Automatic Cyclone Tracking System

A PROJECT REPORT

Submitted by

Moreddy Sankeerth Reddy
19MIS1086

in partial fulfillment for the award of the degree of

Master of Technology

in

Software Engineering (5 Year Integrated Programme)

School of Computer Science and Engineering

Vellore Institute of Technology

Vandalur - Kelambakkam Road, Chennai - 600 127

April - 2024

VIT®
Vellore Institute of Technology

School of Computer Science and Engineering

DECLARATION

I hereby declare that the project entitled Automatic cyclone tracking system submitted by me to the School of Computer Science and Engineering, Vellore Institute of Technology, Chennai, 600 127, in partial fulfillment of the requirements of the award of the degree of Master of Technology in Software Engineering (5 year Integrated Programme) and as part of SWE1904 - Capstone Project is a bona-fide record of the work carried out by me under the supervision of Prof. Dr MENAKA PUSHPA A . I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma of this institute or of any other institute or University.

Place: Chennai

Date:

Signature of Candidate

VIT®
Vellore Institute of Technology

School of Computer Science and Engineering

CERTIFICATE

This is to certify that the report entitled Automatic cyclone tracking system is prepared and submitted by Moreddy Sankeerth Reddy (Reg. No. 19MIS1086) to Vellore Institute of Technology, Chennai, in partial fulfillment of the requirement for the award of the degree of Master of Technology in Software Engineering (5 year Integrated Programme) and as part of SWE1904 – Capstone Project is a bona-fide record carried out under my guidance. The project fulfills the requirements as per the regulations of this University and in my opinion meets the necessary standards for submission.

Guide/Supervisor	HoD
Name:	Name: Dr. Nisha V M
Date:	Date:
Examiner	Examiner
Name:	Name:
Date:	Date:

(Seal of SCOPE)

Acknowledgement

I am obliged to give my appreciation to a number of people without whom I could not have completed this thesis successfully.

I would like to place on record my deep sense of gratitude and thanks to my internal guide Prof. DR. MENAKA PUSHPA A, School of Computer Science and Engineering (SCOPE), Vellore Institute of Technology, Chennai, whose esteemed support and immense guidance encouraged me to complete the project successfully.

I would like to thank our HoD Dr. Nisha V M, School of Computer Science and Engineering (SCOPE) and Project Coordinator Dr.Kanchana Devi V, Vellore Institute of Technology, Chennai, for their valuable support and encouragement to take up and complete this thesis.

Special mention to our Dean Dr. Ganesan R, Associate Deans Dr. Parvathi R and Dr. S. Geetha, School of Computer Science and Engineering (SCOPE), Vellore Institute of Technology, Chennai, for motivating us in every aspect of software engineering.

I thank our management of Vellore Institute of Technology, Chennai, for permitting me to use the library and laboratory resources. I also thank all the faculty members for giving me the courage and the strength that I needed to complete my goal. This acknowledgment would be incomplete without expressing the whole hearted thanks to my family and friends who motivated me during the course of my work.

Moreddy Sankeerth Reddy

19MIS1086

Abstract

In India, the frequency and intensity of cyclones pose significant challenges to disaster management. Existing tracking systems often struggle to accurately predict the path of cyclones, leading to ineffective preparedness and response efforts. To address these shortcomings, this capstone project focuses on developing an automatic cyclone tracking system using advanced deep learning models.The proposed system utilizes cutting-edge models such as CNN-GRU and CNN-LSTM, which have not been extensively applied to cyclone tracking in the Indian context. By comparing these models to traditional deep learning approaches, the project aims to improve the accuracy and reliability of cyclone tracking. This advancement is crucial for enhancing disaster management strategies and minimizing the impact of cyclones on vulnerable communities.The significance of this research lies in its potential to use cyclone tracking in India. By leveraging the power of deep learning, the project seeks to provide more precise predictions of cyclone paths, enabling authorities to take proactive measures to protect lives and property. Additionally, the project's focus on the Indian context is crucial, as the country faces unique challenges due to its geographical location and population density.

Contents

List of figures and tables

Chapter 1
Introduction

1.1 Overview

In a country like India, where cyclones frequently wreak havoc, having an effective tracking system is crucial for minimizing damage and saving lives. However, existing methods often fall short in accurately predicting the path of these cyclones. Traditional techniques lack the precision needed to anticipate the exact latitude and longitude coordinates of cyclones, leaving communities vulnerable to their unpredictable nature.

To address this gap, I have developed an automatic cyclone tracking system using advanced and hybrid deep learning models: CNN-GRU and CNN-LSTM while comparing it to normal deep learning models of those ones. These models have not been widely implemented for Indian cyclones before, making this research particularly significant. By harnessing the power of deep learning, we aim to enhance the accuracy and reliability of cyclone tracking, ultimately improving preparedness and response efforts.

Through this project, we hope to provide valuable insights into the behavior of cyclones in the Indian context, enabling better-informed decisions and more effective disaster management strategies. By predicting the latitude and longitude of cyclones with greater precision, we aspire to contribute to the safety and resilience of communities vulnerable to these natural disasters.

1.2 Background

1.2.1 History of Cyclones in India

Tropical cyclones or cyclones in Indian and Australian context are called with different names throughout the world like Hurricane in Atlantic and east pacific oceans, typhoons in west pacific are very powerful storms characterized by a low-pressure center and strong rotating winds, typically accompanied by thunderstorms and heavy rainfall. These intense weather systems form over warm ocean waters and are classified by wind speeds, ranging from Category 1 (weakest) to Category 5 (strongest), with winds exceeding 119 km (about 73.94 mi) per hour. Cyclones bring significant dangers, including high-speed winds causing structural damage, uprooting trees, and leading to storm surges resulting in coastal flooding. According to a research paper **[16]** covering a span of 50 years from 1970 to 2019, India experienced a staggering 117 cyclones, leading to the tragic loss of over 40,000 lives. This study on extreme weather events also highlights a significant decline in the mortality rate caused by tropical cyclones over the past decade.

The below picture is the depection of the cyclones in India from 1900-2023

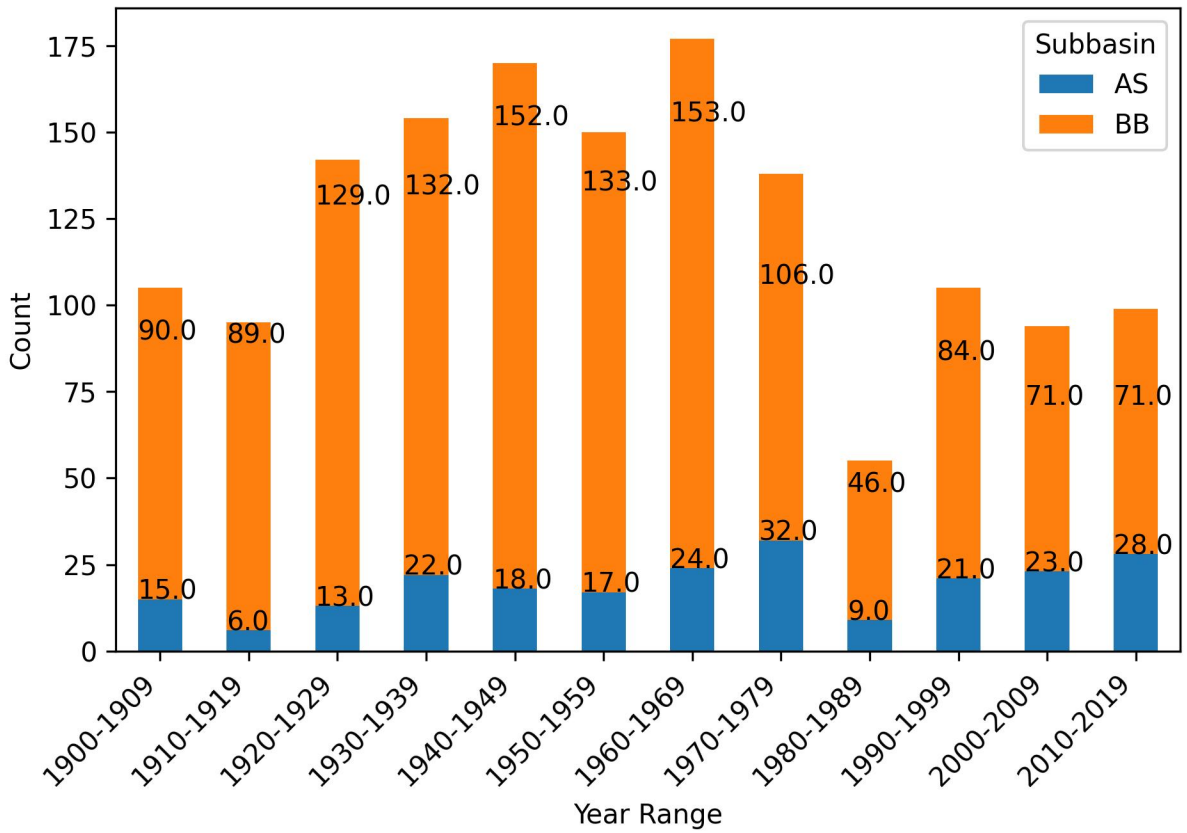

Fig 1.1 : Cyclones from 1900 to 2023 in Arabian Sea (AS) Bay of Bengal(BB)

The above picture tells us how important it is to predict the cyclone track as it is very important to our country which get 5 to 7 cyclones every year the above data was retrieved from the data IBTrACS which is also used for the prediction of the tracks in this project.

1.2.2 Cyclone track prediction

Cyclone track prediction involves forecasting the future latitude and longitude coordinates of a cyclone's path. This process is crucial for minimizing the potential impact of cyclones on lives and property. By analyzing various meteorological factors such as wind speed, air pressure, and temperature, along with historical cyclone data, predictive models can estimate the trajectory of cyclones over time. These models use sophisticated algorithms, often based on deep learning techniques, to interpret complex patterns and make accurate predictions. The predicted latitude and longitude coordinates enable authorities

to issue timely warnings and implement appropriate disaster preparedness measures, ultimately helping to mitigate the adverse effects of cyclones.

1.2.3 Deep learning

Deep learning is a type of machine learning that uses deep neural networks to make smart decisions, kind of like how our brains work. These networks have lots of layers, and they're trained on big sets of data to figure out patterns, make predictions, and help with decision-making.

Imagine a simple prediction like guessing if it's going to rain tomorrow. A basic system might just look at one or two things, like the temperature and the clouds. But with deep learning, we can look at a whole bunch of factors, like temperature, humidity, wind speed, and past weather patterns. This helps us make much more accurate predictions.

