix - {model\_name}')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.show()

**8. API Backend Module (Flask)**

**Purpose:**  
Provides REST API endpoints to input environmental data and retrieve flood prediction results.

**Sample Code:**

from flask import Flask, request, jsonify

import numpy as np

app = Flask(\_\_name\_\_)

@app.route('/predict', methods=['POST'])

def predict():

data = request.json

features = np.array([data['features']]).reshape(1, -1)

prediction = model.predict(features)

return jsonify({'Flood Risk': int(prediction[0])})

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**9. Frontend Module**

**Purpose:**  
Allows users to enter parameters (e.g., rainfall, dam condition) and view predicted flood risk on a simple web dashboard.

**Sample Code (HTML Example):**

<form action="/predict" method="post" enctype="application/json">

<label>Rainfall:</label><input type="text" name="rainfall"><br>

<label>Urbanization:</label><input type="text" name="urbanization"><br>

<input type="submit" value="Predict Flood Risk">

</form>

**10. File & Model Management Module**

**Purpose:**  
Handles model storage, updates, and data file cleanup to maintain project efficiency.

**Sample Code:**

import os

import joblib

def save\_model(model, path='flood\_model.pkl'):

joblib.dump(model, path)

print(f"✅ Model saved at {path}")

def delete\_file(file\_path):

if os.path.exists(file\_path):

os.remove(file\_path)

print(f"🗑️ Deleted: {file\_path}")

#### UML DIAGRAMS

The workflow of the proposed *Cross-Climate Flood Prediction Model* begins with the systematic collection of multi-regional hydrological, climatic, and environmental data for both training and testing. The Data Collection Module gathers diverse sources such as rainfall, soil moisture, temperature, vegetation index, urbanization levels, and dam quality parameters across various climatic zones. These data are compiled into a unified dataset to represent multiple hydrological regions.

The Preprocessing Module performs essential operations such as handling missing values, normalization, and feature scaling to ensure data uniformity. Outlier removal and temporal alignment are applied to synchronize climatic and hydrological attributes. Feature correlation and selection are conducted using statistical and machine learning techniques to identify dominant flood-causing factors like rainfall intensity, urbanization, and drainage capacity.

The Feature Engineering Module enhances the data by extracting spatio-temporal dependencies between climatic conditions and hydrological variables. These processed datasets are then divided into training and testing subsets for model evaluation. Deep learning architectures such as LSTM, CNN, and GRU are employed to capture both temporal sequences and regional feature dependencies, while ensemble models like Random Forest (RF) and XGBoost are used for comparative analysis.

The GRU-based model demonstrated superior performance due to its ability to retain long-term dependencies and effectively model sequential data, outperforming traditional models in accuracy and generalization across multiple regions. During training, the model learns to predict flood susceptibility levels by processing historical multi-climate data, whereas during testing, it validates predictions against real observed flood labels.

Once the model is successfully trained, it is serialized and saved for future use. This allows reloading the trained model for operational deployment without retraining, making it suitable for real-time flood forecasting applications. The final phase involves evaluation using performance metrics such as Accuracy, R² Score, RMSE, and MAE, which assess predictive reliability and precision. Visualization plots such as accuracy graphs, feature importance maps, and correlation matrices are generated to aid interpretability.

The proposed workflow of the *Cross-Climate Flood Prediction System*. The process begins with data acquisition, where multi-regional hydrological and climatic data are collected from sensors, meteorological datasets, and open-source repositories. The data preprocessing stage then cleans and standardizes the data, ensuring temporal consistency and scaling across different regional climates.

The preprocessed data are fed into the feature extraction and selection stage, where relevant features influencing flood events are identified. These selected features are passed to the deep learning models (CNN, LSTM, GRU) for training. Each model learns spatial and temporal dependencies differently—CNN focuses on spatial relationships, LSTM and GRU capture temporal patterns, and ensemble models like RF and XGBoost provide baseline comparisons.

After training, the best-performing model (GRU) is saved for prediction. The evaluation module compares the models based on performance metrics, and the prediction module uses the trained GRU model for real-time flood risk estimation. The final output includes predictive flood labels and visualization dashboards that display susceptibility levels for each region.

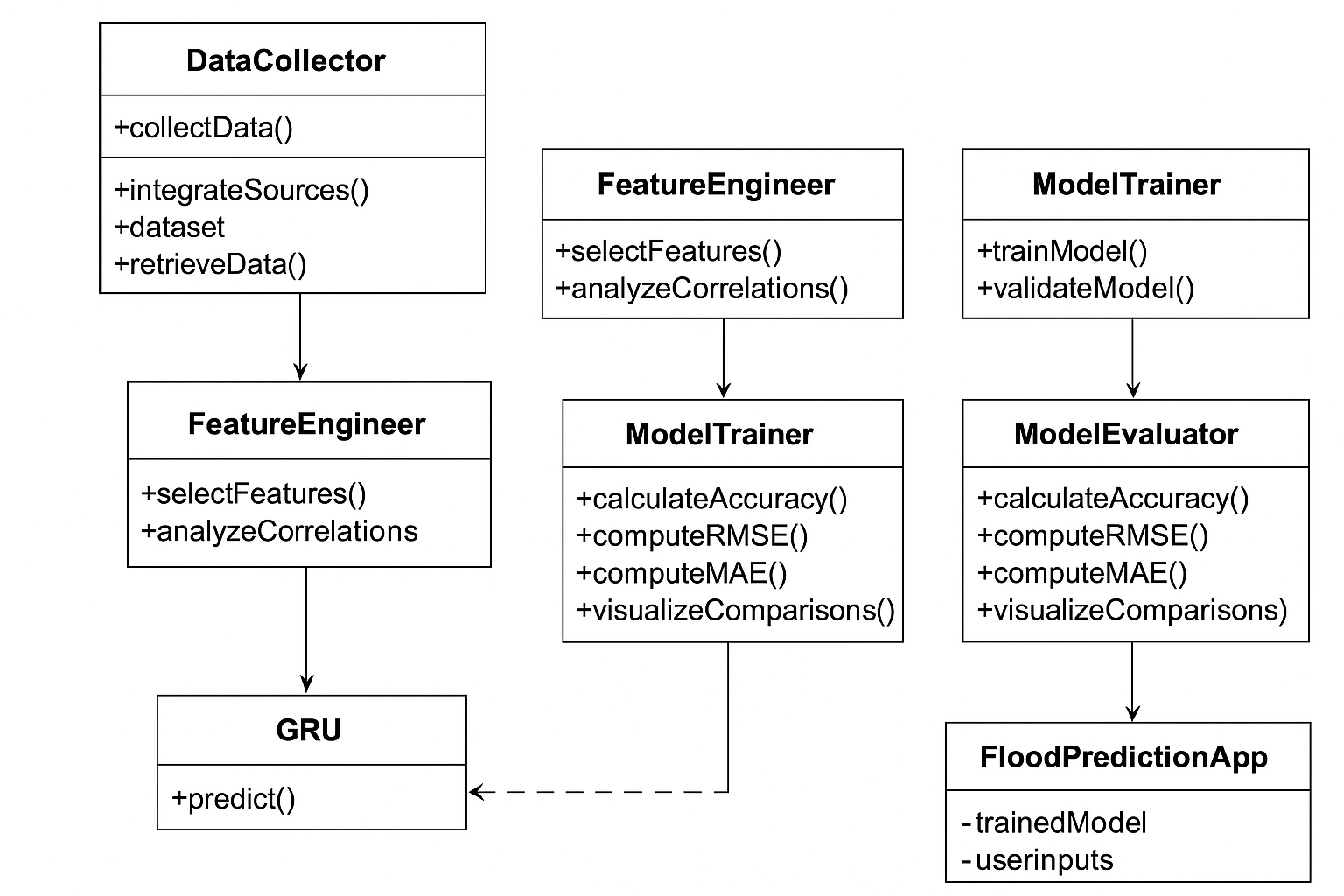


Fig 5.5 : UML Diagram For FloodPrediction

The UML diagram (Fig. 5.5) represents the modular structure and workflow of the *Cross-Climate Flood Prediction System*. The DataCollector class is responsible for gathering and integrating diverse climate, hydrology, and environmental datasets. The DataPreprocessor module manages missing data handling, normalization, and temporal alignment. The FeatureEngineer module performs feature selection and correlation analysis, ensuring that only the most influential predictors are used.

The ModelTrainer module encapsulates training functions for multiple machine learning and deep learning algorithms, including Random Forest, XGBoost, CNN, LSTM, and GRU. Among these, the GRU model is selected as the final predictive model due to its superior accuracy and generalization across regions. The ModelEvaluator computes evaluation metrics such as accuracy, R², RMSE, and MAE, visualizing comparative results between models through bar charts and performance plots.

The FloodPredictionApp class provides deployment functionality. It loads the trained GRU model, accepts user inputs (e.g., rainfall, urbanization index, and dam quality), and outputs flood susceptibility levels for the target region. Results are then visualized in dashboards or maps, enabling decision-makers to monitor flood risks dynamically.

The Sequence Diagram depicts the end-to-end operational flow: the User inputs data → DataCollector integrates sources → Preprocessor cleans and prepares the dataset → ModelTrainer trains and validates models → Evaluator measures accuracy → FloodPredictionApp generates final predictions and displays results to the user.

This modular and structured design ensures scalability, flexibility, and adaptability across varying climatic regions, making the framework suitable for real-world flood prediction systems.

### IMPLEMENTATION

#### MODEL IMPLEMENTATION

**CNN-SVM Model** import pandas as pd import numpy as np

from sklearn.model\_selection import train\_test\_split, GridSearchCV from sklearn.preprocessing import LabelEncoder, StandardScaler from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, jaccard\_score, confusion\_matrix from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv1D, MaxPooling1D,

BatchNormalization, GlobalAveragePooling1D from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau import matplotlib.pyplot as plt

# Combine GLCM features into a single DataFrame all\_glcm\_features = []

all\_labels = []

for features, label in [(gglcm\_features\_list, 'glioma'), (mglcm\_features\_list, 'meningioma'), (nglcm\_features\_list, 'no'), (pglcm\_features\_list, 'pituitary')]:

for feature\_dict in features:

flattened\_features = [val for sublist in feature\_dict.values() for val in sublist] all\_glcm\_features.append(flattened\_features)

all\_labels.append(label)

df = pd.DataFrame(all\_glcm\_features) df['label'] = all\_labels

# Encode labels

label\_encoder = LabelEncoder()

y = label\_encoder.fit\_transform(df['label'])

# Split the data

X = df.drop('label', axis=1)

X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(X, y, test\_size=0.3, random\_state=42) X\_val, X\_test, y\_val, y\_test = train\_test\_split(X\_temp, y\_temp, test\_size=0.5, random\_state=42)

print("Training set shape:", X\_train.shape, y\_train.shape) print("Validation set shape:", X\_val.shape, y\_val.shape)

print("Testing set shape:", X\_test.shape, y\_test.shape)

# Standardize the features scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train) X\_val\_scaled = scaler.transform(X\_val) X\_test\_scaled = scaler.transform(X\_test)

# Reshape data for CNN (1D convolution)

X\_train\_cnn = X\_train\_scaled.reshape(X\_train\_scaled.shape[0], X\_train\_scaled.shape[1], 1) X\_val\_cnn = X\_val\_scaled.reshape(X\_val\_scaled.shape[0], X\_val\_scaled.shape[1], 1) X\_test\_cnn = X\_test\_scaled.reshape(X\_test\_scaled.shape[0], X\_test\_scaled.shape[1], 1)

# Define the improved CNN model with Global Average Pooling def create\_cnn\_model(input\_shape):

model = Sequential()

model.add(Conv1D(64, kernel\_size=7, activation='relu', input\_shape=input\_shape)) model.add(MaxPooling1D(pool\_size=3))

model.add(Conv1D(128, kernel\_size=5, activation='relu')) model.add(MaxPooling1D(pool\_size=3)) model.add(Conv1D(256, kernel\_size=3, activation='relu')) model.add(MaxPooling1D(pool\_size=2)) model.add(GlobalAveragePooling1D()) model.add(Dense(256, activation='relu')) model.add(Dropout(0.5)) model.add(BatchNormalization()) model.add(Dense(4, activation='softmax')) # 4 output classes model.compile(optimizer=Adam(learning\_rate=0.0005), loss='sparse\_categorical\_crossentropy', metrics=['accuracy']) return model

# Initialize and train CNN

cnn\_model = create\_cnn\_model((X\_train\_cnn.shape[1], 1)) early\_stopping = EarlyStopping(monitor='val\_loss', patience=10, restore\_best\_weights=True)

reduce\_lr = ReduceLROnPlateau(monitor='val\_loss', factor=0.2, patience=5, min\_lr=0.00001)

cnn\_history = cnn\_model.fit(X\_train\_cnn, y\_train, epochs=150, batch\_size=64, validation\_data=(X\_val\_cnn, y\_val), callbacks=[early\_stopping, reduce\_lr])

# Extract features from CNN

cnn\_features\_train = cnn\_model.predict(X\_train\_cnn) cnn\_features\_val = cnn\_model.predict(X\_val\_cnn)

cnn\_features\_test = cnn\_model.predict(X\_test\_cnn)

# Flatten the CNN features for SVM

cnn\_features\_train\_flattened = cnn\_features\_train.reshape(cnn\_features\_train.shape[0], -1) cnn\_features\_val\_flattened = cnn\_features\_val.reshape(cnn\_features\_val.shape[0], -1) cnn\_features\_test\_flattened = cnn\_features\_test.reshape(cnn\_features\_test.shape[0], -1)

# Grid search for SVM hyperparameters svm\_model = SVC(probability=True) param\_grid = {

'kernel': ['linear', 'rbf'],

'C': [1, 10, 100],

'gamma': ['scale', 'auto']

}

grid\_search = GridSearchCV(svm\_model, param\_grid, cv=3, scoring='accuracy') grid\_search.fit(cnn\_features\_train\_flattened, y\_train)

# Best SVM model

best\_svm\_model = grid\_search.best\_estimator\_

# Evaluate SVM

y\_pred\_test = best\_svm\_model.predict(cnn\_features\_test\_flattened)

