



# Predicaster

---

[AccessDenied](#)

Tanisha Basu

Adarsh Rout

Shashank Raj

# Table of Content

## Phase 1 Introduction

1. Problem Statement
2. Objective
3. Domain
4. Flowchart

## Phase 2 Implementation

1. Dataset Selection
2. Earthquake Model
  - a. Methodology
  - b. Model Training & Evaluation
3. Flood Model
  - a. Methodology
  - b. Model Training & Evaluation
4. Hurricane Model
  - a. Methodology
  - b. Model Training & Evaluation
5. YOLOv

## Phase 3 Integration

1. Flask Server
  - a. Model Integration
  - b. API Design
2. Web Application
  - a. Frontend Development
  - b. Backend Integration
  - c. API Testing
  - d. ChatBot Integration

## Phase 4 How to run

## Phase 5 Result

## Phase 6 Conclusion and Future Scope

## Phase 7 References

# Introduction

## Problem Statement

Natural disasters cause significant loss of life and property, and timely prediction and response are critical to minimizing their impact. Develop an AI system that predicts natural disasters such as earthquakes, floods, and hurricanes using historical data and real-time inputs, and suggests optimal response strategies.

## Objective

1. Analyze the extent of natural disasters in various places
2. Able to predict its extent and protect them
3. Able to do real-time victim prediction using Yolov detection that will significantly help the rescuers in tracing people