These deep learning systems are behind lots of things we use every day, like voice-controlled devices, fraud detection for credit cards, and even self-driving cars. They're always getting smarter because they're constantly learning from new data, just like we do when we experience new things.

1.2.3 Importance of Deep learning models on the dataset

Implementing deep learning techniques on the IBTrACS dataset for predicting cyclone tracks brings significant advantages, especially when considering Indian cyclones, where such methods haven't been extensively explored in existing literature. By leveraging deep learning, we enhance our ability to forecast cyclone trajectories with greater accuracy and reliability. Traditional models like CNN, LSTM, and MLP have shown promise, but they may struggle to capture the intricate patterns and relationships within the complex cyclone data. However, hybrid models like CNN-LSTM and CNN-GRU offer a more robust solution by combining the strengths of convolutional and recurrent neural networks.

These hybrid models excel at processing sequential and spatial data simultaneously, making them ideal for predicting cyclone tracks defined by latitude and longitude coordinates. CNN-LSTM integrates convolutional layers to extract spatial features from the data and LSTM layers to capture temporal dependencies, allowing for a comprehensive understanding of cyclone behavior over time. Similarly, CNN-GRU combines convolutional layers with gated recurrent units (GRU), offering efficient memory utilization and improved gradient flow during training.

By comparing the performance of simple models with hybrid architectures, we can evaluate the effectiveness of deep learning techniques in cyclone track prediction. The CNN-LSTM and CNN-GRU models are expected to outperform

traditional models by preserving spatial correlations and capturing long-term dependencies inherent in cyclone data. This enhanced predictive capability enables better preparedness and response to cyclone events, ultimately minimizing the potential impact on communities and infrastructure.

Overall, using deep learning techniques on the IBTrACS dataset opens up new possibilities for advancing cyclone track prediction, particularly for Indian cyclones where such approaches have been underexplored. Through rigorous comparison and evaluation, we can harness the full potential of hybrid models to achieve more accurate and reliable predictions of cyclone tracks, contributing to improved disaster management and mitigation efforts.

1.2.3 CNN

A Convolutional Neural Network (CNN) is a type of deep learning model specifically designed for processing and analyzing visual data, such as images. It is inspired by the visual cortex of the human brain and is particularly effective in tasks involving spatial relationships within the data. In a CNN, the input data is passed through a series of layers, each designed to extract different features from the input. The basic components of a CNN include convolutional layers, pooling layers, and fully connected layers.

In the project, CNN model is used to predict latitude and longitude coordinates of cyclone tracks. The model architecture consists of four layers:

Convolutional Layer (Conv1D): This layer applies 64 filters of size 3 to the input data, using the ReLU activation function to introduce non-linearity.

MaxPooling1D Layer: This layer performs max pooling with a pool size of 2, reducing the dimensionality of the feature maps.

Convolutional Layer (Conv1D): Similar to the first convolutional layer, this layer applies 32 filters of size 3 with ReLU activation.

Flatten Layer: This layer flattens the output from the previous layers into a one-dimensional vector.

Dense Layers: Two dense layers with 100 and 50 neurons, respectively, followed by an output layer with a single neuron. These layers perform regression to predict the latitude and longitude coordinates.

The model is trained using the Mean Squared Error (MSE) loss function and the Adam optimizer.

1.2.4 LSTM

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) architecture designed to capture long-term dependencies and relationships within sequential data. Unlike traditional RNNs, LSTM networks have specialized memory cells that can maintain information over extended time intervals, making them well-suited for tasks involving time series or sequential data.

In the project, LSTM models are used to predict latitude and longitude coordinates of cyclone tracks. The LSTM architecture consists of several key components:

LSTM Layers: LSTM layers contain memory cells that allow the network to retain information over multiple time steps. Each LSTM unit has three gates—input gate, forget gate, and output gate—that control the flow of information and regulate the cell state. The activation function (in this case, ReLU) determines how the information is processed within each LSTM unit.

Return Sequences: The `return_sequences=True` parameter in the first LSTM layer indicates that it should return the full sequence of output values rather than just the output at the last time step. This is important when stacking multiple LSTM layers, as it allows subsequent layers to receive the entire sequence of intermediate outputs.

Dense Layer: A fully connected dense layer is added after the LSTM layers to perform the final regression task. This layer maps the features extracted by the LSTM layers to the predicted latitude and longitude coordinates.

The model is trained using the Mean Squared Error (MSE) loss function and the Adam optimizer. After training, the model is evaluated on a separate test set to assess its performance in predicting cyclone track coordinates. The test loss indicates the degree of deviation between the predicted and actual coordinates, with lower values indicating better predictive accuracy.

1.2.5 MLP

A Multilayer Perceptron (MLP) is a type of artificial neural network composed of multiple layers of nodes (or neurons) arranged in a feedforward manner. Each node in a layer is connected to every node in the subsequent layer, forming a densely connected network. MLPs are widely used for regression and classification tasks where the input data does not have a sequential or temporal structure.

In the project, MLP models are employed to predict latitude and longitude coordinates of cyclone tracks. The architecture of an MLP model consists of the following components:

Input Layer: The input layer consists of nodes representing the input features. Each feature is connected to every node in the first hidden layer.

Hidden Layers: The hidden layers are composed of densely connected nodes that perform nonlinear transformations on the input data. In this example, the MLP model includes multiple hidden layers (four in this case), each with a varying number of nodes. The activation function used in the hidden layers (ReLU) introduces nonlinearity to the model, enabling it to learn complex patterns in the data.

Output Layer: The output layer consists of a single node in this regression task, as the goal is to predict a continuous value (latitude or longitude). The output node produces the predicted value based on the transformations applied by the preceding hidden layers.

The model is trained using the Mean Squared Error (MSE) loss function and the Adam optimizer. After training, the model is evaluated on a separate test set to assess its performance in predicting cyclone track coordinates. The test loss indicates the degree of deviation between the predicted and actual coordinates, with lower values indicating better predictive accuracy.

1.2.6 CNN-LSTM

A Hybrid CNN-LSTM model combines the strengths of Convolutional Neural Networks (CNNs) and Long Short-Term Memory networks (LSTMs) to effectively capture both spatial and temporal dependencies in sequential data, such as time-series data.

In this model, the Conv1D layers of the CNN are initially used to extract spatial features from the input data, which in this case represents the cyclone track features. These layers apply convolutional filters across the input data to detect patterns and features, followed by max-pooling layers to downsample the spatial dimensions. After the CNN layers, LSTM layers are employed to capture temporal dependencies and patterns in the sequential data. LSTMs are well-suited for modeling time-series data due to their ability to retain long-term dependencies and handle sequences of varying lengths.

The architecture of the Hybrid CNN-LSTM model for latitude and longitude prediction consists of the following components:

Input Layer: The input layer accepts the input data, which is reshaped to match the required input shape for the CNN layers.

Convolutional Layers (CNN): These layers consist of Conv1D filters followed by max-pooling layers. The Conv1D filters learn spatial features from the input data, while the max-pooling layers downsample the spatial dimensions to reduce computational complexity.

Long Short-Term Memory (LSTM) Layers: These layers capture temporal dependencies in the sequential data extracted by the CNN layers. The LSTM units process the sequential data and retain information over time, allowing the model to learn patterns in the cyclone track data.

Dense Output Layer: The final output layer produces the predicted latitude or longitude values based on the features learned by the preceding layers.

The model is trained using Mean Squared Error (MSE) loss and the Adam optimizer. After training, the model is evaluated on a separate test set to assess its performance in predicting cyclone track coordinates. The test loss indicates the degree of deviation between the predicted and actual coordinates, with lower values indicating better predictive accuracy.

1.2.7 CNN-GRU

A CNN-GRU model combines Convolutional Neural Networks (CNNs) and Gated Recurrent Units (GRUs) to effectively capture spatial features and temporal dependencies in sequential data, such as time-series data like cyclone tracks.

In this model:

Convolutional Layers (CNN): These layers apply convolutional filters to extract spatial features from the input data, which in this case represents the cyclone track features. The filters detect patterns and features in the spatial domain of the data, followed by max-pooling layers to downsample the spatial dimensions and reduce computational complexity.

Gated Recurrent Units (GRUs): GRUs are a type of recurrent neural network (RNN) that can capture temporal dependencies in sequential data. They utilize gating mechanisms to selectively update and reset their internal states, allowing them to retain relevant information over time. In the CNN-GRU model, GRU layers are used to capture temporal patterns in the cyclone track data extracted by the CNN layers.

Dense Layers: After the convolutional and GRU layers, dense layers are employed to perform feature aggregation and produce the final output. These layers integrate the spatial and temporal features learned by the preceding layers and map them to the output space, predicting the latitude and longitude coordinates of the cyclone tracks.

The architecture of the CNN-GRU model for latitude and longitude prediction consists of the following components:

a) Convolutional layers for spatial feature extraction.
b) Max-pooling layers for spatial dimension reduction.
c) GRU layers for capturing temporal dependencies.
d) Dense layers for feature aggregation and output prediction

The model is trained using Mean Squared Error (MSE) loss and the Adam optimizer. After training, it is evaluated on a separate test set to assess its performance in predicting cyclone track coordinates, with lower test loss indicating better predictive accuracy.

1.2.8 Evaluation metrics

As we are using Deep learning to predict the latitude and longitude models we are going to use RMSE and Test loss as the metrics for comparing the best model and then comparing them on the actual map would give us more understanding so the below given are the formulas for RMSE and Test Loss

1. Root Mean Squared Error (RMSE) is a commonly used metric to evaluate the performance of a predictive model, especially in regression tasks like predicting latitude and longitude. It measures the average difference between the predicted values and the actual values, taking into account both the magnitude and direction of the errors.

$$RMSE = \sqrt{1/n * \sum_{i=1}^{n} (y - yp)\text{^}2}$$ Formula 1

where:
- n is the number of data points.
- y is the actual value.
- yp is the predicted value.

2. Test loss is typically calculated using the loss function chosen during model compilation, such as Mean Squared Error (MSE) or another appropriate loss function. For example, in the code provided, the models are compiled using the mean squared error loss function (loss='mse'). The test loss is computed during model evaluation using this loss function.

$$Test\_loss = 1/n \sum_{i=1}^{n} (y - yp)\text{^}2$$ Formula 2

where:
- n is the number of data points.
- y is the actual value.
- yp is the predicted value.