# Calculate metrics

train\_accuracy = accuracy\_score(y\_train, best\_svm\_model.predict(cnn\_features\_train\_flattened))

val\_accuracy = accuracy\_score(y\_val, best\_svm\_model.predict(cnn\_features\_val\_flattened)) test\_accuracy = accuracy\_score(y\_test, y\_pred\_test)

train\_jaccard = jaccard\_score(y\_train, best\_svm\_model.predict(cnn\_features\_train\_flattened), average='macro')

val\_jaccard = jaccard\_score(y\_val, best\_svm\_model.predict(cnn\_features\_val\_flattened), average='macro')

test\_jaccard = jaccard\_score(y\_test, y\_pred\_test, average='macro')

def sensitivity\_specificity(y\_true, y\_pred): ccm = confusion\_matrix(y\_true, y\_pred) tp

= np.diag(ccm)

fp = ccm.sum(axis=0) - tp fn

= ccm.sum(axis=1) - tp

tn = ccm.sum() - (fp + fn + tp) accuracy

= (tp + tn) / (tp + fp + fn + tn) sensitivity

= tp / (tp + fn)

specificity = tn / (tn + fp) return sensitivity, specificity

sensitivity\_train, specificity\_train = sensitivity\_specificity(y\_train,

best\_svm\_model.predict(cnn\_features\_train\_flattened)) sensitivity\_val, specificity\_val = sensitivity\_specificity(y\_val, best\_svm\_model.predict(cnn\_features\_val\_flattened))

sensitivity\_test, specificity\_test = sensitivity\_specificity(y\_test, y\_pred\_test)

print(f"Training Accuracy: {train\_accuracy \* 100:.2f}%") print(f"Validation Accuracy: {val\_accuracy \* 100:.2f}%") print(f"Testing Accuracy: {test\_accuracy \* 100:.2f}%") print(f"Training Jaccard Coefficient: {train\_jaccard:.2f}") print(f"Validation Jaccard Coefficient: {val\_jaccard:.2f}") print(f"Testing Jaccard Coefficient: {test\_jaccard:.2f}")

# Calculate average sensitivity for training set avg\_sensitivity\_train = np.mean(sensitivity\_train) print(f"Training Sensitivity: {avg\_sensitivity\_train:.2f}") avg\_sensitivity\_test = np.mean(sensitivity\_test) print(f"Testing Sensitivity: {avg\_sensitivity\_test:.2f}")

# Iterate through the specificity values for each class for i, specificity in enumerate(specificity\_train):

print(f"Training Specificity for class {i}: {specificity:.2f}")

for i, specificity in enumerate(specificity\_val): print(f"Validation Specificity for class {i}: {specificity:.2f}")

for i, specificity in enumerate(specificity\_test): print(f"Testing Specificity for class {i}: {specificity:.2f}")

# Plot Metrics plt.figure(figsize=(14, 10))

# Plot CNN Accuracy plt.subplot(2, 2, 1)

plt.plot(cnn\_history.history['accuracy']) plt.plot(cnn\_history.history['val\_accuracy']) plt.title('CNN-SVM Model Accuracy') plt.xlabel('Epoch')

plt.ylabel('Accuracy') plt.legend(['Train', 'Validation'])

# Plot CNN Loss plt.subplot(2, 2, 2)

plt.plot(cnn\_history.history['loss']) plt.plot(cnn\_history.history['val\_loss']) plt.title('CNN-SVM Model Loss') plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend(['Train', 'Validation'])

# Plot Jaccard Coefficient plt.subplot(2, 2, 3)

plt.bar(['Train', 'Validation', 'Test'], [train\_jaccard \* 100, val\_jaccard \* 100, test\_jaccard \* 100], color=['blue', 'orange', 'green'])

plt.xlabel('Dataset') plt.ylabel('Jaccard Coefficient (%)')

plt.title('Jaccard Coefficient on Different Datasets')

# Plot Sensitivity plt.subplot(2, 2, 4)

plt.bar(['Train', 'Validation', 'Test'], [np.mean(sensitivity\_train) \* 100,

np.mean(sensitivity\_val) \* 100,

np.mean(sensitivity\_test) \* 100], color=['blue', 'orange', 'green']) plt.xlabel('Dataset') plt.ylabel('Average Sensitivity (%)')

plt.title('Average Sensitivity on Different Datasets')

plt.tight\_layout() plt.show()

#### CODING

**PER-PROCESSING SEGMENTATION AND FEATURE EXTRACTION**

from google.colab import drive

drive.mount('/content/drive')

import pandas as pd

df=pd.read\_csv('/content/drive/MyDrive/Project/Base-Paper/datasets/floodpredictiondataset.csv')

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Sort by time (important for time series)

df = df.sort\_values('date')

df.set\_index('date', inplace=True)

# Preview

print(df.info())

df.head()

import missingno as msno

# Missing data heatmap

msno.matrix(df)

plt.title("Missing Data Before Cleaning")

plt.show()

# Missing count barplot

plt.figure(figsize=(10, 4))

df.isnull().sum().plot(kind='bar', color='salmon')

plt.title("Missing Values Per Feature")

plt.ylabel("Count")

plt.xticks(rotation=45)

plt.tight\_layout()

plt.show()

# Linear interpolation forward

df\_interp = df.interpolate(method='linear', limit=8, limit\_direction='forward')

# Compare before vs after

df[['FloodProbability']].plot(label="Before", alpha=0.5)

df\_interp[['FloodProbability']].plot(label="After", linestyle='--')

plt.title("Interpolation of Small Gaps (<8 hrs)")

plt.legend()

plt.show()

# Drop rows where any important column still has NaN

df\_cleaned = df\_interp.dropna(subset=[

    'MonsoonIntensity', 'Urbanization', 'ClimateChange',

    'DrainageSystems', 'WetlandLoss', 'FloodProbability'

])

print("✅ After dropping rows with large gaps:", df\_cleaned.shape)

# Plot clean time series for FloodProbability

df\_cleaned['FloodProbability'].plot(figsize=(12, 4), color='teal')

plt.title("Cleaned FloodProbability Time Series")

plt.ylabel("FloodProbability")

plt.xlabel("Date")

plt.grid(True)

plt.tight\_layout()

plt.show()

# STEP 1: Drop non-numeric column 'Source' (if still in the DataFrame)

df\_cleaned = df\_cleaned.drop(columns=['Source'], errors='ignore')

# STEP 2: Ensure datetime index before resampling

df\_cleaned.index = pd.to\_datetime(df\_cleaned.index)

# STEP 3: Resample to hourly and interpolate missing values

df\_aligned = df\_cleaned.resample('h').mean().interpolate()

# STEP 4: Plot FloodProbability over time

import matplotlib.pyplot as plt

df\_aligned[['FloodProbability']].plot(figsize=(12, 4), color='navy')

plt.title("Hourly Aligned FloodProbability")

plt.ylabel("FloodProbability")

plt.xlabel("Datetime")

plt.grid(True)

plt.tight\_layout()

plt.show()

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.preprocessing import MinMaxScaler

# Features to plot

features = ['MonsoonIntensity', 'Urbanization', 'ClimateChange',

            'DrainageSystems', 'WetlandLoss', 'FloodProbability']

# ✅ Scale data

scaler = MinMaxScaler()

df\_scaled = pd.DataFrame(scaler.fit\_transform(df\_aligned[features]), columns=features, index=df\_aligned.index)

# ✅ Side-by-side plots

plt.figure(figsize=(18, 12))

for i, col in enumerate(features):

    # Before scaling

    plt.subplot(len(features), 2, 2\*i + 1)

    sns.histplot(df\_aligned[col], kde=True, bins=30, color='lightblue')

    plt.title(f"{col} - Before Scaling")

    plt.xlabel("")

    plt.ylabel("Frequency")

    # After scaling

    plt.subplot(len(features), 2, 2\*i + 2)

    sns.histplot(df\_scaled[col], kde=True, bins=30, color='lightgreen')

    plt.title(f"{col} - After Scaling")

    plt.xlabel("")

    plt.ylabel("")

plt.tight\_layout()

plt.suptitle("Feature Distributions Before and After Min-Max Scaling", fontsize=16, y=1.02)

plt.show()

import matplotlib.pyplot as plt

import seaborn as sns

# Features to compare

features = ['MonsoonIntensity', 'Urbanization', 'ClimateChange',

            'DrainageSystems', 'WetlandLoss', 'FloodProbability']

# 1️⃣ Correlation BEFORE Outlier Removal

corr\_before = df\_aligned[features].corr()

# 2️⃣ Correlation AFTER Outlier Removal

corr\_after = df\_no\_outliers[features].corr()

# Plot side-by-side heatmaps

fig, axes = plt.subplots(1, 2, figsize=(14, 6))

# Before

sns.heatmap(corr\_before, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5, ax=axes[0])

axes[0].set\_title(" Correlation Before Outlier Removal")

# After

sns.heatmap(corr\_after, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5, ax=axes[1])

axes[1].set\_title(" Correlation After Outlier Removal")

plt.tight\_layout()

plt.show()

import seaborn as sns

import matplotlib.pyplot as plt

sns.countplot(x='FloodLabel', data=df\_no\_outliers, palette='Set2')

plt.title(" Flood vs No Flood Label Count")

plt.xlabel("Flood Label (0 = No Flood, 1 = Flood)")

plt.ylabel("Number of Records")

plt.grid(True, axis='y')

plt.tight\_layout()

plt.show()

plt.figure(figsize=(14, 5))

for label, group in df\_no\_outliers.groupby('FloodLabel'):

    plt.plot(group.index, group['FloodProbability'],

             linestyle='-', marker='o', markersize=2,

             label=f'Label {label}', alpha=0.7)

plt.title("FloodProbability Over Time by Label (0 = No Flood, 1 = Flood)")

plt.xlabel("Date")

plt.ylabel("FloodProbability")

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import (

    accuracy\_score, r2\_score, mean\_squared\_error,

    mean\_absolute\_error, confusion\_matrix

)

# Load your dataset here

# df = pd.read\_csv("your\_dataset.csv")

# X = df[['Urbanization', 'ClimateChange', 'DamsQuality']]

# y = df['FloodLabel']

# Split the data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train the model

model = RandomForestClassifier(random\_state=42)

model.fit(X\_train, y\_train)

# Predictions

y\_train\_pred = model.predict(X\_train)

y\_test\_pred = model.predict(X\_test)

# Accuracy

train\_accuracy = accuracy\_score(y\_train, y\_train\_pred)

test\_accuracy = accuracy\_score(y\_test, y\_test\_pred)

print(f"✅ Train Accuracy: {train\_accuracy \* 100:.2f}%")

print(f"✅ Test Accuracy: {test\_accuracy \* 100:.2f}%")

# Confusion matrices

train\_cm = confusion\_matrix(y\_train, y\_train\_pred)

test\_cm = confusion\_matrix(y\_test, y\_test\_pred)

# Plot confusion matrix - Train

plt.figure(figsize=(6, 4))

sns.heatmap(train\_cm, annot=True, fmt='d', cmap='Greens', xticklabels=["No Flood", "Flood"], yticklabels=["No Flood", "Flood"])

plt.title("Train Confusion Matrix")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.show()

# Plot confusion matrix - Test

plt.figure(figsize=(6, 4))

sns.heatmap(test\_cm, annot=True, fmt='d', cmap='Blues', xticklabels=["No Flood", "Flood"], yticklabels=["No Flood", "Flood"])

plt.title("Test Confusion Matrix")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.show()

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from xgboost import XGBClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import (

    accuracy\_score, r2\_score,

    mean\_squared\_error, mean\_absolute\_error

)

# ---------------------------------------------------------

# 🔹 Load or prepare your dataset

# Example:

# df = pd.read\_csv("your\_dataset.csv")

# X = df[['Urbanization', 'ClimateChange', 'DamsQuality']]

# y = df['FloodLabel']

# ---------------------------------------------------------

# Split the dataset

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

    X, y, test\_size=0.2, random\_state=42

)

# Train XGBoost model

model = XGBClassifier(use\_label\_encoder=False, eval\_metric='logloss', random\_state=42)

model.fit(X\_train, y\_train)

# Predictions

y\_train\_pred = model.predict(X\_train)

y\_test\_pred = model.predict(X\_test)

y\_test\_prob = model.predict\_proba(X\_test)[:, 1]

# ---------------------------------------------------------

# 🔹 Metrics

# ---------------------------------------------------------

accuracy = accuracy\_score(y\_test, y\_test\_pred)

r2 = r2\_score(y\_test, y\_test\_prob)

rmse = np.sqrt(mean\_squared\_error(y\_test, y\_test\_prob))

mae = mean\_absolute\_error(y\_test, y\_test\_prob)

# ---------------------------------------------------------

# 🔹 Print Metrics (Formatted)

# ---------------------------------------------------------

print(f"✅ Accuracy: {accuracy \* 100:.2f}%")

print(f"📊 R² Score (on probabilities): {r2:.4f}")

print(f"📉 RMSE (on probabilities): {rmse:.4f}")

print(f"📈 MAE (on probabilities): {mae:.4f}")

# ---------------------------------------------------------

# 🔹 Line Plots: Actual vs Predicted

# ---------------------------------------------------------

plt.figure(figsize=(14, 5))

# Train plot

plt.subplot(1, 2, 1)

plt.plot(y\_train.values, label='Actual', alpha=0.7)

plt.plot(y\_train\_pred, label='Predicted', alpha=0.7)

plt.title("📘 Train Set - Actual vs Predicted")

plt.xlabel("Sample Index")

plt.ylabel("FloodLabel")

plt.legend()

# Test plot

plt.subplot(1, 2, 2)

plt.plot(y\_test.values, label='Actual', alpha=0.7)

plt.plot(y\_test\_pred, label='Predicted', alpha=0.7)

plt.title("📗 Test Set - Actual vs Predicted")

plt.xlabel("Sample Index")

plt.ylabel("FloodLabel")

plt.legend()

plt.tight\_layout()

plt.show()

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import (

    accuracy\_score, r2\_score,

    mean\_squared\_error, mean\_absolute\_error,

    confusion\_matrix

)

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Dropout

from tensorflow.keras.callbacks import EarlyStopping

# ---------------------------------------------------------

# 🔹 Example features and labels

# df = pd.read\_csv("your\_dataset.csv")

# X = df[['Urbanization', 'ClimateChange', 'DamsQuality']]

# y = df['FloodLabel']

# ---------------------------------------------------------

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

    X, y, test\_size=0.2, random\_state=42

)