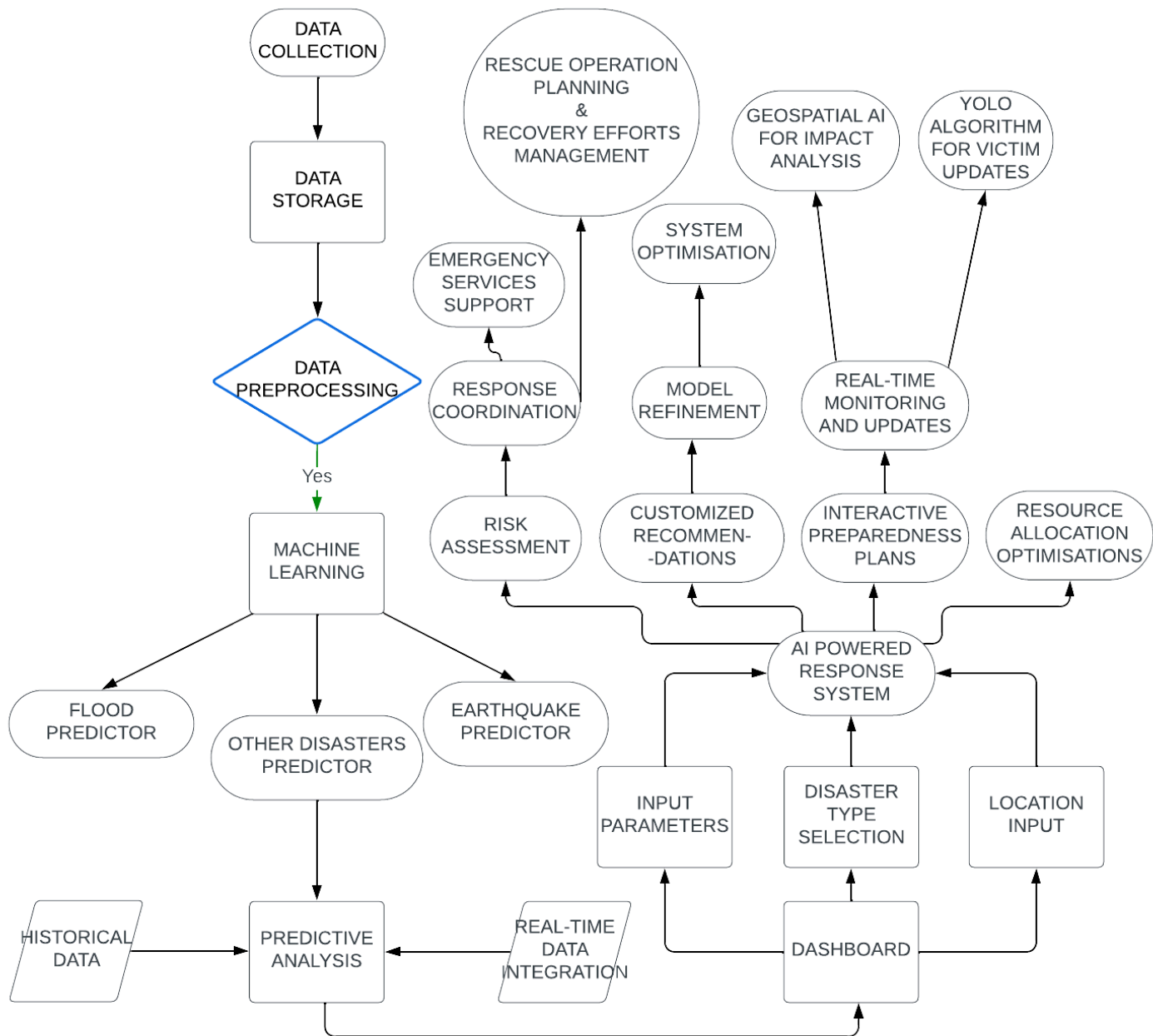
## Domain

Environmental monitoring, Emergency Management, Public Safety, Government agencies, Disaster relief organizations, Infrastructure management, Climate research, and Insurance.

## Overview

By encapsulating these functionalities, Predicaster should be able to deliver a user-centric and simplified user experience. This not only analyses the extent of natural disasters at various places in India and Worldwide but also predicts its extent and provides a response and precautions. With the help of a Chatbot, providing prediction and response is much simplified

## Flowchart



# Implementation

## Dataset Selection

1. Earthquake Dataset
  - a. Source: [Earthquake Dataset](#)
  - b. Description: The dataset contains information on global earthquake events, including date, time, location, depth, magnitude, and impact.
  - c. Reason: This dataset provides comprehensive and up-to-date information on earthquake events, which is crucial for analyzing patterns, predicting future occurrences, and understanding the impact on different regions.
2. Flood Dataset
  - a. Source: [Flood Dataset](#)
  - b. Description: This is the record of the monthly rainfall index from 1900- 2018 while telling whether a flood occurred that month or not and can be used to predict floods by observing the rainfall pattern.
  - c. Reason: This dataset offers detailed insights into flood events, which are essential for risk assessment, emergency response planning, and understanding the socio-economic impacts of flooding.
3. Hurricane Dataset
  - a. Source: [Hurricane Dataset](#)
  - b. Description: The National Hurricane Center (NHC) conducts a post-storm analysis of each tropical cyclone in the Atlantic basin (i.e., North Atlantic Ocean, Gulf of Mexico, and Caribbean Sea) and the North Pacific Ocean to determine the official assessment of the cyclone's history. This analysis uses all available observations, including those that may not have been available in real-time. In addition, NHC conducts ongoing reviews of any retrospective tropical cyclone analyses brought to its attention and regularly updates the historical record to reflect changes introduced.
  - c. Reason: This dataset is critical for tracking hurricane patterns, predicting future events, and assessing their impact on affected regions. It helps in improving preparedness and mitigation strategies for hurricane-prone areas.

## Earthquake Model

### Methodology

1. Libraries Used: TensorFlow, NumPy, Pandas, Matplotlib, Sklearn, Basemap
2. Dataset
  - a. Imported earthquake data from earthquake.csv
  - b. Selected relevant columns: **Date**, **Time**, **Latitude**, **Longitude**, **Depth**, and **Magnitude**.
3. Data Visualisation
  - a. Utilized **Basemap** for plotting global maps.
  - b. Converted longitudes and latitudes to map coordinates.
  - c. Plotted earthquake locations with blue markers on the map.