3. Comparing Predicted Latitude and Longitude on a Map:In this evaluation method, the predicted latitude and longitude coordinates generated by the model are plotted on a map alongside the actual coordinates. This visual comparison allows for a qualitative assessment of the model's performance in predicting geographical positions.

a) **Accuracy Assessment:** By visually inspecting the plotted points, one can assess how closely the predicted positions align with the actual positions. Close alignment indicates higher accuracy, while significant deviations suggest areas where the model may need improvement.
b) **Performance Visualization:** Plotting the predicted and actual positions on a map provides an intuitive way to understand how well the model captures spatial relationships and patterns. It allows stakeholders to assess the model's ability to generalize to unseen data and make reliable predictions in real-world scenarios.
c) **Decision Support:** Insights gained from comparing predicted and actual positions can inform decision-making processes, such as resource allocation, risk assessment, or route planning, in applications where accurate geographical predictions are crucial.

1.3 Statement

Creating a system that accurately predicts cyclone tracks using machine learning faces challenges like data quality and imbalanced datasets. We're working to overcome these hurdles by developing strong algorithms. Our goal is to use biomarkers and other data to find cyclone eyes in satellite images reliably. We want these models to work well with new data and make ethical decisions about cyclone tracking. This system could improve weather forecasts and help predict cyclone paths better.

1.4 Motivation

Automatic cyclone track prediction systems are vital tools for weather forecasting and disaster management. These systems help identify cyclone eyes, which indicate the intensity and behavior of cyclones. By accurately tracking these features in satellite images, we can better understand cyclone dynamics, predict storm paths, and issue timely warnings. This technology enables us to reduce risks, save lives, and lessen the impact of severe weather events. Automatic cyclone track prediction systems are crucial for improving early warning systems and taking proactive steps to protect communities from the devastating effects of storms.

1.5 Research challenges

Data Preprocessing: Cleaning and preprocessing the cyclone data to remove noise, handle missing values, and standardize features for effective model training.

Feature Selection and Engineering: Identifying the most relevant features from the provided columns and exploring additional features that could improve prediction accuracy.

Model Selection and Architecture: Choosing appropriate deep learning architectures (CNN-GRU, CNN-LSTM) and optimizing their configurations for effective cyclone trajectory prediction.

Training Data Availability: Ensuring an adequate amount of labeled training data for model training, which may be challenging due to the scarcity of cyclone events and the need for historical data.

Temporal Dynamics: Addressing the temporal nature of cyclone data, such as changes in storm attributes over time, and incorporating time-series modeling techniques into the deep learning models.

Model Interpretability: Developing methods to interpret the predictions of deep learning models and understand the factors contributing to cyclone trajectory forecasts.

Generalization and Transfer Learning: Investigating techniques for model generalization across different cyclone datasets and exploring transfer learning approaches to leverage knowledge from pre-trained models.

Model Evaluation Metrics: Defining appropriate evaluation metrics to assess the performance of the deep learning models accurately, considering factors such as RMSE, Test_Loss and Comparing of the latitude and longitude.

Real-time Processing: Addressing the computational challenges associated with real-time processing of cyclone data, especially considering the resource constraints in deployment environments.

1.6 Literature review

The literature review in the project on predicting cyclone tracks using various techniques serves as the foundational background and introduction to existing research and studies related to the topic.

Y. Zhang, R. Chandra **[1]** Cyclone Track Prediction with Matrix Neural Networks: This study explores cyclone trajectory prediction using Matrix Neural Networks (MNNs) in the South Indian Ocean. Unlike recurrent neural network architectures, MNNs can handle spatial correlations in the data effectively, preserving spatial information without loss. Results demonstrate superior prediction performance of MNNs compared to prominent recurrent neural network architectures due to their ability to maintain spatial correlation.

S. Kumar, K. Biswas and A. K. Pandey **[2]** Track Prediction of Tropical Cyclones Using Long Short-Term Memory Network: Proposed is a Long Short-Term Memory network for accurate tropical cyclone track prediction in the

Atlantic and north Indian oceans. Utilizing variables like central pressure and wind speed, the model outperforms existing methods in mean absolute error and prediction timespan. Unlike traditional methods, it uses a GridID output obtained through grid functions, improving efficiency and accuracy.

Fang, W **[3]** Combining Support Vector Machine and Polynomial Regression to Predict Tropical Cyclone Track: This research integrates Support Vector Machine (SVM) and polynomial regression for cyclone tracking. It detects cyclone centers with SVM and predicts tracks using polynomial regression. The method is tested on five cyclones affecting Myanmar, providing accurate predictions by combining different models.

Giffard-Roisin S **[4]** Tropical Cyclone Track Forecasting Using Fused Deep Learning From Aligned Reanalysis Data: The study proposes a neural network model fused with reanalysis data for tropical cyclone track forecasting. The model integrates past trajectory data and atmospheric images, offering high-precision forecasts with reduced computational demands. It outperforms traditional forecast models and provides real-time forecasting capabilities.

Wang **[5]** Forecasting Tropical Cyclone Tracks in the Northwestern Pacific Based on a Deep-Learning Model: This research employs deep-learning technology to forecast tropical cyclone tracks in the northwestern Pacific. Utilizing recurrent neural networks and convolutional neural networks, the proposed GRU_CNN model outperforms other models in short-term forecasting. By incorporating environmental factors, it achieves improved accuracy, making it suitable for disaster prevention.

Lian, J.**[6]** A Novel Deep Learning Approach for Tropical Cyclone Track Prediction Based on Auto-Encoder and Gated Recurrent Unit Networks: The study presents a data-driven approach combining auto-encoder and GRU models for tropical cyclone track prediction. The proposed model outperforms existing methods, showing significant improvements in forecast accuracy and efficiency. By leveraging historical data and meteorological attributes, it enhances prediction capabilities, aiding in disaster mitigation.

Chen, R. **[7]** Machine Learning in Tropical Cyclone Forecast Modeling: A Review: This review discusses the application of machine learning in tropical cyclone forecasting. It highlights the potential of machine learning to address forecast challenges and improve accuracy. By summarizing recent progress and

successful cases, the review emphasizes the role of machine learning in enhancing numerical forecast models and disaster preparedness.

Tong B.**[8]** Short-term Prediction of the Intensity and Track of Tropical Cyclone via ConvLSTM Model: This study proposes a Convolutional Long and Short Term Memory (ConvLSTM) network for short-term tropical cyclone prediction. Unlike traditional methods, the model efficiently extracts temporal and parameter-related correlations from input data, achieving high prediction accuracy and stability. With its computational efficiency, it offers practical applications in engineering.

Akila Rajini Selvaraj **[9]** An Optimal Model Using Single-Dimensional CAE-IRNN Based SPOA for Cyclone Track Prediction: The research introduces a three-step approach for predicting cyclone tracks using single-dimensional CAE-IRNN models. By preprocessing meteorological data and optimizing feature selection, the proposed method enhances prediction accuracy and efficiency. Comparative analysis demonstrates the superiority of the proposed approach, offering significant advancements in cyclone prediction.

Pérez-Alarcón, Albenis & Coll-Hidalgo**[10]** CyTRACK: An Open-Source Python Toolbox for Detecting and Tracking Cyclones: This work presents CyTRACK, a comprehensive Python toolbox for cyclone detection and tracking in model and reanalysis datasets. Utilizing critical cyclone centers and threshold parameters, CyTRACK captures cyclone variability and spatial distribution accurately. The toolbox facilitates sensitivity analysis and climatological studies, providing valuable insights into cyclone features.

Tiantian Wu**[11]** A New and Efficient Method for Tropical Cyclone Detection and Tracking in Gridded Datasets: The study proposes a novel method based on mathematical morphology for detecting and tracking tropical cyclones in gridded reanalysis data. By transforming vorticity fields into binary images and applying connected component labeling, the method accurately reproduces cyclone tracks with high efficiency. Evaluation against observed tracks demonstrates its effectiveness in tracking cyclones.

Enz B**[12]** Use of Threshold Parameter Variation for Tropical Cyclone Tracking: This study introduces a tracking algorithm employing multiple threshold parameter combinations for tropical cyclone detection. By combining weaker and stronger thresholds, the algorithm accurately tracks cyclones within model data,

demonstrating adequate estimation of cyclone energy. The approach offers a balanced assessment of model performance and cyclone behavior.

Paul, Tushar & Ranganayakulu **[13]** A Study of Cyclone Tracking Mechanism Using Deep Learning Techniques: The research presents a deep learning-based approach for automated tropical cyclone detection from satellite photos. Through a three-phase architecture incorporating convolutional neural networks and Bayesian optimization, the method achieves high specificity, precision, and accuracy in cyclone detection, providing advanced warning capabilities.

Wang, Yuanfei & Zhang **[14]** Back Propagation Neural Network for Tropical Cyclone Track Forecast: This study employs a Back Propagation neural network for tropical cyclone track prediction. Trained on historical track data, the model demonstrates effective performance in cyclone prediction, contributing to improved forecasting accuracy and reliability.

Kumar, Sandeep & Biswas **[15]** Track Prediction of Tropical Cyclones Using Long Short-Term Memory Network: This paper introduces a Long Short-Term Memory network for precise tropical cyclone track prediction. By utilizing variables like central pressure and wind speed, the model outperforms existing methods in mean absolute error and prediction timespan, offering enhanced accuracy and efficiency in track forecasting.
Here's a tabulated classification of the types of research papers based on their focus and methodologies:

Table 1.1 : Literature review

Paper title	Journal / Conference	Description
Cyclone Track Prediction with Matrix Neural Networks[1]	2018 International Joint Conference on Neural Networks(IEEE)	Matrix neural networks leverage spatial correlations without vectorization, making them adept for cyclone trajectory prediction, especially in the South Indian Ocean.

Track Prediction of Tropical Cyclones Using Long Short-Term Memory Network[2]	2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)(IEEE)	A Long Short-Term Memory (LSTM) network-based Recurrent Neural Network accurately predicts cyclone tracks, surpassing existing models in performance metrics.
Combining Support Vector Machine and Polynomial Regressing to Predict Tropical Cyclone Track[3]	2021 IEEE 3rd Global Conference on Life Sciences and Technologies	This paper proposes a Tropical Cyclone tracking system merging SVM and polynomial regression to predict cyclone trajectories accurately.
Tropical Cyclone Track Forecasting Using Fused Deep Learning From Aligned Reanalysis Data[4]	Data-driven Climate Sciences,a section of the journal Frontiers in Big Data(2020)(Frontiers)	This paper proposes a neural network model combining past trajectory data and atmospheric images for accurate tropical cyclone tracking.
Forecasting tropical cyclone tracks in the northwestern Pacific based on a deep-learning model[5]	2023 - Geoscientific Model Development EGU	This paper employs deep learning to forecast TC tracks in the northwestern Pacific, with GRU_CNN outperforming other models.
A Novel Deep Learning Approach for Tropical Cyclone Track Prediction Based on Auto-Encoder and Gated Recurrent Unit Networks[6]	2020 - IEEE,Environmental Modelling and Software	This paper introduces a novel data-driven approach using AE and GRU models for forecasting tropical cyclone landing locations.
Machine Learning in Tropical Cyclone Forecast Modeling: A Review[7]	2020 - IEEE , 19th International Conference on Geoinformatics	This review highlights the potential of machine learning in improving tropical cyclone forecasts while addressing ongoing challenges and opportunities.
Short-term prediction of the intensity and track of tropical cyclone via ConvLSTM model [8]	2022 - Journal of Wind Engineering & Industrial Aerodynamics (Elsevier)	This paper introduces a ConvLSTM network for short-term TC prediction in the Northwest Pacific basin, offering accurate forecasts efficiently.