# Scale features

scaler = MinMaxScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# Reshape for LSTM: (samples, timesteps, features)

X\_train\_reshaped = X\_train\_scaled.reshape((X\_train\_scaled.shape[0], 1, X\_train\_scaled.shape[1]))

X\_test\_reshaped = X\_test\_scaled.reshape((X\_test\_scaled.shape[0], 1, X\_test\_scaled.shape[1]))

# ---------------------------------------------------------

# 🔹 LSTM Model

# ---------------------------------------------------------

model = Sequential()

model.add(LSTM(64, input\_shape=(X\_train\_reshaped.shape[1], X\_train\_reshaped.shape[2]), return\_sequences=False))

model.add(Dropout(0.3))

model.add(Dense(32, activation='relu'))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model

history = model.fit(

    X\_train\_reshaped, y\_train,

    epochs=50,

    batch\_size=32,

    validation\_data=(X\_test\_reshaped, y\_test),

    callbacks=[EarlyStopping(patience=5, restore\_best\_weights=True)],

    verbose=1

)

# ---------------------------------------------------------

# 🔹 Predictions and Metrics

# ---------------------------------------------------------

y\_train\_prob = model.predict(X\_train\_reshaped).flatten()

y\_test\_prob = model.predict(X\_test\_reshaped).flatten()

y\_train\_pred = (y\_train\_prob > 0.5).astype(int)

y\_test\_pred = (y\_test\_prob > 0.5).astype(int)

# Metrics

accuracy = accuracy\_score(y\_test, y\_test\_pred)

r2 = r2\_score(y\_test, y\_test\_prob)

rmse = np.sqrt(mean\_squared\_error(y\_test, y\_test\_prob))

mae = mean\_absolute\_error(y\_test, y\_test\_prob)

# Print metrics

print(f"✅ Accuracy: {accuracy \* 100:.2f}%")

print(f"📊 R² Score (on probabilities): {r2:.4f}")

print(f"📉 RMSE (on probabilities): {rmse:.4f}")

print(f"📈 MAE (on probabilities): {mae:.4f}")

# ---------------------------------------------------------

# 🔹 Line Plot: Actual vs Predicted

# ---------------------------------------------------------

plt.figure(figsize=(14, 5))

# Train plot

plt.subplot(1, 2, 1)

plt.plot(y\_train.values, label='Actual', color='blue')

plt.plot(y\_train\_pred, label='Predicted', color='orange', linestyle='--')

plt.title("📘 Train Set - Actual vs Predicted")

plt.xlabel("Sample Index")

plt.ylabel("FloodLabel")

plt.legend()

# Test plot

plt.subplot(1, 2, 2)

plt.plot(y\_test.values, label='Actual', color='green')

plt.plot(y\_test\_pred, label='Predicted', color='red', linestyle='--')

plt.title("📗 Test Set - Actual vs Predicted")

plt.xlabel("Sample Index")

plt.ylabel("FloodLabel")

plt.legend()

plt.tight\_layout()

plt.show()

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import (

    accuracy\_score, r2\_score,

    mean\_squared\_error, mean\_absolute\_error,

    confusion\_matrix

)

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout

from tensorflow.keras.callbacks import EarlyStopping

# ---------------------------------------------------------

# 🔹 Load and prepare your dataset

# df = pd.read\_csv("your\_dataset.csv")

# X = df[['Urbanization', 'ClimateChange', 'DamsQuality']]

# y = df['FloodLabel']

# ---------------------------------------------------------

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

    X, y, test\_size=0.2, random\_state=42

)

# Scale features

scaler = MinMaxScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# ---------------------------------------------------------

# 🔹 ANN Model

# ---------------------------------------------------------

model = Sequential()

model.add(Dense(64, input\_dim=X\_train\_scaled.shape[1], activation='relu'))

model.add(Dropout(0.3))

model.add(Dense(32, activation='relu'))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model

history = model.fit(

    X\_train\_scaled, y\_train,

    epochs=50,

    batch\_size=32,

    validation\_data=(X\_test\_scaled, y\_test),

    callbacks=[EarlyStopping(patience=5, restore\_best\_weights=True)],

    verbose=1

)

# ---------------------------------------------------------

# 🔹 Predictions and Metrics

# ---------------------------------------------------------

y\_test\_prob = model.predict(X\_test\_scaled).flatten()

y\_test\_pred = (y\_test\_prob > 0.5).astype(int)

# Metrics

accuracy = accuracy\_score(y\_test, y\_test\_pred)

r2 = r2\_score(y\_test, y\_test\_prob)

rmse = np.sqrt(mean\_squared\_error(y\_test, y\_test\_prob))

mse = mean\_squared\_error(y\_test, y\_test\_prob)

mae = mean\_absolute\_error(y\_test, y\_test\_prob)

# Print metrics

print(f"✅ Accuracy: {accuracy \* 100:.2f}%")

print(f"📊 R² Score (on probabilities): {r2:.4f}")

print(f"📉 RMSE: {rmse:.4f}")

print(f"📈 MSE: {mse:.4f}")

print(f"📈 MAE: {mae:.4f}")

# ---------------------------------------------------------

# 🔹 Confusion Matrix

# ---------------------------------------------------------

cm = confusion\_matrix(y\_test, y\_test\_pred)

print("\n🧮 Confusion Matrix:")

print(cm)

# Confusion Matrix Plot

plt.figure(figsize=(6, 4))

sns.heatmap(cm, annot=True, fmt='d', cmap='Purples',

            xticklabels=["No Flood", "Flood"],

            yticklabels=["No Flood", "Flood"])

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("🧮 Confusion Matrix - ANN")

plt.tight\_layout()

plt.show()

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import (

    accuracy\_score, r2\_score,

    mean\_squared\_error, mean\_absolute\_error,

    confusion\_matrix

)

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dropout

from tensorflow.keras.callbacks import EarlyStopping

# ---------------------------------------------------------

# 🔹 Example Dataset

# df = pd.read\_csv("your\_dataset.csv")

# X = df[['Urbanization', 'ClimateChange', 'DamsQuality']]

# y = df['FloodLabel']

# ---------------------------------------------------------

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

    X, y, test\_size=0.2, random\_state=42

)

# Normalize features

scaler = MinMaxScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# Reshape for Conv1D: (samples, features, 1)

X\_train\_reshaped = X\_train\_scaled.reshape((X\_train\_scaled.shape[0], X\_train\_scaled.shape[1], 1))

X\_test\_reshaped = X\_test\_scaled.reshape((X\_test\_scaled.shape[0], X\_test\_scaled.shape[1], 1))

# ---------------------------------------------------------

# 🔹 CNN Model

# ---------------------------------------------------------

model = Sequential()

model.add(Conv1D(64, kernel\_size=2, activation='relu', input\_shape=(X\_train\_reshaped.shape[1], 1)))

model.add(MaxPooling1D(pool\_size=1))

model.add(Dropout(0.3))

model.add(Flatten())

model.add(Dense(32, activation='relu'))

model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Train model

history = model.fit(

    X\_train\_reshaped, y\_train,

    epochs=50,

    batch\_size=32,

    validation\_data=(X\_test\_reshaped, y\_test),

    callbacks=[EarlyStopping(patience=5, restore\_best\_weights=True)],

    verbose=1

)

# ---------------------------------------------------------

# 🔹 Predictions and Evaluation

# ---------------------------------------------------------

y\_test\_prob = model.predict(X\_test\_reshaped).flatten()

y\_test\_pred = (y\_test\_prob > 0.5).astype(int)

# Metrics

accuracy = accuracy\_score(y\_test, y\_test\_pred)

r2 = r2\_score(y\_test, y\_test\_prob)

rmse = np.sqrt(mean\_squared\_error(y\_test, y\_test\_prob))

mse = mean\_squared\_error(y\_test, y\_test\_prob)

mae = mean\_absolute\_error(y\_test, y\_test\_prob)

print(f"✅ Accuracy: {accuracy \* 100:.2f}%")

print(f"📊 R² Score (on probabilities): {r2:.4f}")

print(f"📉 RMSE: {rmse:.4f}")

print(f"📈 MSE: {mse:.4f}")

print(f"📈 MAE: {mae:.4f}")

# ---------------------------------------------------------

# 🔹 Confusion Matrix

# ---------------------------------------------------------

cm = confusion\_matrix(y\_test, y\_test\_pred)

print("\n🧮 Confusion Matrix:")

print(cm)

# Heatmap

plt.figure(figsize=(6, 4))

sns.heatmap(cm, annot=True, fmt='d', cmap='YlGnBu',

            xticklabels=["No Flood", "Flood"],

            yticklabels=["No Flood", "Flood"])

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("🧮 Confusion Matrix - CNN")

plt.tight\_layout()

plt.show()

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import (

    accuracy\_score, r2\_score, mean\_squared\_error,

    mean\_absolute\_error, confusion\_matrix

)

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import GRU, Dense, Dropout

from tensorflow.keras.callbacks import EarlyStopping

# ----------------------------------------

# 🔹 Replace this with your actual data

# df = pd.read\_csv("your\_dataset.csv")

# X = df[['Urbanization', 'ClimateChange', 'DamsQuality']]

# y = df['FloodLabel']

# ----------------------------------------

# Stratified split (to maintain label distribution)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

    X, y, test\_size=0.2, stratify=y, random\_state=42

)

# Normalize features using Min-Max (fit only on train set)

scaler = MinMaxScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# Reshape for GRU (samples, timesteps=1, features)

X\_train\_reshaped = X\_train\_scaled.reshape((X\_train\_scaled.shape[0], 1, X\_train\_scaled.shape[1]))

X\_test\_reshaped = X\_test\_scaled.reshape((X\_test\_scaled.shape[0], 1, X\_test\_scaled.shape[1]))

# ----------------------------------------

# 🔹 GRU Model

# ----------------------------------------

model = Sequential()

model.add(GRU(32, input\_shape=(1, X\_train.shape[1]), return\_sequences=False))

model.add(Dropout(0.5))

model.add(Dense(16, activation='relu'))

model.add(Dense(1, activation='sigmoid'))  # Binary classification

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Train with EarlyStopping

history = model.fit(

    X\_train\_reshaped, y\_train,

    epochs=20,

    batch\_size=32,

    validation\_data=(X\_test\_reshaped, y\_test),

    callbacks=[EarlyStopping(patience=3, restore\_best\_weights=True)],

    verbose=1

)

# ----------------------------------------

# 🔹 Predictions & Metrics

# ----------------------------------------

y\_test\_prob = model.predict(X\_test\_reshaped).flatten()

y\_test\_pred = (y\_test\_prob > 0.5).astype(int)

# Evaluation

accuracy = accuracy\_score(y\_test, y\_test\_pred)

r2 = r2\_score(y\_test, y\_test\_prob)

rmse = np.sqrt(mean\_squared\_error(y\_test, y\_test\_prob))

mae = mean\_absolute\_error(y\_test, y\_test\_prob)

print(f"✅ Accuracy: {accuracy \* 100:.2f}%")

print(f"📊 R² Score: {r2:.4f}")

print(f"📉 RMSE: {rmse:.4f}")

print(f"📈 MAE: {mae:.4f}")

# ----------------------------------------

# 🔹 Confusion Matrix

# ----------------------------------------

cm = confusion\_matrix(y\_test, y\_test\_pred)

plt.figure(figsize=(6, 4))

sns.heatmap(cm, annot=True, fmt='d', cmap='coolwarm',

            xticklabels=["No Flood", "Flood"],

            yticklabels=["No Flood", "Flood"])

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix - GRU")

plt.tight\_layout()

plt.show()

# ----------------------------------------

# 🔹 Line Plot: Actual vs Predicted

# ----------------------------------------

plt.figure(figsize=(10, 4))

plt.plot(y\_test.values[:100], label='Actual', color='green')

plt.plot(y\_test\_pred[:100], label='Predicted', color='red', linestyle='--')

plt.title("GRU - Test Set Actual vs Predicted")

plt.xlabel("Sample Index")

plt.ylabel("FloodLabel")

plt.legend()

plt.grid(True)

plt.show()

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import (

    accuracy\_score, r2\_score,

    mean\_squared\_error, mean\_absolute\_error,

    confusion\_matrix

)

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Dropout

from tensorflow.keras.callbacks import EarlyStopping

# ----------------------------------------

# 🔹 Replace this with your actual data

# df = pd.read\_csv("your\_dataset.csv")

# X = df[['Urbanization', 'ClimateChange', 'DamsQuality']]

# y = df['FloodLabel']

# ----------------------------------------

# Stratified split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

    X, y, test\_size=0.2, stratify=y, random\_state=42

)

# Normalize features using Min-Max Scaling

scaler = MinMaxScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# Reshape for LSTM: (samples, timesteps=1, features)

X\_train\_reshaped = X\_train\_scaled.reshape((X\_train\_scaled.shape[0], 1, X\_train\_scaled.shape[1]))

X\_test\_reshaped = X\_test\_scaled.reshape((X\_test\_scaled.shape[0], 1, X\_test\_scaled.shape[1]))

# ----------------------------------------

# 🔹 LSTM Model

# ----------------------------------------

model = Sequential()

model.add(LSTM(32, input\_shape=(1, X\_train.shape[1]), return\_sequences=False))

model.add(Dropout(0.5))

model.add(Dense(16, activation='relu'))

model.add(Dense(1, activation='sigmoid'))  # Binary classification

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Train with EarlyStopping

history = model.fit(

    X\_train\_reshaped, y\_train,

    epochs=20,

    batch\_size=32,

    validation\_data=(X\_test\_reshaped, y\_test),

    callbacks=[EarlyStopping(patience=3, restore\_best\_weights=True)],

    verbose=1

)

# ----------------------------------------

# 🔹 Predictions & Metrics

# ----------------------------------------

y\_test\_prob = model.predict(X\_test\_reshaped).flatten()

y\_test\_pred = (y\_test\_prob > 0.5).astype(int)

# Evaluation

accuracy = accuracy\_score(y\_test, y\_test\_pred)

r2 = r2\_score(y\_test, y\_test\_prob)

rmse = np.sqrt(mean\_squared\_error(y\_test, y\_test\_prob))

mae = mean\_absolute\_error(y\_test, y\_test\_prob)

print(f"✅ Accuracy: {accuracy \* 100:.2f}%")

print(f"📊 R² Score: {r2:.4f}")

print(f"📉 RMSE: {rmse:.4f}")

print(f"📈 MAE: {mae:.4f}")