### Model Training & Evaluation

1. **Chose `sklearn.ensemble.RandomForestRegressor`**: Selected this regressor for its robustness and high performance. Configured with `n_estimators=100` and `random_state=42` for reproducibility.
2. **Fitted the model using `sklearn`**: Trained the `RandomForestRegressor` on the training data (`X_train` and `y_train`) to learn patterns and relationships in the data.
3. **Utilized `sklearn`**: Predicted the earthquake magnitude on the test set (`X_test`).
4. **Predicted Earthquake Magnitude using `RandomForestRegressor`**: Utilized the trained model to predict the earthquake magnitude based on the input data.

## Flood Model

### Methodology

1. Libraries Used: Pandas, NumPy, Sklearn
2. Dataset: Imported from flood.csv
3. Data Preprocessing:
  - a. **Used pandas:** Checked for the existence of the target variable column ('RL\_FLOODS' or 'RL\_FLOODS') to ensure the correct column name is used for subsequent operations.
  - b. **Employed sklearn.preprocessing.LabelEncoder:** Transformed the categorical target variable (FLOODS) into a numerical format to make it suitable for machine learning algorithms.
4. Feature Scaling:
  - a. **Used sklearn.model\_selection.train\_test\_split:** Divided the dataset into training and testing sets with an 80-20 split (`test_size=0.2`, `random_state=42`) to evaluate model performance on unseen data.
  - b. **Utilized pandas:** Extracted the encoded FLOODS column to use as the target variable in model training.

### Model Training & Evaluation

1. **Chose sklearn.ensemble.RandomForestClassifier:** Selected this classifier for its robustness and ability to handle high-dimensional data with numerous features. Configured with `n_estimators=100` and `random_state=42` to ensure reproducibility.
2. **Fitted the model using sklearn:** Trained the `RandomForestClassifier` on the training data (`X_train` and `y_train`) to learn patterns and relationships in the data.
3. **Utilized sklearn:** Predicted flood occurrences on the test set (`X_test`) to assess the model's performance.
4. **Calculated accuracy using sklearn.metrics.accuracy\_score:** Measured the model's performance by comparing the predicted values with the actual values in the test set. Generated a classification report to provide detailed performance metrics.

## Hurricane Model

### Methodology

1. Libraries Used: Pandas, NumPy, Sklearn
2. Dataset: Imported from hurricane.csv
3. Data Preprocessing:
  - a. **Used pandas:** Applied a custom function to convert latitude and longitude values from string format (with 'N', 'S', 'E', 'W') to numeric values.
  - b. **Used numpy:** Handled NaN values during conversion.
  - c. **Filled NaN Values with pandas:** Filled missing values in numeric columns with the mean of their respective columns.
4. Feature Scaling
  - a. **Utilized pandas:** Selected relevant features for the model, including latitude, longitude, wind data, and extracted date components.
  - b. **Used sklearn.model\_selection.train\_test\_split:** Split the dataset into training and testing sets with an 80-20 split (`test_size=0.2`, `random_state=42`) to evaluate model performance on unseen data.
5. Data Manipulation
  - a. **Employed sklearn.impute.SimpleImputer:** Used the imputer to replace missing values with the mean in both training and test sets.

### Model Training & Evaluation

1. **Chose sklearn.ensemble.RandomForestRegressor:** Selected this regressor for its robustness and high performance. Configured with `n_estimators=100` and `random_state=42` for reproducibility.
2. **Fitted the model using sklearn:** Trained the `RandomForestRegressor` on the training data (`X_train` and `y_train`) to learn patterns and relationships in the data.
3. **Utilized numpy:** Constructed a NumPy array from the input parameters to format the data correctly for the model's prediction function.
4. **Applied the imputer:** Used the previously fitted imputer to handle any missing values in the input data.
5. **Predicted maximum wind speed using the trained RandomForestRegressor model:** Utilized the model to predict the maximum wind speed based on the input data.