An optimal model using single-dimensional CAE-IRNN based SPOA for cyclone track prediction[9]	2023 - Expert Systems With Applications (Elsevier)	.The research introduces a three-step approach for predicting cyclone tracks using single-dimensional CAE-IRNN models
CyTRACK: An open-source and user-friendly python toolbox for detecting and tracking cyclones[10]	2024 - Environmental Modelling and Software (Elsevier)	This paper introduces CyTRACK, a Python toolbox for cyclone detection and tracking, validated against reanalysis data and best-track archives.
A new and efficient method for tropical cyclone detection and tracking in gridded datasets[11]	2023 - Weather and Climate Extremes (Elsevier)	A novel method based on mathematical morphology is proposed to detect and track tropical cyclones using reanalysis data effectively.
Use of threshold parameter variation for tropical cyclone tracking[12]	2023 - Geoscientific Model Development EGU	This study evaluates various threshold parameter combinations to objectively track tropical cyclones in numerical models, yielding reliable estimates of accumulated cyclone energy.
A Study Of Cyclone Tracking Mechanism Using Deep Learning Techniques[13]	2023 International Conference on System, Computation, Automation and Networking (ICSCAN)(IEEE)	The paper introduces a deep learning-based approach for automated tropical cyclone detection from satellite images, achieving high accuracy.
Back Propagation (BP)-Neural Network for Tropical Cyclone Track Forecast[14]	2011 19th International Conference on Geoinformatics(IEEE)	This study addresses the challenge of Tropical Cyclone track prediction using BP-neural networks, demonstrating promising results in simulation accuracy.
Track Prediction of Tropical Cyclones Using Long Short-Term Memory Network[15]	2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC) (IEEE)	The paper introduces a LSTM-based Recurrent Neural Network for accurate tropical cyclone track prediction, outperforming existing models.

In all these research papers there is no single paper who have used the cyclone track prediction on Indian cyclone data from IBTrACS so I have used various deep learning models and done the prediction of both latitude and longitude.

Chapter 2
Planning and system Specification

2.1 System Planning

The system goes through several steps. First, it prepares the data, getting it ready for analysis. Then, it splits the data into two parts: one for training the model and the other for testing its accuracy. During training, the model learns from the data to make predictions. After training, the system tests the model using the test data to see how well it performs on new, unseen information. This process helps ensure that the model can generalize well and make accurate predictions in real-world situations.

Table 2.1 : System planning

System planning	Dates
Literature review	Feb 22nd 2024
80% completion	March 25th 2024
100% implementation	April 10th 2024

2.2 Requirements

2.2.1 User Requirements

Ease of Use: Users expect a user-friendly interface and clear documentation to navigate the Colab-based cyclone track prediction system easily.

Computational Resources: Users require access to sufficient computational resources on Colab to train deep learning models effectively, including GPU or TPU support for faster processing.

Customization: Users need the ability to customize and fine-tune the deep learning models according to their specific requirements, such as adjusting hyperparameters or selecting different architectures.

Data Integration: The system should allow users to seamlessly integrate their cyclone data into the Colab environment for training and testing the models, with support for various data formats.

Performance Evaluation: Users expect tools for evaluating model performance, including metrics like RMSE, test loss, and visualizations of predicted cyclone tracks, to assess the effectiveness of the trained models.

2.2.2 Non-Functional Requirements

Performance: The system should handle large datasets efficiently and execute deep learning computations within reasonable timeframes, ensuring minimal latency and high responsiveness.

Scalability: It should support scaling to accommodate increased user demand and larger datasets without sacrificing performance or reliability.

Reliability: The system should maintain high availability and uptime, with robust error handling mechanisms to recover from failures gracefully.

Security: User data and model training processes must be secured through encryption, authentication, and authorization mechanisms to prevent unauthorized access or data breaches.

Maintainability: The codebase should be well-documented, modular, and adhere to best practices to facilitate easy maintenance, updates, and enhancements over time.

2.3 System Requirements

2.3.1 Hardware Requirements

A stable internet connection to access Google Colab.

A laptop or desktop computer with a web browser to access and interact with the Colab environment.

Sufficient RAM and CPU power for running deep learning algorithms efficiently.

Although Colab provides access to GPUs and TPUs for faster computation, it's not mandatory for the project implementation.

2.3.2 Software Requirements

A Google account to access Google Colab.

Basic knowledge of Python programming language and deep learning frameworks such as TensorFlow or PyTorch.

Access to the required datasets for training and testing the models.

Installation of necessary Python libraries like TensorFlow, Keras, scikit-learn, and pandas for data manipulation, model building, and evaluation.

Familiarity with Jupyter notebooks for writing and executing Python code in Colab environment.

Chapter 3
System Design

In this section I have done system design explanation and the system architecture of the whole system .

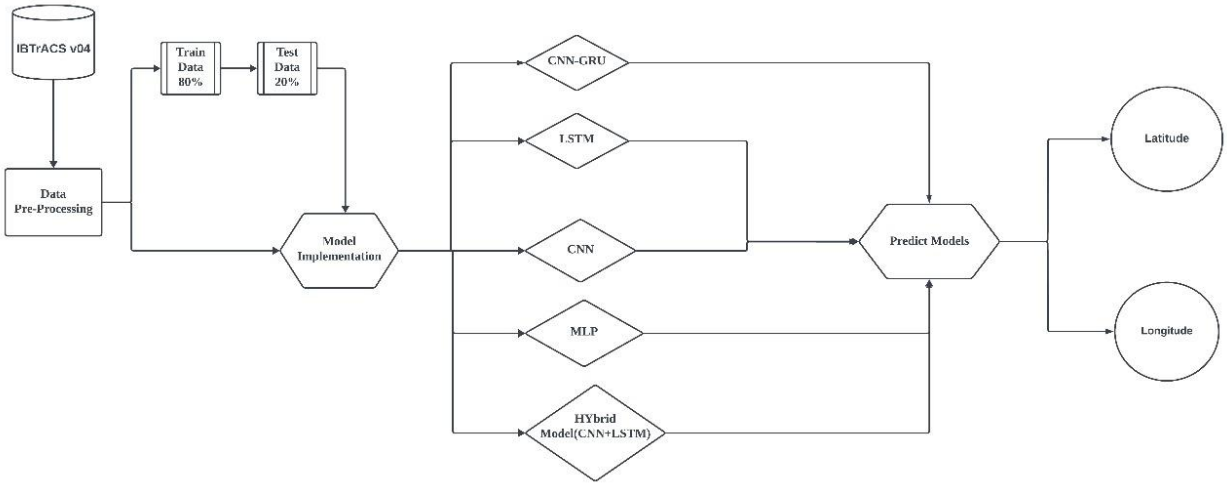

Figure 3.1 : System architecture

3.1 Overall system architecture and methodology:

a. Data Pre-processing:

Raw data is loaded from the CSV file using pandas.Columns with irrelevant or redundant information are removed.Data is filtered to include only relevant observations and to ensure data integrity.Temporal and spatial features are extracted from the data to provide additional context for model training.

b. Data Splitting:

The pre-processed data is split into training and testing sets.A portion of the data is reserved for training the models, while the remaining portion is used for evaluating model performance.

c. Model Training:

Deep learning models such as Convolutional Neural Networks (CNN), Long Short-Term Memory (LSTM), Multi-Layer Perceptron (MLP), CNN-GRU, and CNN-LSTM are implemented using TensorFlow/Keras. Each model architecture is defined with appropriate layers for predicting cyclone latitude and longitude.The models are compiled with suitable loss functions and optimizers for training.Training data is fed into the models iteratively for a fixed number of epochs to update model parameters and minimize the loss function.

d. Model Evaluation:

The trained models are evaluated using metrics such as Root Mean Squared Error (RMSE) and Test Loss.RMSE is calculated to measure the average deviation between predicted and actual cyclone coordinates.Test loss, computed using the mean squared error, provides an overall measure of model performance on unseen data.Additionally, the predicted latitude and longitude values are compared to actual values and visualized on a map to assess the accuracy of the predictions.

e. Comparison of Models:

The performance of different deep learning models (CNN, LSTM, MLP, CNN-GRU, CNN-LSTM) is compared based on their respective RMSE scores and test losses.Insights are drawn from the comparison to identify the most effective model for cyclone track prediction.Factors such as computational efficiency, model complexity, and interpretability are considered in selecting the optimal model for the application.

By following this system design, the cyclone track prediction system can effectively preprocess data, train deep learning models, evaluate their performance, and make informed decisions regarding model selection for accurate prediction of cyclone tracks.

3.2 Data Preprocessing:

The preprocessing techniques used in the project along with their corresponding formulas:

a. Filtering Columns:

Objective: Remove irrelevant or redundant columns from the dataset.

Formula: No specific formula, it involves selecting columns based on certain criteria like relevance to the task, data quality, or domain knowledge.

b. Filtering Rows Based on Criteria:

Objective: Exclude rows that do not meet specific criteria, such as data integrity or relevance.

Formula: No specific formula, but it typically involves applying conditional statements to filter rows based on certain column values.

c. Temporal Feature Extraction:

Objective: Extract temporal features from datetime columns to provide additional context for model training.

Formula:Hour of the Day: Extract the hour component from the datetime column.Day of the Week: Extract the day of the week (0 for

Monday, 6 for Sunday) from the datetime column.Month: Extract the month from the datetime column.Season: Divide the months into seasons (e.g., 0 for winter, 1 for spring, etc.).

d. Spatial Feature Calculation:

Objective: Compute spatial features such as angles and distances between consecutive data points.

Formulas:

Angle Calculation: Calculate the angle between two consecutive points using their latitude and longitude coordinates. This can be done using trigonometric functions like atan2.