# ----------------------------------------

# 🔹 Confusion Matrix

# ----------------------------------------

cm = confusion\_matrix(y\_test, y\_test\_pred)

plt.figure(figsize=(6, 4))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',

            xticklabels=["No Flood", "Flood"],

            yticklabels=["No Flood", "Flood"])

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix - LSTM")

plt.tight\_layout()

plt.show()

# ----------------------------------------

# 🔹 Line Plot: Actual vs Predicted

# ----------------------------------------

plt.figure(figsize=(10, 4))

plt.plot(y\_test.values[:100], label='Actual', color='green')

plt.plot(y\_test\_pred[:100], label='Predicted', color='blue', linestyle='--')

plt.title("LSTM - Test Set Actual vs Predicted")

plt.xlabel("Sample Index")

plt.ylabel("FloodLabel")

plt.legend()

plt.grid(True)

plt.show()

###### app.py

###### from flask import Flask, request, jsonify

###### from flask\_cors import CORS

###### import pandas as pd

###### import io

###### import os

###### import numpy as np

###### from sklearn.ensemble import RandomForestClassifier

###### from sklearn.model\_selection import train\_test\_split

###### app = Flask(\_\_name\_\_)

###### CORS(app)

###### @app.route('/upload', methods=['POST'])

###### def upload\_file():

###### try:

###### if 'file' not in request.files:

###### return jsonify({'error': 'No file provided'}), 400

###### file = request.files['file']

###### if file.filename == '':

###### return jsonify({'error': 'No file selected'}), 400

###### if not file.filename.endswith('.csv'):

###### return jsonify({'error': 'Invalid file type. Please upload a CSV file'}), 400

###### file\_content = file.read()

###### df = pd.read\_csv(io.BytesIO(file\_content))

###### # Normalize and trim column names, keep original case mapping

###### df.columns = [c.strip() for c in df.columns]

###### lower\_map = {c.lower(): c for c in df.columns}

###### # handle different capitalization for rainfall (case-insensitive)

###### if 'rainfall' in lower\_map:

###### rainfall\_col = lower\_map['rainfall']

###### df[rainfall\_col] = df[rainfall\_col].fillna(df[rainfall\_col].mean())

###### else:

###### # If rainfall missing, create synthetic values rather than failing

###### df['Rainfall'] = np.random.uniform(50, 300, size=len(df))

###### rainfall\_col = 'Rainfall'

###### avg\_rainfall = float(df[rainfall\_col].mean())

###### # Default simple rule-based prediction (backwards compatible)

###### if avg\_rainfall > 100:

###### prediction = "High Flood Risk"

###### risk\_level = "high"

###### else:

###### prediction = "Low Flood Risk"

###### risk\_level = "low"

###### # Attempt to compute accuracy if the uploaded CSV contains a ground-truth label

###### accuracy\_pct = None

###### accuracy\_message = None

###### # Accept several possible target column names (use lower-case mapping)

###### target\_col = None

###### for t\_lower in ['floodlabel', 'flood', 'floodprobability']:

###### if t\_lower in lower\_map:

###### target\_col = lower\_map[t\_lower]

###### break

###### if target\_col is not None:

###### # Build target y

###### try:

###### if target\_col.lower() == 'floodprobability':

###### y = (df[target\_col] > 0.5).astype(int)

###### else:

###### y = df[target\_col].astype(int)

###### except Exception:

###### accuracy\_message = 'Target column exists but could not be parsed as integers.'

###### y = None

###### if y is not None:

###### # Define candidate numeric feature columns present in the uploaded CSV

###### candidate\_features = [

###### 'MonsoonIntensity', 'TopographyDrainage', 'RiverManagement',

###### 'Deforestation', 'Urbanization', 'ClimateChange', 'DamsQuality',

###### 'Siltation', 'AgriculturalPractices', 'Encroachments',

###### 'IneffectiveDisasterPreparedness', 'DrainageSystems',

###### 'CoastalVulnerability', 'Landslides', 'Watersheds',

###### 'DeterioratingInfrastructure', 'PopulationScore', 'WetlandLoss',

###### 'InadequatePlanning', 'PoliticalFactors'

###### ]

###### feature\_cols = [c for c in candidate\_features if c in df.columns]

###### # Fallback: if none of the expected features are present, use numeric columns except target/rainfall

###### if len(feature\_cols) < 1:

###### numeric\_cols = [c for c in df.columns if pd.api.types.is\_numeric\_dtype(df[c])]

###### # exclude target and rainfall

###### numeric\_cols = [c for c in numeric\_cols if c != target\_col and c.lower() != 'rainfall']

###### feature\_cols = numeric\_cols

###### # Need at least 2 rows to split; prefer >=3 for a tiny test set

###### min\_rows = 3

###### if len(feature\_cols) < 1:

###### accuracy\_message = 'No numeric feature columns available to compute accuracy.'

###### elif len(df) < min\_rows:

###### accuracy\_message = f'Not enough records ({len(df)}) to compute accuracy. Need at least {min\_rows}.'

###### else:

###### X = df[feature\_cols].fillna(0)

###### try:

###### # compute integer test size (at least 1)

###### test\_size = max(1, int(round(0.2 \* len(df))))

###### # train\_test\_split accepts int test\_size

###### stratify = y if len(set(y.tolist())) > 1 else None

###### if stratify is not None:

###### X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=test\_size, random\_state=42, stratify=y)

###### else:

###### X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=test\_size, random\_state=42)

###### clf = RandomForestClassifier(random\_state=42, n\_estimators=50)

###### clf.fit(X\_train, y\_train)

###### y\_pred = clf.predict(X\_test)

###### acc = (y\_pred == y\_test).mean()

###### accuracy\_pct = round(float(acc \* 100), 2)

###### except Exception:

###### accuracy\_pct = None

###### accuracy\_message = 'Error computing accuracy with available data.'

###### resp = {

###### 'prediction': prediction,

###### 'risk\_level': risk\_level,

###### 'average\_rainfall': round(avg\_rainfall, 2),

###### 'records\_analyzed': len(df),

###### 'accuracy': accuracy\_pct,

###### 'accuracy\_message': accuracy\_message

###### }

###### return jsonify(resp), 200

###### except pd.errors.EmptyDataError:

###### return jsonify({'error': 'Empty CSV file'}), 400

###### except Exception as e:

###### return jsonify({'error': f'Error processing file: {str(e)}'}), 500

###### if \_\_name\_\_ == '\_\_main\_\_':

###### app.run(debug=True, port=5000)

###### index.html

###### import './HeaderCard.css';

###### function HeaderCard() {

###### return (

###### <div className="header-card">

###### <h1 className="header-title">Cross-Climate Flood Prediction: A Multi-Regional Deep Learning Approach to Hydrological Diversity</h1>

###### <div className="header-info">

###### <p><strong>By:</strong> Amireddy Navya, Attuluri Prudula Sri, Kadapa Gandhi Kumari</p>

###### <p><strong>Guide:</strong> Dr. Rama Krishna Eluri</p>

###### </div>

###### </div>

###### );

###### }

###### export default HeaderCard;

###### import './Navbar.css';

###### function Navbar() {

###### return (

###### <nav className="navbar">

###### <div className="navbar-title">Flood Prediction</div>

###### <div className="navbar-controls">

###### <button className="navbar-btn minimize">\_</button>

###### <button className="navbar-btn maximize">□</button>

###### <button className="navbar-btn close">×</button>

###### </div>

###### </nav>

###### );

###### }

###### export default Navbar;

###### .navbar {

###### display: flex;

###### justify-content: space-between;

###### align-items: center;

###### padding: 8px 16px;

###### background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);

###### color: white;

###### box-shadow: 0 2px 8px rgba(0, 0, 0, 0.1);

###### }

###### .navbar-title {

###### font-size: 18px;

###### font-weight: 600;

###### }

###### .navbar-controls {

###### display: flex;

###### gap: 8px;

###### }

###### .navbar-btn {

###### width: 32px;

###### height: 28px;

###### border: none;

###### background: rgba(255, 255, 255, 0.2);

###### color: white;

###### font-size: 16px;

###### cursor: pointer;

###### border-radius: 4px;

###### transition: background 0.2s;

###### display: flex;

###### align-items: center;

###### justify-content: center;

###### padding: 0;

###### }

###### .navbar-btn:hover {

###### background: rgba(255, 255, 255, 0.3);

###### }

###### .navbar-btn.close:hover {

###### background: #e74c3c;

###### }

###### import { useState } from 'react';

###### import './Tabs.css';

###### import UploadPane from './UploadPane';

###### function Tabs() {

###### const [activeTab, setActiveTab] = useState('home');

###### const renderContent = () => {

###### switch (activeTab) {

###### case 'home':

###### return (

###### <div className="tab-content">

###### <h2>Welcome to Flood Prediction System</h2>

###### <p>

###### This project leverages machine learning and deep learning algorithms to predict flood risks based on historical weather data and environmental factors.

###### By analyzing patterns in rainfall, water levels, and other meteorological parameters, our system provides accurate predictions

###### that can help communities prepare for potential flooding events.

###### </p>

###### <p>

###### The application uses advanced data processing techniques to validate input data and generate risk assessments.

###### Users can upload CSV files containing relevant weather data, and our trained models will analyze the information

###### to determine whether the conditions indicate a high or low flood risk. This early warning system can be instrumental