## Yolov Detection

### Objective

The main objective of YOLO Detection is to

1. Trace out the people Identify and Locate Survivors and Victims.
2. Detect structural damage, hazardous objects, and other environmental dangers.
3. Track the progress and effectiveness of rescue operations.
4. Create detailed visual records of the disaster scene.

### Methodology

1. **Prepare Environment:** Install necessary libraries and obtain YOLO model files.
2. **Load YOLO Model:** Load the YOLO configuration, weights, and class labels.
3. **Process Video:** Capture video frames and process each frame for object detection.
4. **Perform Detection:** Use YOLO to detect objects in each frame and apply post-processing techniques like Non-Maximum Suppression.
5. **Display/Save Output:** Show or save the video with detected objects highlighted.

# Integration

## Flask Server

### Model Integration

1. Three pre-trained models are integrated: earthquake, flood, and hurricane.
2. Models are loaded using joblib, suggesting they were likely trained separately and saved for integration.
3. Each model is designed to handle different types of input data relevant to its prediction task.

### API Design

#### API Framework

1. Utilizes Flask to create a RESTful API.
2. Implements CORS (Cross-Origin Resource Sharing) to handle cross-origin requests.

#### API Structure

1. Defines three main routes: '/earth', '/flood', and '/hurri'.
2. Each route corresponds to a specific prediction model.
3. All routes use POST method for receiving input data.

#### Data Handling

1. Employs numpy for numerical operations across all models.
2. Uses pandas for data manipulation, particularly in the flood prediction route.

#### Response Format

1. All predictions are returned as JSON responses.
2. Each response contains a key specific to the prediction type (e.g., "predicted\_magnitude" for earthquakes).

## Web Application

### Frontend Development

1. Designed a responsive and intuitive user interface for simplified usage across various devices.
2. Utilized ReactJS as the primary framework, providing a single-page application (SPA) architecture for a seamless user experience.
3. Implemented React Hooks for efficient state management and side effects handling.
4. Applied Tailwind CSS for rapid UI development, ensuring a visually appealing and consistent design system.

### Flask Server Integration

1. Utilized Axios library for making API calls, leveraging its features for request/response interception and automatic transforms.
2. Implemented a centralized API service module for better code organization and reusability.
3. User-provided data and input parameters are sent to the server in JSON format.
4. Implemented error handling and loading states to manage API communication effectively.
5. Returned the Predicted Value in JSON format.

### API Testing

1. Implemented integration tests to ensure proper communication between frontend and backend.
2. Used Postman to properly test and validate API endpoints, ensuring correct functionality and response formats.

### ChatBOT Integration

1. Compiled a comprehensive dataset relevant to natural disaster predictions and safety measures.
2. Utilized chatbase.co's platform to create a custom chatbot.
3. Uploaded the prepared dataset to train the chatbot on domain-specific knowledge.
4. Fine-tuned the chatbot's responses and behavior using chatbase.co's tools and settings.
5. Obtained the embedded code snippet provided by chatbase.co for the created chatbot to Incorporate the chatBOT into the React application's codebase.

# How to Run

## Steps

1. Clone the project from Github Repo
2. `cd woodpecker`
3. To initiate the Flask Server
  - a. `cd flaskapp`
    - i. This will take you to the flask app directory where all the files and modules exist.
  - b. `Python install -r requirement.txt`
    - i. All the python libraries used to the model and server to initiate are listed here.
    - ii. Installing these modules in mandatory for the server to start and api calls to be made.
  - c. Wait for it to be completed
    - i. The installation process can take some time. Please make sure that all the modules are installed correctly.
  - d. `Python app.py`
    - i. This will run the app.py file and create a server for the API calls and listening to the web request.
4. To Run the Web Application
  - a. Open a new Terminal
  - b. `cd woodpecker`
  - c. `cd webApp`
  - d. `cd client`
    - i. The above steps will take you to the Web Application directory.
    - ii. The Directory may look like Eg woodpecker/webApp/client
  - e. `npm i`
    - i. This command is used to installing all the frameworks and packages used in the frontend.
    - ii. For this command to work, Please install nodejs in your system before running this command and run this command.
  - f. Wait for it to be completed
  - g. `npm start`
    - i. This will start the Web Application.
5. Your Web Application will be running at <http://localhost:3000>