Angle Calculation Formula:

angle=atan2(sin(Δlon)*coslat2),cos(lat1)*sin(lat2)−sin(lat1)*cos(lat2)*cos(Δlon))

angle=degrees(angle)angle=degrees(angle)

angle=(angle+360)mod 360angle=(angle+360)mod360

Haversine Distance Calculation: Compute the great-circle distance between two points on the Earth's surface given their latitude and longitude using the Haversine formula.

Haversine Distance Calculation Formula:

R = 6371.0 (Radius of the Earth in kilometers)

Δlon = lon2−lon1

Δlon = lon2 −lon1

a = sin^2(Δlat/2)+cos(lat1)*cos(lat2)*sin^2(Δlon/2)

c = 2*a*tan2(sqrt(*a)*,sqrt(1−*a)*)

Distance = $R*c$

These preprocessing techniques help in cleaning and enhancing the dataset, making it suitable for training machine learning models. They provide additional contextual information and ensure data integrity, which is crucial for accurate predictions.

3.3 Python libraries used

pandas: Provides data structures and data analysis tools for working with structured data.

math: Python's built-in library for mathematical operations and functions.

datetime: Offers classes for manipulating dates and times in Python.

matplotlib.pyplot: A plotting library for creating visualizations in Python.

sklearn.model_selection: Provides functions for splitting data into train and test sets, and cross-validation.

sklearn.linear_model.LinearRegression: Implements linear regression models for predictive modeling tasks.

sklearn.metrics: Contains functions for evaluating model performance, such as mean squared error and accuracy score.
sklearn.preprocessing.StandardScaler: Used for standardizing features by removing the mean and scaling to unit variance.
tensorflow.keras.models.Sequential: A linear stack of layers used to build neural network models in TensorFlow.
tensorflow.keras.layers: Contains various types of layers for building neural networks, such as Conv1D, MaxPooling1D, Dense, LSTM, and GRU.
tensorflow: An open-source machine learning library for high-performance numerical computation.
numpy: Provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions.
cartopy.crs: A module for cartographic projection definitions and utilities.
cartopy.feature.NaturalEarthFeature: Provides access to Natural Earth datasets for cartographic features.

Chapter 4
Implementation of the system

In this section, I will detail the implementation of the cyclone track prediction system in a step-by-step manner, following the methodology and algorithms outlined in the project plan. The project is divided into two main phases. First we perform data preprocessing and feature engineering on the cyclone dataset. This involves tasks such as cleaning the data, extracting relevant features, and standardizing the input variables. Additionally, we calculate additional features such as angles and distances between consecutive data points to enrich the dataset. Next we develop and train deep learning models for cyclone track prediction using the preprocessed dataset. Specifically, we utilize convolutional neural networks (CNN), long short-term memory networks (LSTM), multilayer perceptrons (MLP), as well as hybrid models like CNN-GRU and CNN-LSTM. Each model is trained and evaluated using rigorous methodologies, including data splitting, model compilation, and performance evaluation using metrics such as root mean square error (RMSE) and test loss. The trained models are then integrated into the cyclone track prediction system for real-time or batch prediction tasks. Finally, I compare the data predicted with the original tracking by in the data for the michuang cyclone.

4.1 Data preprocessing

The below are the data preprocessing techniques that I have used in this project to fit for the project.

Column Name Standardization: Imported the dataset using pandas from a CSV file.Combined column names with their descriptions to ensure clarity and consistency in the dataset.

Data Filtering: Filtered out rows related to cyclones in the North Indian Ocean basin ('NI'). Removed unnecessary columns by deleting prefixes associated with specific data sources.

Data Cleaning: Removed rows with a 'SEASON_Year' value below 1900 to ensure data consistency and reliability.

Angle Calculation: Defined a function to calculate angles between consecutive latitude and longitude coordinates.

Applied this function to compute angles and append them as a new column ('ANGLE') to the filtered DataFrame.

Angle Calculation Formula:

angle=atan2(sin(Δlon)*coslat2),cos(lat1)*sin(lat2)−sin(lat1)*cos(lat2)*cos(Δlon))
angle=degrees(angle)angle=degrees(angle)
angle=(angle+360)mod 360angle=(angle+360)mod360

Distance Calculation: Implemented the Haversine formula to compute distances between consecutive latitude and longitude coordinates.Applied this function to calculate distances and added them as a new column ('DISTANCE_km') to the DataFrame.

Haversine Distance Calculation Formula:

R = 6371.0 (Radius of the Earth in kilometers)

Δlon = lon2−lon1

Δlon = lon2 −lon1

a = sin^2(Δlat/2)+cos(lat1)*cos(lat2)*sin^2(Δlon/2)

c = 2*a*tan2(sqrt(*a)*,sqrt(1−*a)*)

Distance = *R*c*

Sorting and Time Difference Calculation: Converted the 'ISO_TIME_' column to datetime objects.Sorted the DataFrame by cyclone name ('SID_') and timestamp ('ISO_TIME_').Calculated the time difference in hours between each timestamp and the first instance of each cyclone event, storing the result in a new column ('TIME_DIFFERENCE_hours').

Saving Preprocessed Data: Saved the preprocessed DataFrame to a CSV file named 'processed_data.csv' for future use.

Extracting First Instances of Cyclones: Grouped the DataFrame by cyclone name ('SID_') and selected the first instance of each cyclone event to extract essential information about the cyclone's initial state.

Temporal Feature Extraction: Converted the 'ISO_TIME_' column to datetime objects with a specified format.Extracted temporal features such as hour of the day, day of the week, month, and season from the 'ISO_TIME_' column.Calculated the time elapsed since the start of each cyclone event and stored it as 'Time_Since_Start' in hours.

Numeric Data Conversion: Converted specific columns to numeric data types to ensure consistency and compatibility for further analysis.Replaced non-numeric values with zeros and ensured all numeric columns were of the float data type for numerical computations.

IST2LAND_km	LANDFALL_km	STORM_SPEED_kts	STORM_DIR_degrees	ANGLE	DISTANCE_km	TIME_DIFFERENCE_hours
214	191	4	289	0.000000	0.000000	0.0
191	160	5	289	69.295411	22.281091	3.0
160	130	6	289	67.846901	31.660107	6.0
130	115	6	289	66.187321	32.789848	9.0
100	84	6	289	64.527110	31.007647	12.0

Figure 4.1 :Implementation of angle, distance and time difference

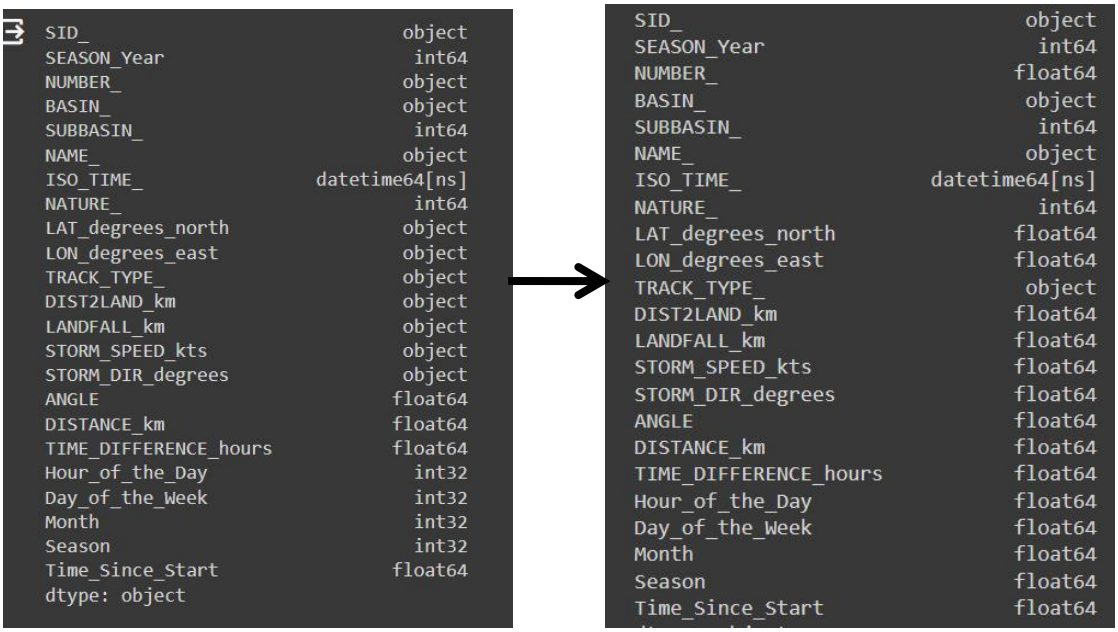

Figure 4.2 : Numeric Data Conversion

4.2 Implementation of deep learning models

At first I have trained three traditional models which are CNN, LSTM, MLP models and then developed two hybrid models which are CNN-LSTM and CNN-GRU models which are far more superior from normal models which will be discussed in results and discussion.

4.2.1 CNN Implementation

This is a Convolutional Neural Network (CNN) model implemented using TensorFlow's Sequential API for predicting cyclone attributes. Here's a breakdown:

Input Layer:

Utilizes a 1D convolutional layer with 64 filters and a kernel size of 3.Applies the ReLU activation function.Input shape is determined by the shape of the training data (`X_train_cnn`).

Pooling Layer:

Performs max pooling with a pool size of 2 to reduce spatial dimensions.

Additional Convolutional and Pooling Layers:

Followed by another convolutional layer with 32 filters and a kernel size of 3, followed by max pooling.These layers extract hierarchical features from the input data.

Flattening Layer:

Flattens the output from the convolutional layers into a one-dimensional array.

Dense Layers:

Consists of two fully connected (dense) layers with 100 and 50 neurons, respectively.Utilizes ReLU activation function for introducing non-linearity.

Output Layer:

A single neuron output layer responsible for predicting a numerical value.No activation function is applied, indicating it's a regression problem.

This architecture combines convolutional layers for feature extraction and fully connected layers for classification or regression tasks. It's suitable for processing sequential data like time series, which is relevant for cyclone attribute prediction.