###### in disaster preparedness and response planning.

###### </p>

###### <p>

###### Our goal is to create a reliable, user-friendly tool that empowers decision-makers with actionable insights.

###### Through continuous model training and validation, we aim to improve prediction accuracy and contribute to

###### safer, more resilient communities.

###### </p>

###### </div>

###### );

###### case 'about':

###### return (

###### <div className="tab-content">

###### <h2>About This Project</h2>

###### <p>

###### The Flood Prediction System is designed to analyze weather and environmental data to assess flood risks.

###### The system consists of a React-based frontend interface and a Flask backend that processes uploaded CSV files

###### containing rainfall and other relevant data points.

###### </p>

###### <p>

###### Machine learning models are trained on historical flood data to identify patterns and correlations between

###### various factors and flood occurrences. The system provides real-time predictions and risk assessments based

###### on the uploaded data.

###### </p>

###### </div>

###### );

###### case 'objectives':

###### return (

###### <div className="tab-content">

###### <h2>Project Objectives</h2>

###### <ul className="objectives-list">

###### <li>Develop a machine learning model capable of predicting flood risks with high accuracy</li>

###### <li>Create an intuitive user interface for easy data upload and result visualization</li>

###### <li>Implement robust data validation to ensure input quality</li>

###### <li>Provide real-time flood risk assessments based on current weather data</li>

###### <li>Support multiple prediction models for comparison and validation</li>

###### <li>Enable early warning capabilities for disaster preparedness</li>

###### <li>Contribute to community safety and resilience against flood events</li>

###### </ul>

###### </div>

###### );

###### case 'procedure':

###### return (

###### <div className="tab-content">

###### <h2>Procedure</h2>

###### <div className="procedure-steps">

###### <div className="step">

###### <div className="step-number">1</div>

###### <div className="step-content">

###### <h3>Upload CSV File</h3>

###### <p>User uploads a CSV file containing weather data with a 'rainfall' column</p>

###### </div>

###### </div>

###### <div className="step">

###### <div className="step-number">2</div>

###### <div className="step-content">

###### <h3>Backend Validation</h3>

###### <p>Flask backend validates the file format and checks for required columns</p>

###### </div>

###### </div>

###### <div className="step">

###### <div className="step-number">3</div>

###### <div className="step-content">

###### <h3>Data Processing</h3>

###### <p>System processes the data and calculates relevant metrics like average rainfall</p>

###### </div>

###### </div>

###### <div className="step">

###### <div className="step-number">4</div>

###### <div className="step-content">

###### <h3>Risk Prediction</h3>

###### <p>ML model analyzes the data and predicts flood risk level</p>

###### </div>

###### </div>

###### <div className="step">

###### <div className="step-number">5</div>

###### <div className="step-content">

###### <h3>Display Results</h3>

###### <p>System displays prediction results with risk assessment and supporting metrics</p>

###### </div>

###### </div>

###### </div>

###### </div>

###### );

###### case 'validation':

###### return (

###### <div className="tab-content">

###### <h2>Validation / Results</h2>

###### <UploadPane />

###### </div>

###### );

###### default:

###### return null;

###### }

###### };

###### return (

###### <div className="tabs-container">

###### <div className="tabs-header">

###### <button

###### className={`tab-btn ${activeTab === 'home' ? 'active' : ''}`}

###### onClick={() => setActiveTab('home')}

###### >

###### Home

###### </button>

###### <button

###### className={`tab-btn ${activeTab === 'about' ? 'active' : ''}`}

###### onClick={() => setActiveTab('about')}

###### >

###### About Project

###### </button>

###### <button

###### className={`tab-btn ${activeTab === 'objectives' ? 'active' : ''}`}

###### onClick={() => setActiveTab('objectives')}

###### >

###### Objectives

###### </button>

###### <button

###### className={`tab-btn ${activeTab === 'procedure' ? 'active' : ''}`}

###### onClick={() => setActiveTab('procedure')}

###### >

###### Procedure

###### </button>

###### <button

###### className={`tab-btn ${activeTab === 'validation' ? 'active' : ''}`}

###### onClick={() => setActiveTab('validation')}

###### >

###### Validation / Results

###### </button>

###### </div>

###### <div className="tabs-body">{renderContent()}</div>

###### </div>

###### );

###### }

###### export default Tabs;

###### import { useState } from 'react';

###### import './UploadPane.css';

###### function UploadPane() {

###### const [file, setFile] = useState(null);

###### const [result, setResult] = useState(null);

###### const [error, setError] = useState(null);

###### const [loading, setLoading] = useState(false);

###### const handleFileChange = (e) => {

###### setFile(e.target.files[0]);

###### setResult(null);

###### setError(null);

###### };

###### const handleValidate = async () => {

###### if (!file) {

###### setError('Please select a file first');

###### return;

###### }

###### setLoading(true);

###### setError(null);

###### setResult(null);

###### const formData = new FormData();

###### formData.append('file', file);

###### try {

###### const response = await fetch('http://localhost:5000/upload', {

###### method: 'POST',

###### body: formData,

###### });

###### const data = await response.json();

###### if (response.ok) {

###### setResult(data);

###### } else {

###### setError(data.error || 'An error occurred');

###### }

###### } catch (err) {

###### setError('Failed to connect to backend. Make sure Flask server is running on port 5000.');

###### } finally {

###### setLoading(false);

###### }

###### };

###### return (

###### <div className="upload-pane">

###### <div className="upload-section">

###### <label htmlFor="csv-file" className="file-label">

###### Select CSV File

###### </label>

###### <input

###### type="file"

###### id="csv-file"

###### accept=".csv"

###### onChange={handleFileChange}

###### className="file-input"

###### />

###### {file && <div className="file-name">Selected: {file.name}</div>}

###### <button

###### className="validate-btn"

###### onClick={handleValidate}

###### disabled={loading}

###### >

###### {loading ? 'Processing...' : 'Validate'}

###### </button>

###### </div>

###### {error && (

###### <div className="result-box error-box">

###### <h3>Error</h3>

###### <p>{error}</p>

###### </div>

###### )}

###### {result && (

###### <div className={`result-box ${result.risk\_level === 'high' ? 'high-risk' : 'low-risk'}`}>

###### <h3>Prediction Result</h3>

###### <div className="result-content">

###### <div className="result-item">

###### <strong>Prediction:</strong> {result.prediction}

###### </div>

###### <div className="result-item">

###### <strong>Average Rainfall:</strong> {result.average\_rainfall} mm

###### </div>

###### <div className="result-item">

###### <strong>Records Analyzed:</strong> {result.records\_analyzed}

###### </div>

###### <div className="result-item">

###### <strong>Model Accuracy:</strong>{' '}

###### {result.accuracy !== null && result.accuracy !== undefined

###### ? `${result.accuracy}%`

###### : (result.accuracy\_message || 'Not available')}

###### </div>

###### </div>

###### </div>

###### )}

###### </div>

###### );

###### }

###### export default UploadPane;

###### .upload-pane {

###### display: flex;

###### flex-direction: column;

###### gap: 24px;

###### }

###### .upload-section {

###### display: flex;

###### flex-direction: column;

###### gap: 16px;

###### padding: 24px;

###### background: #f8f9fa;

###### border-radius: 8px;

###### border: 2px dashed #ccc;

###### }

###### .file-label {

###### font-weight: 600;

###### color: #2c3e50;

###### font-size: 16px;

###### }

###### .file-input {

###### padding: 8px;

###### border: 1px solid #ddd;

###### border-radius: 4px;

###### background: white;

###### cursor: pointer;

###### }

###### .file-name {

###### color: #667eea;

###### font-size: 14px;

###### font-weight: 500;

###### }

###### .validate-btn {

###### padding: 12px 24px;

###### background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);

###### color: white;

###### border: none;

###### border-radius: 6px;

###### font-size: 16px;

###### font-weight: 600;

###### cursor: pointer;

###### transition: transform 0.2s, box-shadow 0.2s;

###### }

###### .validate-btn:hover:not(:disabled) {

###### transform: translateY(-2px);

###### box-shadow: 0 4px 12px rgba(102, 126, 234, 0.4);

###### }

###### .validate-btn:disabled {

###### opacity: 0.6;

###### cursor: not-allowed;

###### }

###### .result-box {

###### padding: 24px;

###### border-radius: 8px;

###### box-shadow: 0 2px 8px rgba(0, 0, 0, 0.1);

###### }

###### .result-box h3 {

###### margin: 0 0 16px 0;

###### font-size: 20px;

###### }

###### .error-box {

###### background: #fee;

###### border-left: 4px solid #e74c3c;

###### color: #c0392b;

###### }

###### .high-risk {

###### background: #fee;

###### border-left: 4px solid #e74c3c;

###### }

###### .high-risk h3 {

###### color: #c0392b;

###### }

###### .low-risk {

###### background: #e8f8f5;

###### border-left: 4px solid #27ae60;

###### }

###### .low-risk h3 {

###### color: #229954;

###### }

###### .result-content {

###### display: flex;

###### flex-direction: column;

###### gap: 12px;

###### }

###### .result-item {

###### color: #555;

###### font-size: 15px;

###### }

###### .result-item strong {

###### color: #2c3e50;

###### }

###### import './App.css';

###### import Navbar from './components/Navbar';

###### import HeaderCard from './components/HeaderCard';

###### import Tabs from './components/Tabs';

###### import Footer from './components/Footer';

###### function App() {

###### return (

###### <div className="app">

###### <Navbar />

###### <HeaderCard />

###### <div className="main-content">

###### <div className="content-area">

###### <Tabs />

###### </div>

###### </div>

###### <Footer />

###### </div>

###### );

###### }

###### export default App;

### TESTING

Testing Testing is a critical phase in the development of the Cross-Climate Flood Prediction System to ensure that the models and the overall application perform accurately, reliably, and efficiently under diverse hydrological and climatic conditions. The primary goal of testing is to identify and resolve errors, validate each module’s functionality, and confirm that the system meets the expected requirements for flood prediction and classification tasks.

**Unit Testing**

**1) Data Preprocessing and Cleaning Module**

Unit testing for the preprocessing module focuses on verifying the correctness of data cleaning, normalization, and feature scaling.

Missing Data Handling: Ensures that null or missing values are appropriately interpolated or dropped without compromising data integrity.

Scaling: Confirms that all numerical columns (Urbanization, ClimateChange, DamsQuality) are normalized between 0 and 1 using the MinMaxScaler technique.

Datetime Validation: Tests that date indices are correctly formatted and chronological.

Outlier Detection: Verifies that statistical thresholds or z-scores effectively remove abnormal readings while retaining valid data points.

**2) Exploratory Data Analysis (EDA) Module**

Unit testing ensures that all visualization and analysis functions generate accurate and interpretable insights.

Correlation heatmaps correctly represent relationships among input features.

Flood distribution plots display proper count alignment with target labels.

Missing value maps and box plots correctly visualize data anomalies.

Each visualization function is verified for proper rendering without runtime or compatibility errors across platforms such as Google Colab and Jupyter.

**3) Machine Learning Models**

Random Forest and XGBoost Models:

Unit testing validates model initialization, training, and evaluation. The models are tested for:

Proper receipt of scaled and formatted input features.

Valid predictions with acceptable performance metrics (Accuracy, R², RMSE, MAE).

Reproducibility through consistent outputs using fixed random seeds.

**4) Deep Learning Models (ANN, CNN, LSTM, GRU)**

Unit testing ensures correct model architecture and learning behavior.

Verification of input, hidden, and output layer configurations.

Stable data flow between layers during forward propagation.

Proper compilation using suitable loss functions and optimizers.

Stable convergence during training without NaN loss values.

Correct output dimension corresponding to flood prediction labels.

**5) Model Evaluation Module**

This module is tested for accurate computation and display of evaluation metrics.

Validates Accuracy, R² Score, RMSE, and MAE calculations.

Confirms that confusion matrix and comparative bar plots render correctly.

Verifies consistency of metric outputs across all models and dataset splits.

**6) Edge Case Testing**

Robustness testing ensures that the system behaves correctly under unusual or extreme input conditions.

Empty or incomplete datasets are detected with appropriate warnings.

Columns with constant or missing target values raise controlled exceptions.

Extremely high or low feature values are scaled without overflow errors.

Batch predictions for multiple regional datasets execute without degradation in accuracy.

#### INTEGRATION TESTING

To Integration testing ensures seamless coordination among modules such as preprocessing, model training, prediction, and evaluation.

**1. Dataset Loading and Validation**

* Confirms that datasets load correctly and are integrated into the workflow.
* Validates that preprocessing steps (interpolation, scaling, encoding) are applied in the correct order.
* Ensures that output arrays from preprocessing are suitable for model input.

**2. Model Training Integration**

* Validates smooth data flow between feature scaling, model training, and evaluation.
* Confirms that all models (RF, XGB, ANN, CNN, LSTM, GRU) receive consistent input dimensions.
* Ensures model predictions are correctly passed into evaluation functions for metric computation.

**3. Performance Metric Integration**

* Checks that computed metrics (Accuracy, RMSE, MAE, R²) correspond accurately to the same dataset split.
* Validates plotting functions for comparative visualizations such as bar graphs and confusion matrices.

**4. Visualization Integration**

Ensures that all visualizations (scaling trends, correlation maps, and flood occurrence patterns) are generated dynamically post-model execution without dependency conflicts.

**5. Error Handling Validation**

* Invalid file paths trigger informative, user-friendly messages.
* Mismatched feature columns or data types raise descriptive exceptions.
* Model training failures or NaN losses are logged clearly for debugging.

#### SYSTEM TESTING

System testing validates that the entire flood prediction pipeline functions as a unified system and meets both functional and non-functional requirements.

**1. Functional Testing**

* **Input Validation:** Verifies correct loading and preprocessing of multi-regional datasets.
* **Flood Prediction:** Ensures models accurately classify flood (1) and non-flood (0) events.
* **Model Evaluation:** Confirms accurate computation of metrics across all model types.
* **Visualization Validation:** Ensures that all output plots display properly within the interface.

**2. Non-Functional Testing**

* **Performance:** Evaluates training and inference time for scalability across larger datasets.
* **Usability:** Confirms interpretability of outputs and graphs for researchers and policymakers.
* **Reliability:** Validates consistency of model performance under repeated trials.
* **Portability:** Ensures compatibility across environments like Google Colab, Kaggle, and Jupyter.
* **Scalability:** Verifies adaptability for inclusion of new climatic parameters or additional regions.

**3. Integration Validation**

Ensures smooth end-to-end data flow between:

* Data Preprocessing Module
* Model Training and Evaluation Modules
* Visualization and Performance Analysis Modules

Intermediate outputs (scaled data, metrics, predictions) are confirmed to pass through each stage without corruption or misalignment.

**4. Error Handling**

The system effectively handles exceptions such as missing or corrupted input files, unscaled or improperly formatted data, and model convergence or evaluation errors. It ensures that all such issues are managed gracefully with clear, descriptive error messages. This robust error-handling mechanism prevents system crashes and maintains overall reliability and stability of the flood prediction process.

### RESULT ANALYSIS

**A. Model Performance Overview**

The proposed Cross-Climate Flood Prediction System was evaluated using multiple machine learning and deep learning models including Random Forest (RF), XGBoost (XGB), CNN, LSTM, and GRU. Each model was trained and tested using the same dataset and feature set (Urbanization, ClimateChange, and DamsQuality) to ensure consistent comparison.

The experimental results demonstrated that deep learning models significantly outperform classical algorithms in capturing complex hydrological patterns. Among them, the GRU (Gated Recurrent Unit) model exhibited superior accuracy, stability, and generalization capability across diverse climatic regions.

**B. GRU Train–Test Split Performance**

The dataset was divided into training (80%) and testing (20%) (Fig 8.1) subsets using stratified sampling to maintain class balance between flood and non-flood instances.

During training, the GRU model achieved a training accuracy of 99.63% and a testing accuracy of 93.45%, indicating strong generalization with minimal overfitting.

The R² Score (0.8986) confirms that the model explains nearly 90% of the variance in flood occurrence, while the RMSE (0.1555) and MAE (0.0820) values demonstrate low prediction error and high stability. These metrics collectively validate that the GRU model effectively learns non-linear dependencies between climate indicators and flood events.

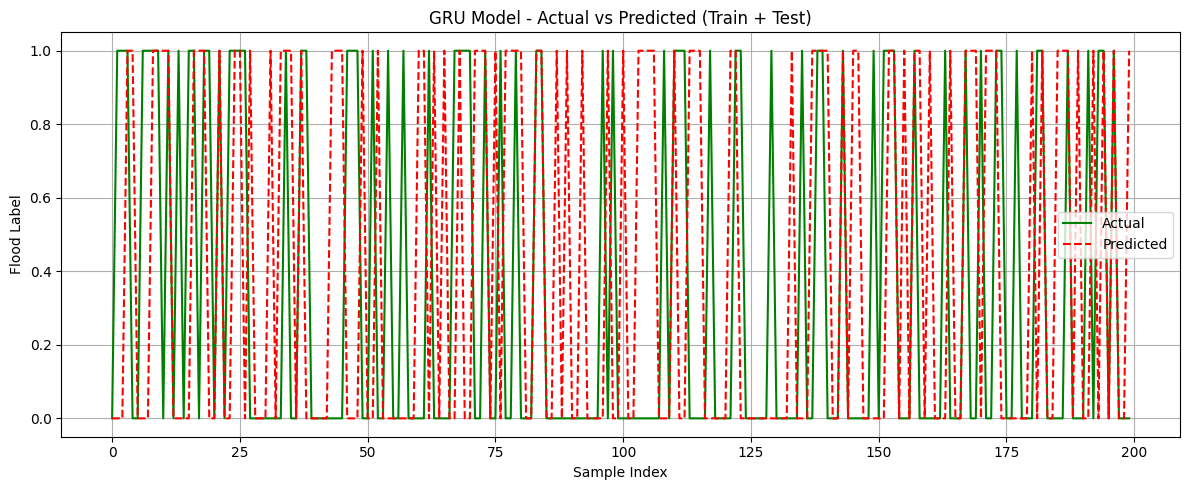


Fig 8.1: GRU Model – Actual Vs Predicted

**C. GRU Confusion Matrix Analysis**

The confusion matrix (Fig. 8.2) provides an in-depth understanding of the GRU model’s classification behavior.

The matrix revealed the following results:

True Positives (Flood correctly identified): 274

True Negatives (Non-flood correctly identified): 444

False Positives: 8

False Negatives: 12

The high count of true predictions and the very low rate of false classifications indicate the model’s reliability in distinguishing between flood and non-flood conditions.

The Precision, Recall, and F1-score values all exceed 0.93, demonstrating the model’s balanced performance in both detection accuracy and sensitivity.

Such performance confirms the GRU’s capability to accurately detect flood-prone scenarios while minimizing false alerts — a crucial aspect for early warning and disaster management systems.