## 6. YOLOv Steps

- a. **Install Python:** Ensure you have Python installed on your system. You can download it from [python.org](https://python.org).
- b. **Create a Virtual Environment (optional but recommended):**
  - i. Navigate to your project directory:
- c. **Install Dependencies:**
- d. Ensure you have Flask and other necessary packages installed. You can install them using pip:
  - i. `sh`
  - ii. `pip install flask werkzeug`
- e. **Clone YOLOv5 Repository:**
  - i. Navigate to your project directory and clone the YOLOv5 repository:  
`sh`  
`git clone https://github.com/ultralytics/yolov5`
  - ii. `cd yolov5`
- f. **Start the Flask Server:**
- g. Ensure you are in your project directory and your virtual environment is activated.
- h. Run the Flask application:
  - i. `python app.py`

**Sample Input Video :**  new\_input.mp4

**Sample Output Video :**  output\_video.mp4



○



# Result

1. **Advanced Predictive Analytics:** Utilize machine learning algorithms and real-time data integration to predict natural disasters such as floods, earthquakes. The system will leverage historical data, weather patterns, seismic activity, and environmental factors to provide accurate predictions up to 15 days in advance.
2. **Location-Specific Recommendations:** Provide customized recommendations and response strategies based on the user's specific location. By entering their location, users can receive tailored advice on how to prepare for and respond to potential disasters.
3. **Interactive Disaster Preparedness Plans:** Offer users interactive and personalized disaster preparedness plans that they can follow before, during, and after a disaster.
4. **AI-Powered Response Coordination:** Leverage AI to coordinate response efforts more effectively during and after a disaster. The system can analyze real-time data to suggest optimal strategies for rescue operations, resource distribution, and recovery efforts.
5. **Real Time Victim Prediction:** Real-time victim prediction using YOLO enables rapid and accurate detection of people in disaster zones, helping rescuers deployed by the Government to quickly locate and assist victims or survivors.

# Conclusion & Future Aspect

1. **Scenario Planning:** Offer scenario planning tools that allow emergency managers to simulate different disaster scenarios and response strategies.
2. **Geospatial AI for Detailed Impact Analysis:** Use geospatial AI to create detailed maps and models of disaster impact areas.
3. **Damage Assessment:** Employ AI to analyze satellite and drone imagery for rapid and accurate damage assessments.

# References

1. Dataset Collection
  - a. <https://www.kaggle.com>
2. Python Documentation and Libraries
  - a. <https://docs.python.org/3/>
3. Sklearn
  - a. <https://scikit-learn.org/stable/>
4. Chatbase
  - a. <https://www.chatbase.co/>
5. YOLO
  - a. <https://pjreddie.com/darknet/yolo/>
  - b.  Object Detection API for images and video using YOLOv5 and Flask
6. Generative Pre-trained Transformer
  - a. <https://chatgpt.com/>
  - b. <https://claude.ai/chat>
7. Hurricane Dataset
  - a. <https://www.kaggle.com/datasets/noaa/hurricane-database>
8. Flood Dataset
  - a. <https://www.kaggle.com/datasets/mukulthakur177/kerela-flood/data>
9. Earthquake Dataset
  - a. <https://www.kaggle.com/code/mahadevmm9/earthquake-prediction/input>
10. Sample Video
  - a.  Video shows tragic moment an Indian family is swept away by floodwat...