Model:

```
model_cnn = Sequential([
    Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(X_train_cnn.shape[1], X_train_cnn.shape[2])),
    MaxPooling1D(pool_size=2),
    Conv1D(filters=32, kernel_size=3, activation='relu'),
    MaxPooling1D(pool_size=2),
    Flatten(),
    Dense(100, activation='relu'),
    Dense(50, activation='relu'),
    Dense(1)])
```

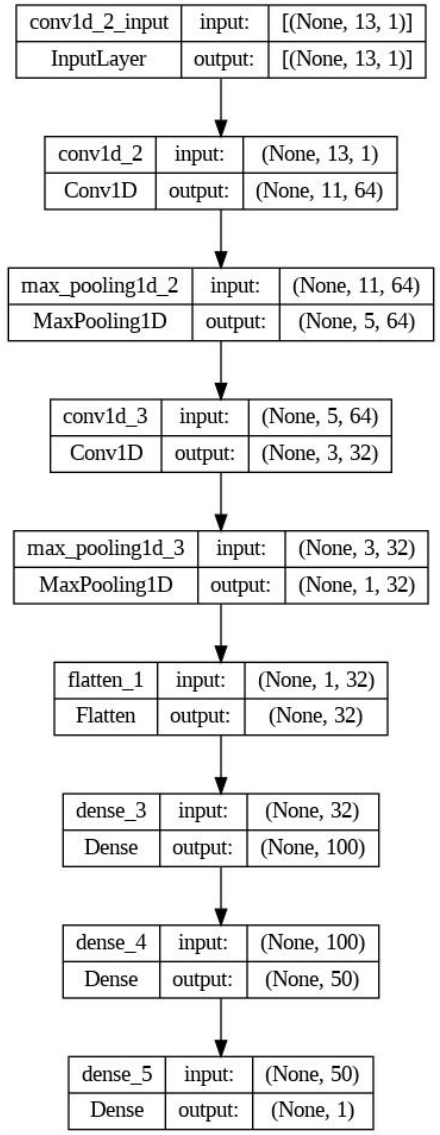

Figure 4.3 : CNN Model

```
Epoch 1/50
1176/1176 [==============================] - 5s 3ms/step - loss: 17.2590
Epoch 2/50
1176/1176 [==============================] - 3s 3ms/step - loss: 0.9897
Epoch 3/50
1176/1176 [==============================] - 4s 3ms/step - loss: 0.3501
Epoch 4/50
1176/1176 [==============================] - 5s 4ms/step - loss: 0.2384
Epoch 5/50
1176/1176 [==============================] - 3s 3ms/step - loss: 0.1800
Epoch 6/50
1176/1176 [==============================] - 3s 3ms/step - loss: 0.1529
Epoch 7/50
1176/1176 [==============================] - 4s 3ms/step - loss: 0.1089
Epoch 8/50
1176/1176 [==============================] - 4s 3ms/step - loss: 0.1229
```

Figure 4.4 : Implementation of CNN

4.2.2 LSTM Implementation

Long Short-Term Memory (LSTM) neural network model for predicting latitude and longitude in a cyclone tracking:

Model Architecture:

The model consists of two LSTM layers followed by a Dense layer.
The first LSTM layer has 32 units, uses the ReLU activation function, and returns sequences.
The second LSTM layer has 16 units and uses the ReLU activation function.
The Dense layer outputs a single value, representing the predicted latitude.

Input Shape:

The input shape is specified as (X_train_cnn.shape[1], X_train_cnn.shape[2]), indicating the number of time steps and features in the input data.

Activation Function:

The ReLU activation function is used for both LSTM layers, promoting non-linearity in the model's learned representations.

Model Training:

The model is trained on the provided training data (X_train_cnn) to learn the temporal patterns and relationships in the cyclone dataset.

Evaluation:

The model's performance can be evaluated using metrics such as Mean Squared Error (MSE) or Root Mean Squared Error (RMSE) on a separate test dataset.

Model:

```
model_lon_lstm = Sequential([
        LSTM(32,activation='relu', return_sequences=True,
input_shape=(X_train_cnn.shape[1], X_train_cnn.shape[2])),
         LSTM(16, activation='relu'),
         Dense(1)
        ])
```

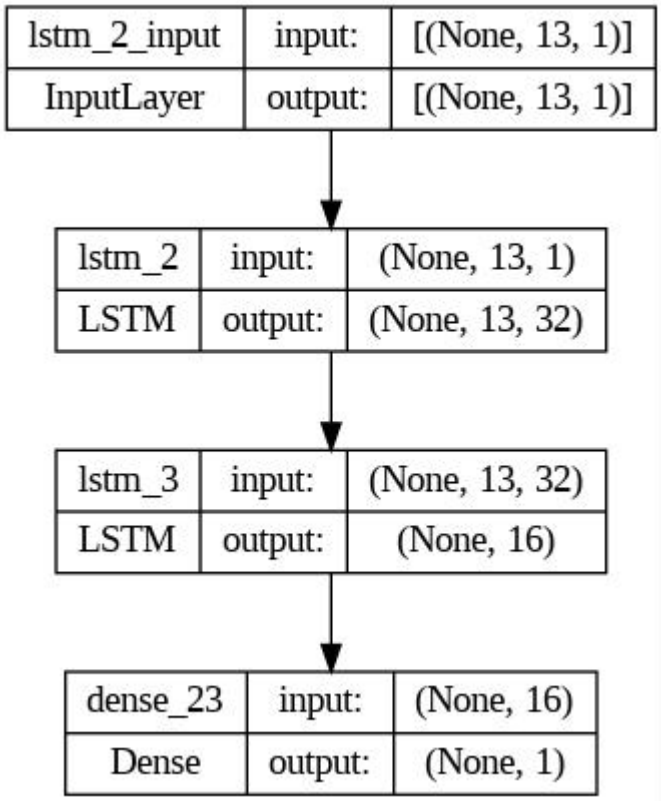

Figure 4.5 : LSTM Model

```
Epoch 1/50
1176/1176 [==============================] - 18s 13ms/step - loss: 34.6572
Epoch 2/50
1176/1176 [==============================] - 19s 16ms/step - loss: 1.7235
Epoch 3/50
1176/1176 [==============================] - 15s 13ms/step - loss: 0.2794
Epoch 4/50
1176/1176 [==============================] - 14s 11ms/step - loss: 0.1860
Epoch 5/50
1176/1176 [==============================] - 13s 11ms/step - loss: 0.1678
Epoch 6/50
1176/1176 [==============================] - 13s 11ms/step - loss: 0.1303
Epoch 7/50
1176/1176 [==============================] - 13s 11ms/step - loss: 0.1621
```

Figure 4.6 : Implementation of LSTM

4.2.3 MLP Implementation

The model is a Multi-Layer Perceptron (MLP) neural network model defined using TensorFlow Keras. Here's a brief explanation of each component:

Input Layer:

Dense(64, activation='relu', input_shape=(X_train.shape[1],)):

This defines the input layer with 64 neurons and ReLU activation function.

The input_shape parameter specifies the shape of the input data, which is determined by the number of features in the training data (X_train.shape[1]).

Hidden Layers:

Dense(32, activation='relu'):

This is the first hidden layer with 32 neurons and ReLU activation function.

Dense(16, activation='relu'):

This is the second hidden layer with 16 neurons and ReLU activation function.

Dense(8, activation='relu'):

This is the third hidden layer with 8 neurons and ReLU activation function.

Output Layer:

Dense(1):

This is the output layer with a single neuron, representing the predicted value.

No activation function is specified, indicating it will output raw values.

Each hidden layer applies the Rectified Linear Unit (ReLU) activation function to introduce non-linearity, while the output layer produces the final prediction.

Model :

```
model_lon_mlp = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(32, activation='relu'),
    Dense(16, activation='relu'),
    Dense(8, activation='relu'),
    Dense(1)])
```

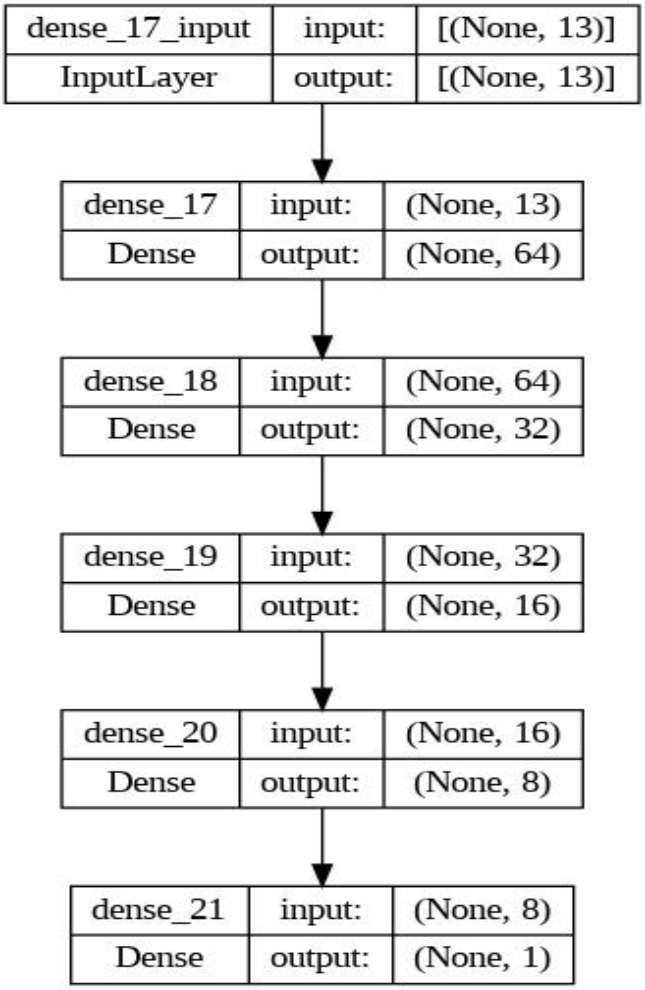

Figure 4.7 : MLP Model

```
Epoch 1/50
1176/1176 [==============================] - 3s 2ms/step - loss: 14.1517
Epoch 2/50
1176/1176 [==============================] - 2s 2ms/step - loss: 0.1439
Epoch 3/50
1176/1176 [==============================] - 2s 2ms/step - loss: 0.0592
Epoch 4/50
1176/1176 [==============================] - 3s 3ms/step - loss: 0.0355
Epoch 5/50
1176/1176 [==============================] - 2s 2ms/step - loss: 0.0253
Epoch 6/50
1176/1176 [==============================] - 2s 2ms/step - loss: 0.0220
Epoch 7/50
1176/1176 [==============================] - 2s 2ms/step - loss: 0.0272
Epoch 8/50
1176/1176 [==============================] - 2s 2ms/step - loss: 0.0097
Epoch 9/50
```

Figure 4.8 : Implementation of MLP

4.2.4 CNN-LSTM Implementation

The model is a neural network architecture comprising Convolutional Neural Network (CNN) layers followed by Long Short-Term Memory (LSTM) layers, designed for sequence data processing. Here's a brief explanation:

Convolutional Layers (Conv1D):

Two Conv1D layers with 64 and 32 filters respectively, each followed by a ReLU activation function.