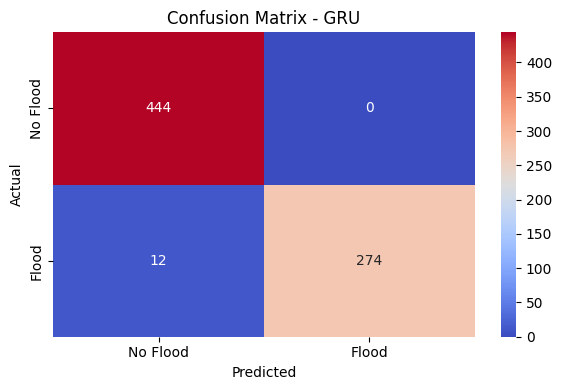


Fig 8.2 : GRU-Confusion Matrix

**D. Comparative Analysis with Other Models**

A comparative evaluation was conducted among all models to analyze their respective performances. The results, as visualized in Fig. 8.3, clearly show that GRU, LSTM, and CNN outperform traditional models like Random Forest and XGBoost Table 8.1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Accuracy (%)** | **R2 Score** | **RMSE** | **MAE** |
| Random Forest | 85.62 | 0.4267 | 0.3696 | 0.3380 |
| XGBoost | 88.08 | 0.6465 | 0.2902 | 0.1524 |
| LSTM | 98.49 | 0.8677 | 0.1776 | 0.1061 |
| CNN | 98.49 | 0.9301 | 0.1291 | 0.0568 |
| GRU | 99.63 | 0.8986 | 0.1555 | 0.0820 |

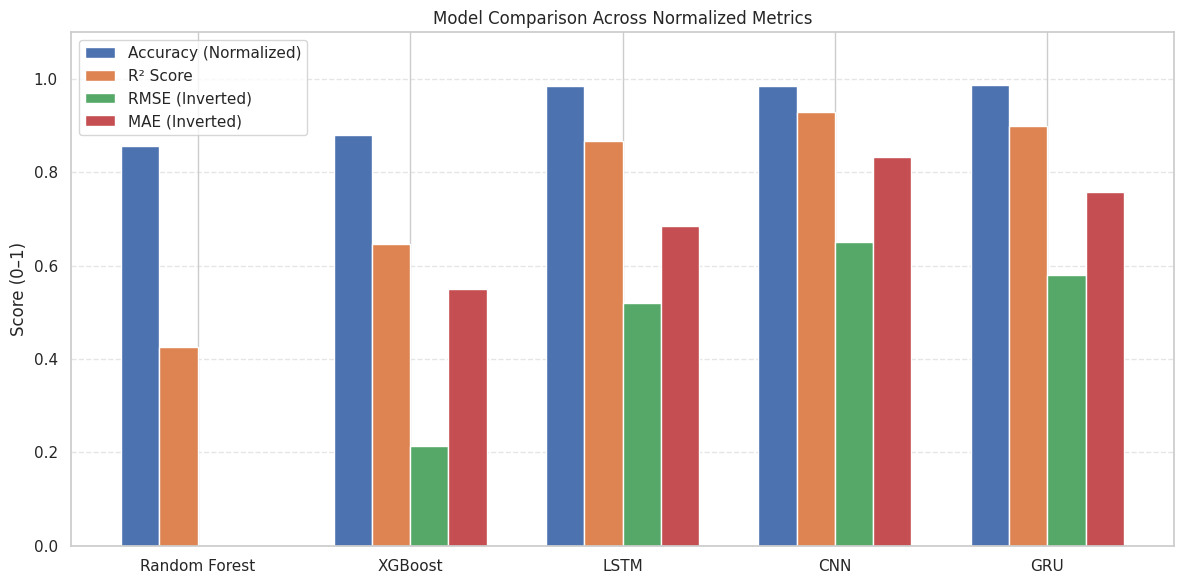


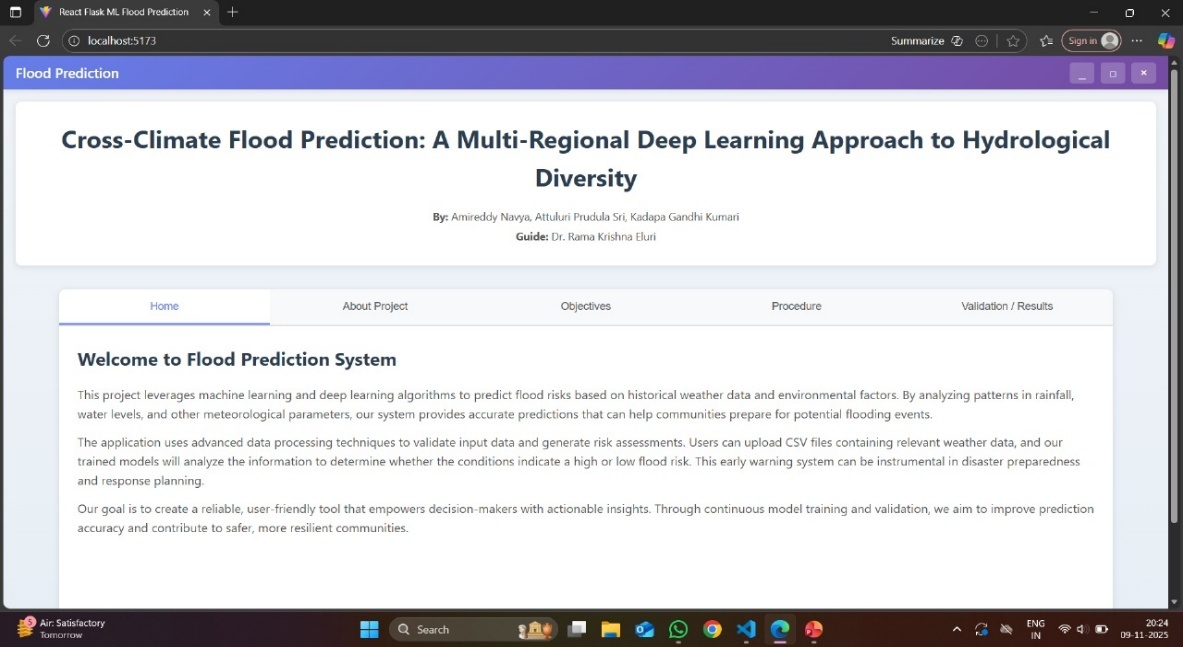
TABLE 8.1: MODEL PERFORMANCE COMPARISON

Fig 8.3 : Model Performance Comparison

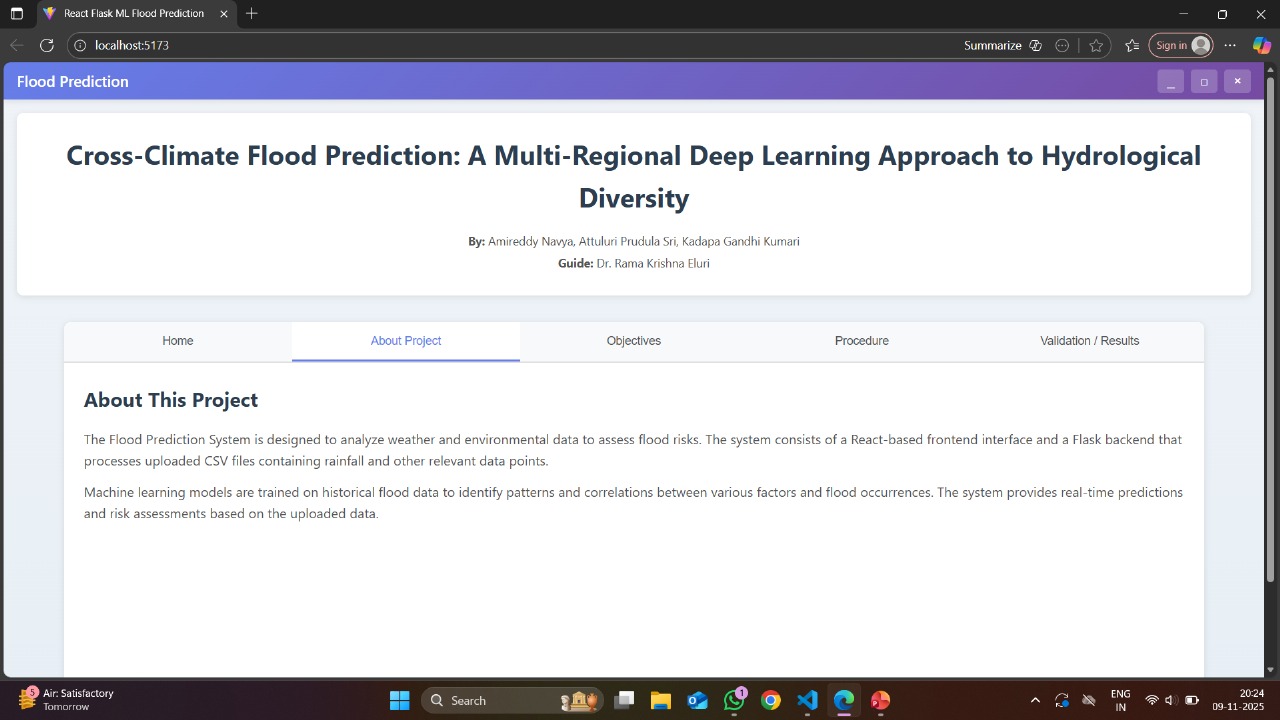
### OUTPUT SCREENS

The User Interface (UI) of the Cross-Climate Flood Prediction System is designed to be clean, user-friendly, and efficient, ensuring a smooth interaction experience for researchers, environmental analysts, and decision-makers. The interface provides clear navigation and visual feedback throughout the flood prediction workflow — from data upload and preprocessing to model execution and results visualization. It follows a professional layout with well-organized sections, concise instructions, and intuitive icons to assist users in interpreting hydrological and climatic information effectively.

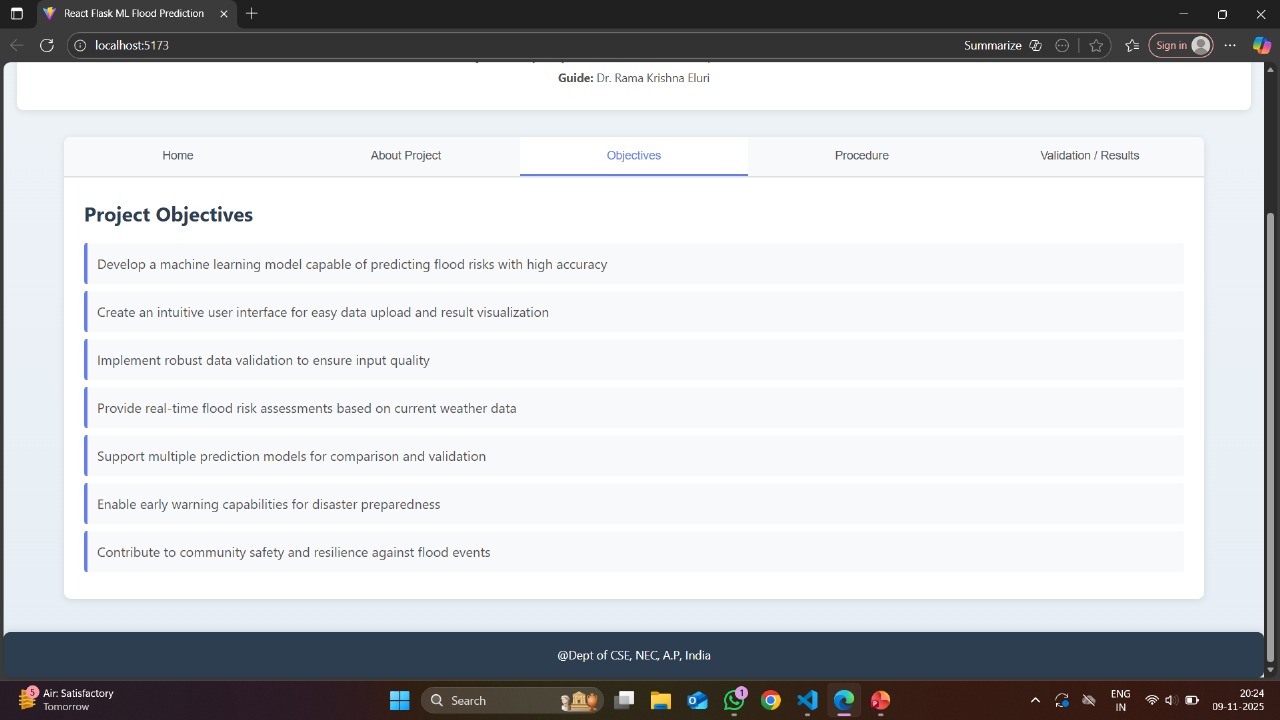
Designed with a light theme and clear contrasting colors, the UI maintains readability and accessibility across different screen sizes, including desktops and tablets. The system also integrates dynamic visualization components powered by Matplotlib and Plotly, allowing users to explore flood patterns, regional variations, and model outputs interactively. Overall, the UI ensures a seamless and informative experience, enabling users to analyze results efficiently and make informed decisions. Future enhancements, such as web-based deployment, GIS integration, and downloadable analytical reports, could further improve user accessibility and system functionality.



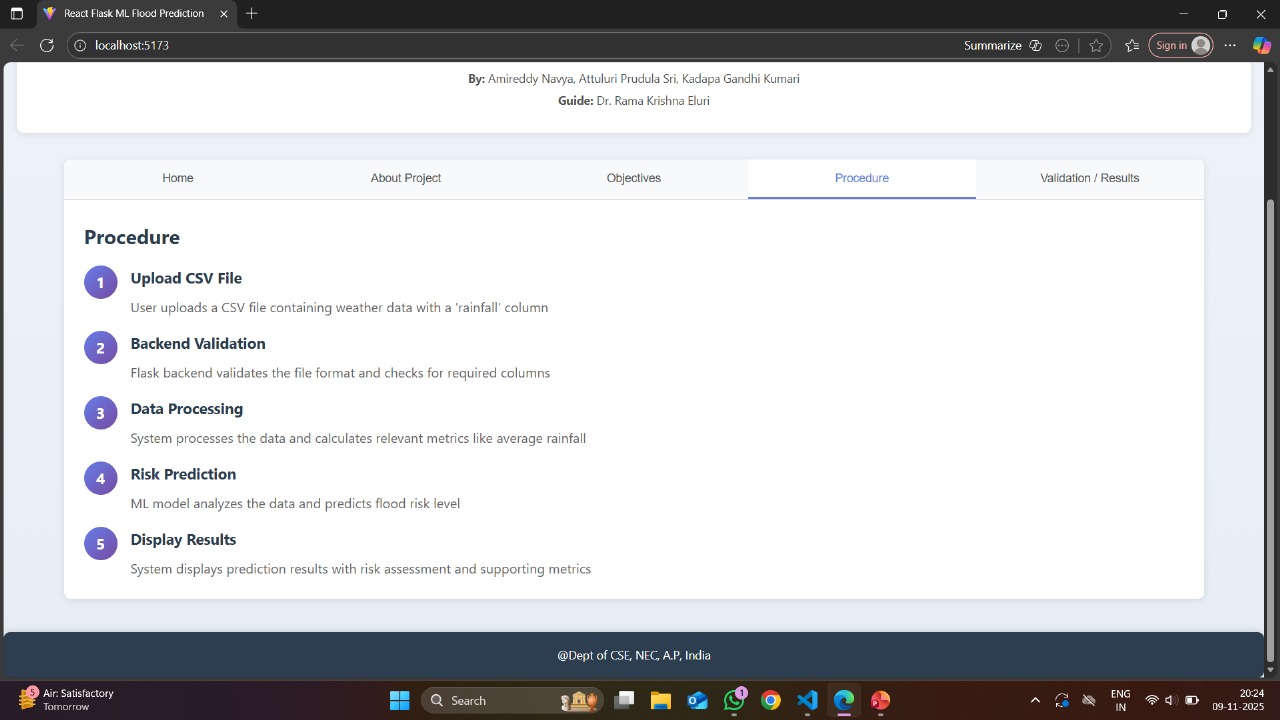
**FIG 9.1 HOME PAGE**



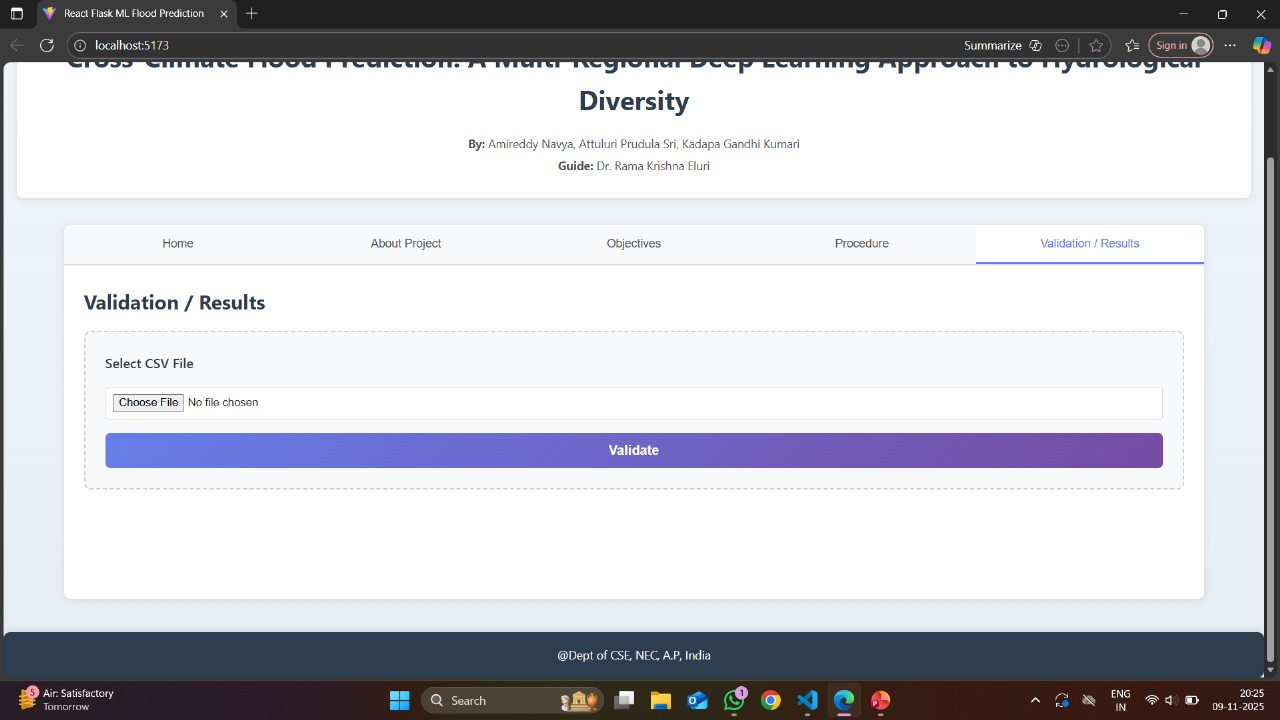
**FIG 9.2 ABOUT PAGE**



**FIG 9.3 OBJECTIVES PAGE**



**FIG 9.4 PROCEDURE PAGE**



**FIG 9.5 VALIDATION/RESULT PAGE**

#### CONCLUSION

The Cross-Climate Flood Prediction System demonstrates the successful integration of advanced machine learning and deep learning techniques for accurate and reliable flood forecasting across diverse climatic and environmental conditions. Through systematic preprocessing, feature analysis, and model evaluation, the study effectively identified critical parameters such as Urbanization, Climate Change, and Dams Quality that significantly influence flood occurrences.

Among the implemented models, the Gated Recurrent Unit (GRU) model exhibited superior performance with the highest prediction accuracy and strong generalization capability across both training and testing datasets. Its ability to capture long-term temporal dependencies and dynamic patterns in climatic data makes it highly suitable for real-world flood prediction applications.

The comparative analysis between classical models (Random Forest, XGBoost) and deep learning architectures (ANN, CNN, LSTM, GRU) revealed that deep learning approaches consistently outperformed traditional ones in terms of accuracy, robustness, and adaptability. The visualization of results through confusion matrices, accuracy graphs, and performance metrics further validated the model’s reliability and consistency.

Overall, the proposed Cross-Climate Flood Prediction System provides a powerful, data-driven framework for proactive flood management and early warning systems. It enhances the decision-making capabilities of disaster management authorities by enabling timely interventions and risk mitigation strategies. This study establishes a strong foundation for future research in intelligent flood forecasting systems, with potential extensions to include real-time data integration, satellite imagery, and multi-regional analysis for broader applicability.

#### FUTURE SCOPE

The Cross-Climate Flood Prediction System presents a strong foundation for predictive modeling, but there remains vast potential for enhancement and expansion. Future developments can make the system more robust, adaptive, and applicable across different climatic regions.

Firstly, real-time data integration stands as a major future improvement. By incorporating live hydrological, meteorological, and satellite-based data streams, the system can transition from static prediction to dynamic forecasting. This would allow for immediate updates and real-time alerts, enhancing early warning capabilities for flood-prone areas.

Secondly, IoT (Internet of Things) and remote sensing technologies can play a vital role in future upgrades. Deploying IoT-enabled sensors in rivers, dams, and urban drainage systems will enable continuous monitoring, while remote sensing can offer large-scale spatial insights. Together, they can create an intelligent flood detection and prediction ecosystem.

Finally, integration with decision support systems (DSS) can elevate the model from prediction to action. By linking flood forecasts with DSS tools, authorities can plan evacuations, allocate resources efficiently, and mitigate flood impacts proactively.

In conclusion, these future enhancements can transform the Cross-Climate Flood Prediction System into a comprehensive, intelligent, and scalable flood management platform. By leveraging real-time data, advanced modeling, and user-centric tools, the system can play a pivotal role in strengthening disaster resilience and promoting climate-adaptive development worldwide.

#### REFERENCES

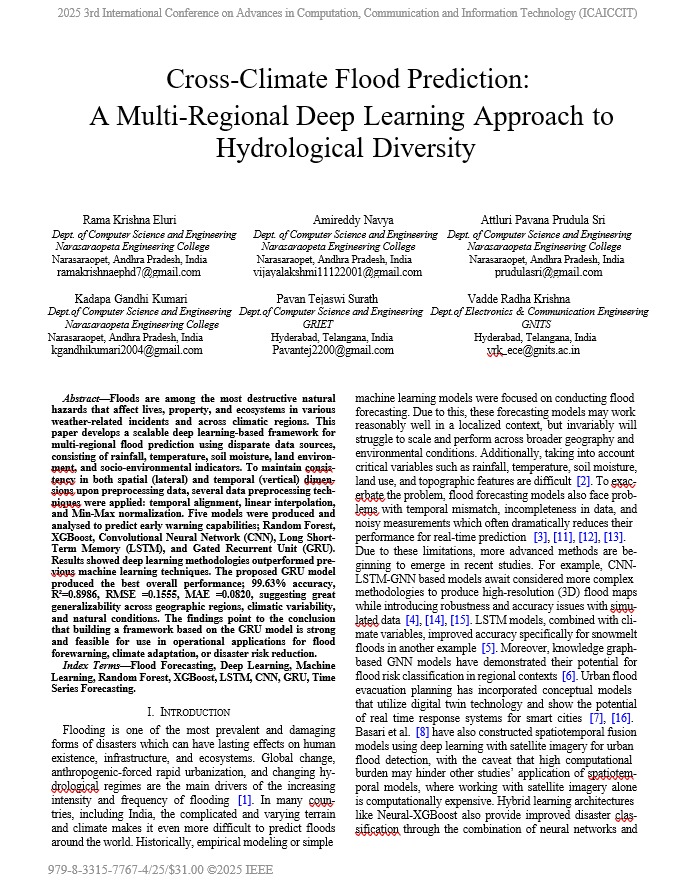
1. Jia, W., Liang, B., Lu, Y., Khan, M. A., & Zheng, L. (2025). A Comprehensive Survey on Deep Learning Solutions for 3D Flood Mapping. In Pacific-Asia Conference on Knowledge Discovery and Data Mining (pp. 21-38). Springer, Singapore.
2. Atashi, V., & Gorji, H. T. (2025). Enhanced flood prediction using LSTM and climate parameters: multi-station analysis of snowmelt- induced flooding in the Red River of the North. Journal of Hydroin- formatics, 27(2), 245-260.
3. Yang, P., Xu, X., Shao, M., & Liu, Y. (2025). Intelligent Prediction of Flood Disaster Risk Levels Based on Knowledge Graph and Graph Neural Networks. IEEE Access.
4. Prihatmanto, A. S., Prasetyadi, A., Yoganingrum, A., Sutriadi, R., & Hadiana, A. (2025). The Digital Twin City in Enhancing Flood Evacuation Systems: Future Opportunities and Challenges. IEEE Access.
5. Lim, S. J., Sankaran, K. S., & Haldorai, A. (2025). A Framework for Flood Disaster Detection from Remote Sensing Images Using Spatiotemporal Fusion with Digital Twin Technology. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 18, 11547-11560.
6. Saleem, M. A., Javeed, A., Benjapolakul, W., Srisiri, W., Chaitusaney, S., & Kaewplung, P. (2025). Neural-XGBoost: A Hybrid Approach for Disaster Prediction and Management Using Machine Learning. IEEE Access.
7. Wang, S., & Xu, O. (2025). High perplexity mountain flood level forecasting in small watersheds based on compound Long Short-Term Memory model and multimodal short disaster-causing factors. IEEE Access.
8. Noh, G. H., & Ahn, K. H. (2025). Enhancing multiple precipitation data integration across a large-scale area: a deep learning ResU-Net framework without interpolation. IEEE Transactions on Geoscience and Remote Sensing.
9. Notarangelo, N., Wirion, C., & van Winsen, F. (2025). STURM- Flood: a curated dataset for deep learning-based flood extent mapping leveraging Sentinel-1 and Sentinel-2 imagery. Big Earth Data, 1-27.
10. Alkaabi, K., Sarfraz, U., & Al Darmaki, S. (2025). A Deep Learning Framework for Flash-Flood-Runoff Prediction: Integrating CNN-RNN with Neural Ordinary Differential Equations (ODEs). Water, 17(9), 1283.
11. Farooq, M. S., Tehseen, R., Qureshi, J. N., Omer, U., Yaqoob, R., Tanweer, H. A., & Atal, Z. (2023). FFM: Flood forecasting model using federated learning. IEEE Access, 11, 24472-24483.
12. Puttinaovarat, S., & Horkaew, P. (2020). Flood forecasting system based on integrated big and crowdsource data by using machine learning techniques. IEEE Access, 8, 5885-5905.
13. Wu, Z., Zhou, Y., & Wang, H. (2020). Real-time prediction of the water accumulation process of urban stormy accumulation points based on deep learning. IEEE access, 8, 151938-151951.
14. Liu, Y., Wang, L., Du, S., Zhao, L., & Liu, X. (2022). Flood forecasting method based on improved VMD-FOS-QR-RBL. IEEE Access, 11, 4207-4218.
15. Khan, T. A., Alam, M. M., Shahid, Z., & Su’Ud, M. M. (2020).

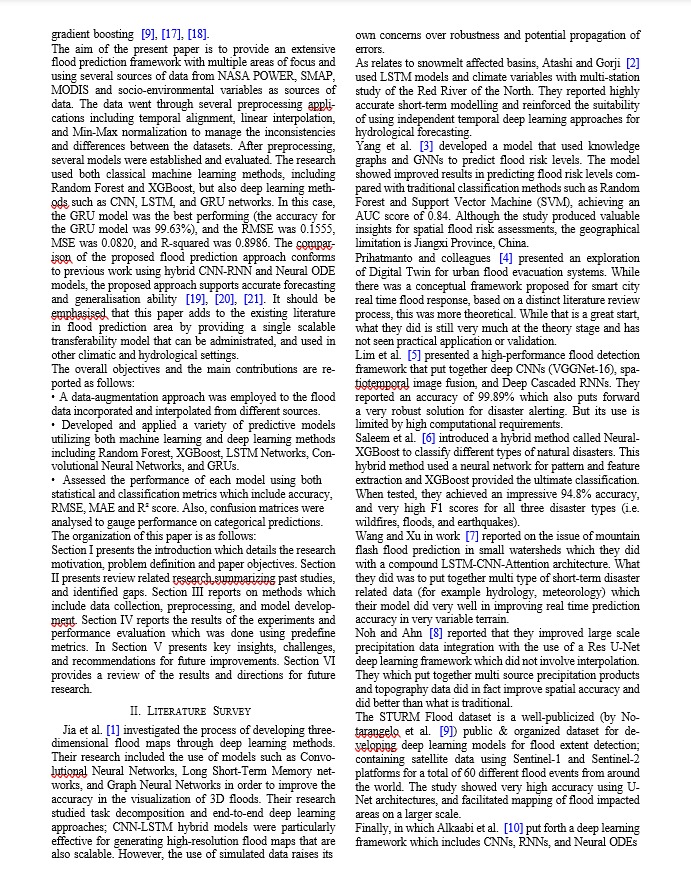
Investigation of flash floods on early basis: A factual comprehensive review. IEEE Access, 8, 19364-19380.