These layers perform feature extraction from the input sequence data.

MaxPooling Layers:

Two MaxPooling1D layers with a pool size of 2.

These layers down-sample the input representation, reducing its dimensionality and extracting important features.

LSTM Layers:

Four LSTM layers with decreasing units (64, 32, 16, and 8) and ReLU activation functions.

These layers capture long-term dependencies in the sequential data, allowing the model to learn patterns over time.

Dense Layer:

A Dense layer with a single neuron, responsible for generating the final output prediction.

The output is a single value, suitable for regression tasks.

Overall, this architecture combines the strengths of CNNs in feature extraction from sequential data with the memory retention capabilities of LSTMs, making it effective for tasks such as time series prediction or sequence classification.

Model :

```
model_lat_cnn_lstm = Sequential([
    Conv1D(filters=64, kernel_size=3, activation='relu',
input_shape=(X_train_cnn.shape[1], X_train_cnn.shape[2])),
    MaxPooling1D(pool_size=2),
    Conv1D(filters=32, kernel_size=3, activation='relu'),
    MaxPooling1D(pool_size=2),
    LSTM(64, activation='relu', return_sequences=True),
    LSTM(32, activation='relu', return_sequences=True),
    LSTM(16, activation='relu', return_sequences=True),
    LSTM(8, activation='relu'),
    Dense(1)
])
```

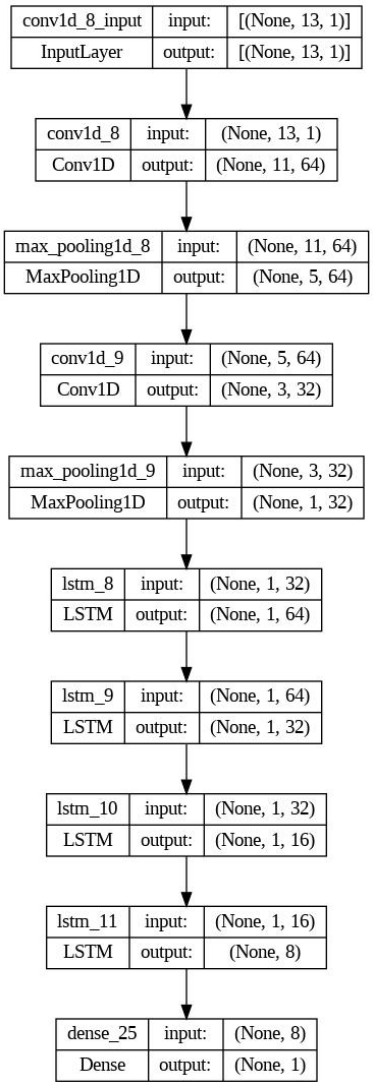

Figure 4.9 : CNN-LSTM Model

```
Epoch 1/50
1176/1176 [==============================] - 19s 11ms/step - loss: 22.3012
Epoch 2/50
1176/1176 [==============================] - 16s 13ms/step - loss: 0.5231
Epoch 3/50
1176/1176 [==============================] - 7s 6ms/step - loss: 0.2798
Epoch 4/50
1176/1176 [==============================] - 9s 8ms/step - loss: 0.2060
Epoch 5/50
1176/1176 [==============================] - 8s 7ms/step - loss: 0.1657
Epoch 6/50
1176/1176 [==============================] - 10s 8ms/step - loss: 0.1507
Epoch 7/50
1176/1176 [==============================] - 9s 8ms/step - loss: 0.1149
Epoch 8/50
1176/1176 [==============================] - 8s 7ms/step - loss: 0.0987
```

Figure 4.10 : Implementation of CNN-LSTM

4.2.5 CNN-GRU Implementation

A hybrid Convolutional Neural Network (CNN) and Gated Recurrent Unit (GRU) model cyclone track prediction based. Here's a brief explanation of each component:

Convolutional Layers:

Two Conv1D layers with 64 filters and a kernel size of 3, followed by max-pooling with a pool size of 2.

Used for feature extraction from sequential data, in this case, likely meteorological data represented in a time-series format.

GRU Layers:

Two GRU layers with 100 and 50 units, respectively, both with ReLU activation functions and returning sequences.

GRU layers process sequential data, capturing temporal dependencies and patterns in the input data.

Flatten Layer:

Flattens the output of the GRU layers into a one-dimensional array, preparing it for dense layers.

Dense Layers:

Two dense layers with 100 and 50 units, respectively, followed by a final dense layer with a single unit.

These dense layers perform classification/regression tasks on the extracted features.

Activation Functions:

ReLU activation functions are used in the convolutional, GRU, and dense layers to introduce non-linearity and enable the model to learn complex patterns.

This model architecture aims to leverage both CNNs and GRUs to capture spatial and temporal dependencies in the input data, making it suitable for tasks requiring prediction or classification based on sequential data, such as cyclone track prediction.

Model :

```
model_lat_cnn_gru = Sequential([
    Conv1D(filters=64,kernel_size=3,activation='relu',
      input_shape=(X_train_cnn.shape[1], X_train_cnn.shape[2])),
    MaxPooling1D(pool_size=2),
    GRU(100, activation='relu', return_sequences=True),
    GRU(50, activation='relu', return_sequences=True),
    Flatten(),
    Dense(100, activation='relu'),
    Dense(50, activation='relu'),
    Dense(1)
])
```

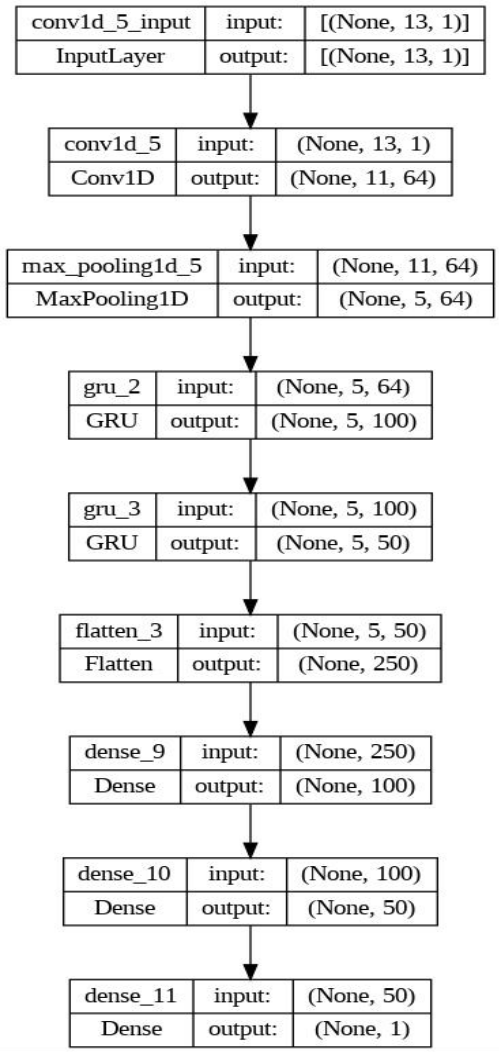

Figure 4.11 : CNN-LSTM Model

```
Epoch 1/50
1176/1176 [==============================] - 15s 10ms/step - loss: 8.1336
Epoch 2/50
1176/1176 [==============================] - 24s 20ms/step - loss: 0.1689
Epoch 3/50
1176/1176 [==============================] - 18s 16ms/step - loss: 0.0977
Epoch 4/50
1176/1176 [==============================] - 21s 18ms/step - loss: 0.0838
Epoch 5/50
1176/1176 [==============================] - 21s 18ms/step - loss: 0.1540
Epoch 6/50
1176/1176 [==============================] - 21s 18ms/step - loss: 0.0548
Epoch 7/50
1176/1176 [==============================] - 13s 11ms/step - loss: 0.0636
Epoch 8/50
```

Figure 4.12 : Implementation of CNN-LSTM

4.2.6 Visualization Implementation

The plot comparing the original track of Cyclone Michuang with its predicted track using a deep learning models. Here's a brief explanation:

Data Visualization:

Plotting the original and predicted track points of Cyclone Michuang on a map using longitude and latitude coordinates.

Plot Features:

Original track points are plotted in red, while predicted track points are plotted in blue.

The map includes land features and gridlines for better visualization.

Projection and Extent:

Utilizes the Plate Carrée projection for mapping.

Sets the plot extent to focus on the region where Cyclone Michuang occurred.

Legend and Title:

Adds a legend to distinguish between the original and predicted tracks.

Includes a title indicating the comparison between the original and predicted tracks.

Saving Plot:

Saves the generated plot for future reference.

This plot visually compares the original and predicted tracks of Cyclone Michuang, providing insight into the performance of the deep learning models in predicting cyclone trajectories.

Chapter 5
Results and discussion

5.1 RMSE

Root Mean Squared Error (RMSE) is a commonly used metric to evaluate the performance of a predictive model, especially in regression tasks like predicting latitude and longitude. It measures the average difference between the predicted values and the actual values, taking into account both the magnitude and direction of the errors.

$$RMSE = \sqrt{1/n * \sum_{i=1}^{n} (y - yp)\text{^}2}$$

where:

n is the number of data points.

y is the actual value.

yp is the predicted value.

Table 5.1 : RMSE

Model	Latitude	Longitude
CNN	0.12	0.43
LSTM	0.05	0.51
MLP	0.05	0.14
CNN-LSTM	0.095	0.11
CNN-GRU	0.06	0.14

With the help of RMSE we can tell that MLP and hybrid models CNN-LSTM, CNN-GRU are giving goof results in predicting latitude and longitude

5.2 Test Loss or MAE

Test loss is typically calculated using the loss function chosen during model compilation, such as Mean Squared Error (MSE) or another appropriate loss function. For example, in the code provided, the models are compiled using the mean squared error loss function (loss='mse'). The test loss is computed during model evaluation using this loss function.

$$Test\_loss = 1/n \sum_{i=1}^{n} (y - yp)\text{^}2$$

where:

n is the number of data points.

y is the actual value.

yp is the predicted value.