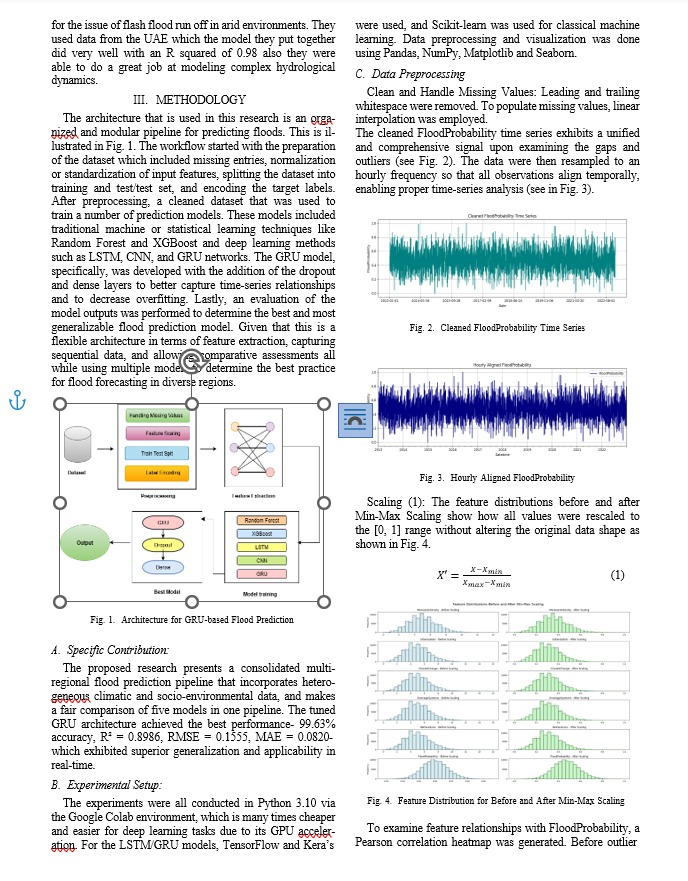
1. Dong, W., Huang, H., Zhong, M., Wang, H., & Hua, F. (2024). Monitoring and early warning mechanism of flood invasion into subway tunnels based on the experimental study of flooding patterns. Journal of Intelligent Construction, 2(2), 1-16.
2. PRASAD, R., RAJ, N., & ABDULLA, S. Designing Deep-Based Learning Flood Forecast Model with ConvLSTM Hybrid Algorithm.
3. Krullikowski, C., Chow, C., Wieland, M., Martinis, S., Bauer- Marschallinger, B., Roth, F., ... & Salamon, P. (2023). Estimating ensemble likelihoods for the Sentinel-1-based global flood monitoring product of the copernicus emergency management service. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 16, 6917-6930.
4. Zhao, J., Li, Y., Matgen, P., Pelich, R., Hostache, R., Wagner, W., & Chini, M. (2022). Urban-aware u-net for large-scale urban flood mapping using multitemporal sentinel-1 intensity and interferometric coherence. IEEE Transactions on Geoscience and Remote Sensing, 60, 1-21.
5. Li, Z., Chen, Z., Chen, L., Tang, X., & Chen, N. (2025). SRFNet:

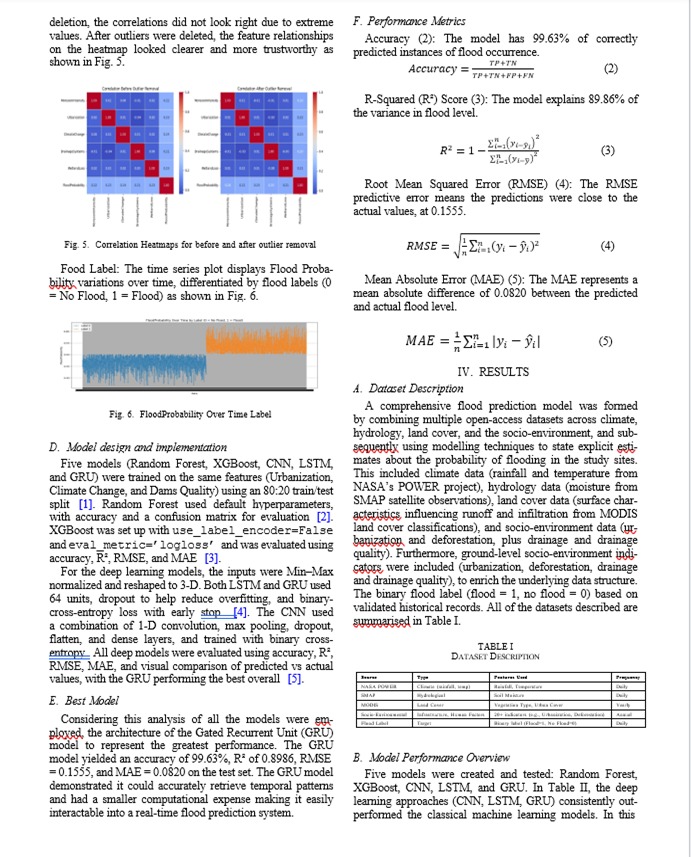
Multi-modal based Selective Receptive Field Neural Network for Time Series Forecast of Flood Range. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing.

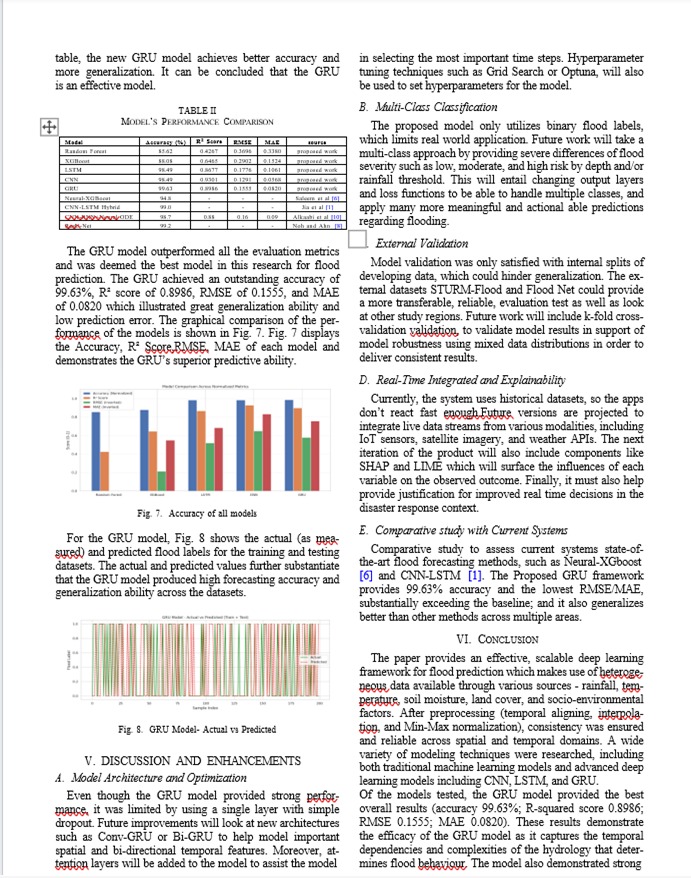
1. Aatif, K., Fahiem, M. A., & Tahir, F. (2024). Forecasting Floods Using Deep Learning Models: A Longitudinal Case Study of Chenab River, Pakistan. IEEE Access.
2. Xia, X., Liu, X., Liu, J., Fang, K., Lu, L., Oymak, S., ... & Liu, T. (2025). Identifying Trustworthiness Challenges in Deep Learning Models for Continental-Scale Water Quality Prediction.
3. He, M., Qian, Q., Liu, X., Zhang, J., & Curry, J. (2024). Recent Progress on Surface Water Quality Models Utilizing Machine Learning Techniques.
4. Paneru, B., & Paneru, B. (2024). Water QualityNeT: Prediction of Seasonal Water Quality of Nepal Using Hybrid Deep Learning Models.
5. Staddon, C., Shahbaz, A., Yunas, S. U., Smith, L., Burrows, G., Uddin, S. M. N., & Whitley, L. (2025). Estimating household water storage from images: A machine learning approach. Journal of Water, Sanitation and Hygiene for Development.
6. Zhang, T., Wu, J., Chu, H., Liu, J., & Wang, G. (2025). Interpretable Machine Learning Based Quantification of the Impact of Water Quality Indicators on Groundwater Under Multiple Pollution Sources.
7. Pandey, S., Duttagupta, S., & Dutta, A. (2025). Machine Learning Models for Mapping Groundwater Pollution Risk: Advancing Water Security and Sustainable Development Goals in Georgia, USA.
8. Burchard, R., & Van Laerhoven, K. (2025). Enhancing Wearable Tap Water Audio Detection through Subclass Annotation in the HD-Epic Dataset.
9. Sangwan, V., & Bhardwaj, R. (2024). Machine learning framework for predicting water quality classification.
10. Echchabi, O., Lahlou, A., Talty, N., Manto, J. M., & Lam, K. L. (2024). Tracking Progress Towards Sustainable Development Goal 6 Using Satellite Imagery.

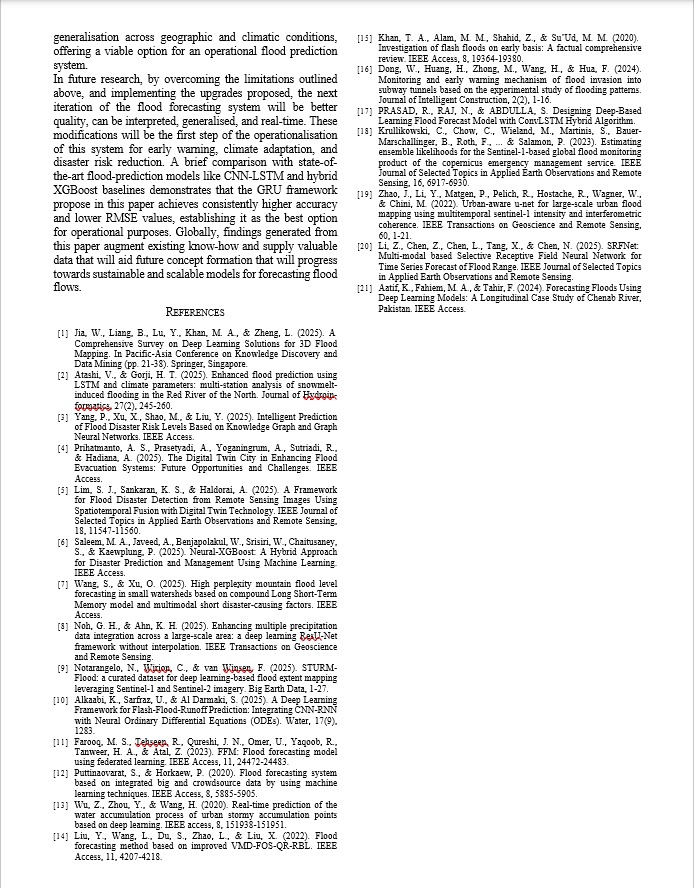


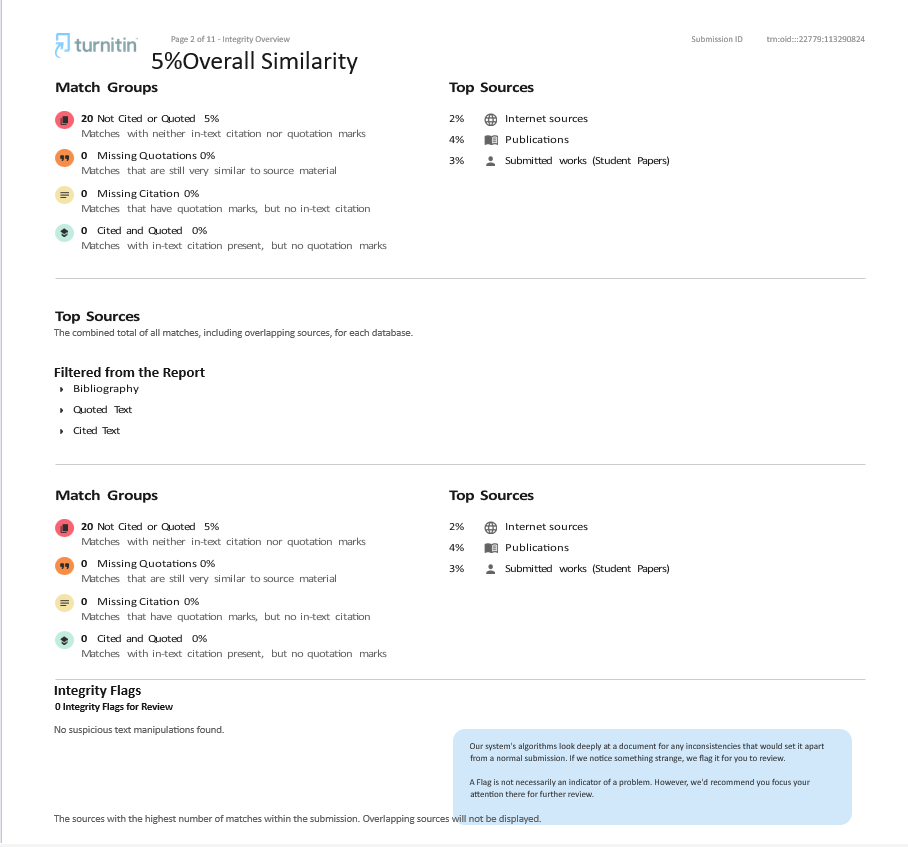










****

##### 