Table 5.2 : Test loss

Model	Latitude	Longitude
CNN	0.01	0.18
LSTM	0.003	0.2
MLP	0.002	0.02
CNN-LSTM	0.02	0.03
CNN-GRU	0.003	0.02

With the help of test loss we can tell that MLP and hybrid models CNN-LSTM, CNN-GRU are giving goof results in predicting latitude and longitude

5.3 Comparision of the Models on actual map

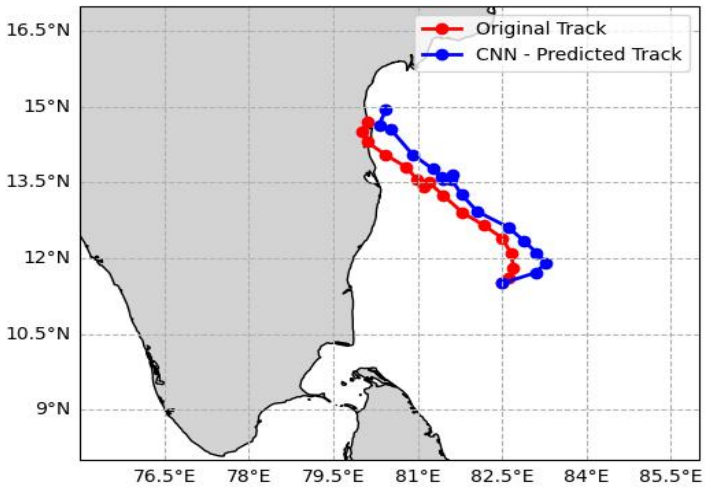

Figure 5.1 : CNN Track Michuang

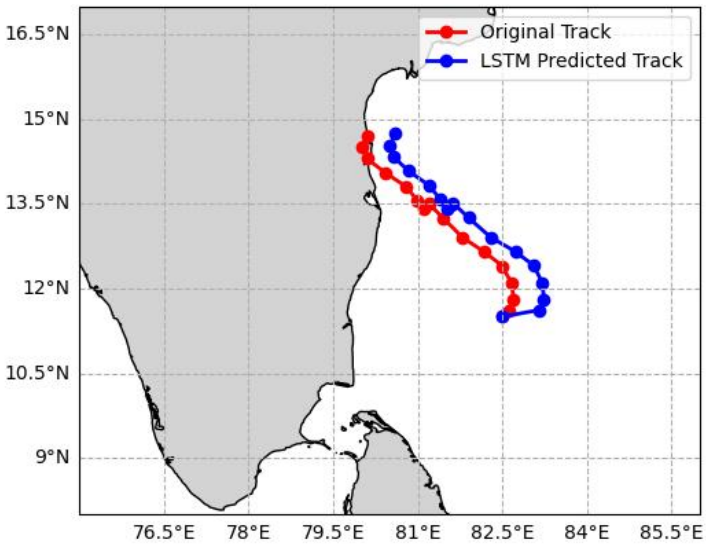

Figure 5.2 : LSTM Track Michuang

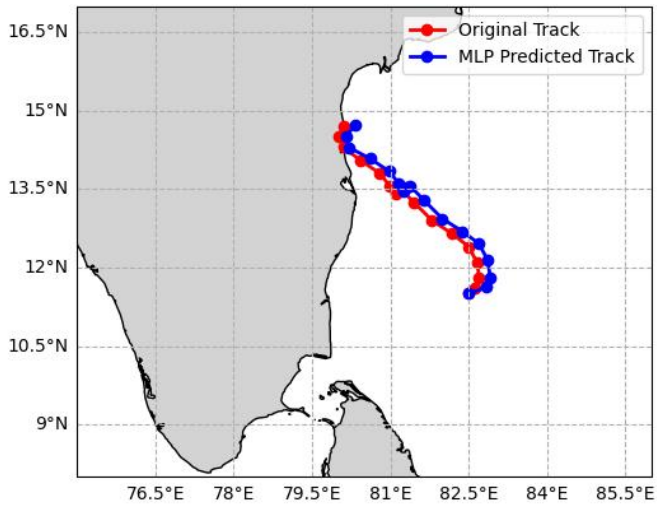

Figure 5.3 : MLP Track Michuang

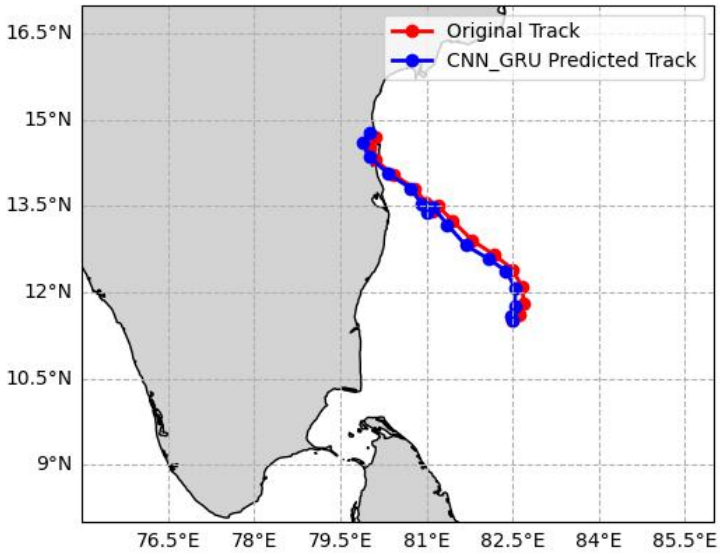

Figure 5.4 : CNN-GRU Track Michuang

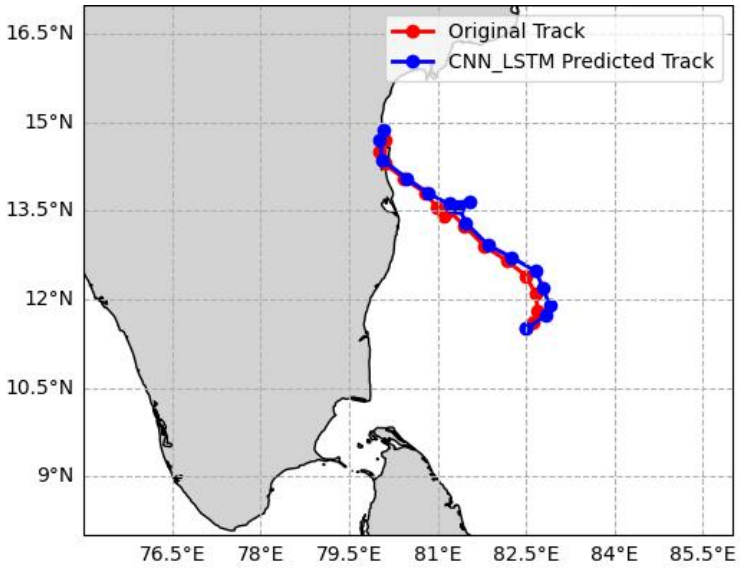

Figure 5.5 : CNN-LSTM Track Michuang

By the above tracks we can tell that the hybrid models CNN-GRU ,CNN-LSTM and MLP are giving more accurate yields in cyclone track prediction.

Chapter 6
Conclusion and future work

Based on the RMSE values and the visual comparison of cyclone tracking on the map, it's evident that the CNN-LSTM, CNN-GRU, and MLP models outperformed others in accurately predicting cyclone trajectories. These models achieved lower RMSE values for both latitude and longitude predictions compared to the CNN and LSTM models. This suggests that the combination of convolutional and recurrent neural network architectures (CNN-LSTM and CNN-GRU) and the multi-layer perceptron (MLP) architecture are more effective in capturing the complex patterns and dynamics of cyclone movements.

Furthermore, when observing the plotted cyclone tracks, it's noticeable that the predicted tracks by CNN-LSTM, CNN-GRU, and MLP closely align with the original tracks of Cyclone Michuang. This alignment indicates a higher level of accuracy and precision in predicting cyclone trajectories, which is crucial for disaster preparedness and response efforts. The ability of these models to accurately forecast cyclone paths can significantly contribute to early warning systems and decision-making processes, thereby enhancing the resilience of communities in cyclone-prone regions.

In conclusion, the CNN-LSTM, CNN-GRU, and MLP models demonstrate promising performance in cyclone track prediction, showcasing their potential for real-world applications in disaster management and meteorology. By leveraging deep learning techniques and effectively integrating spatial and temporal information, these models offer valuable insights into cyclone behavior, aiding in the mitigation of risks and the protection of lives and property during cyclone events. Further research and refinement of these models could lead to even more accurate and reliable predictions, ultimately contributing to the advancement of cyclone forecasting capabilities.

Looking ahead, the future scope of cyclone track prediction using deep learning models is promising. With ongoing advancements in technology and data availability, there are several avenues for further enhancement. One potential direction is the integration of additional environmental factors and satellite imagery data to improve the accuracy and robustness of predictions. Moreover, refining the models by incorporating more sophisticated architectures and optimizing hyperparameters could lead to even better performance. Additionally, the development of user-friendly interfaces and mobile applications for accessing and disseminating real-time cyclone forecasts could greatly benefit vulnerable communities, enabling timely evacuation and preparedness measures. Overall, continued research and innovation in this field hold the potential to enhance disaster resilience and save lives in cyclone-prone regions around the world.

7. References

1. Y. Zhang, R. Chandra and J. Gao, "Cyclone Track Prediction with Matrix Neural Networks," 2018 International Joint Conference on Neural Networks (IJCNN), Rio de Janeiro, Brazil, 2018, pp. 1-8, doi: 10.1109/IJCNN.2018.8489077.
2. S. Kumar, K. Biswas and A. K. Pandey, "Track Prediction of Tropical Cyclones Using Long Short-Term Memory Network," 2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC), NV, USA, 2021, pp. 0251-0257, doi: 10.1109/CCWC51732.2021.9376027.
3. Fang, W.; Lu, W.; Li, J.; Zou, L. A Novel Tropical Cyclone Track Forecast Model Based on Attention Mechanism. Atmosphere 2022, 13, 1607. https://doi.org/10.3390/atmos13101607
4. Giffard-Roisin S, Yang M, Charpiat G, Kumler Bonfanti C, Kégl B, Monteleoni C. Tropical Cyclone Track Forecasting Using Fused Deep Learning From Aligned Reanalysis Data. Front Big Data. 2020 Jan 28;3:1. doi: 10.3389/fdata.2020.00001. PMID: 33693376; PMCID: PMC7931887.
5. Wang, Liang & Wan, Bingcheng & Zhou, Shaohui & Sun, Haofei & Gao, Zhiqiu. (2022). Forecasting tropical cyclone tracks in the Northwest Pacific based on a deep-learning model. 10.5194/egusphere-2022-1216.
6. Lian, J.; Dong, P.; Zhang, Y.; Pan, J. A Novel Deep Learning Approach for Tropical Cyclone Track Prediction Based on Auto-Encoder and Gated Recurrent Unit Networks. Appl. Sci. 2020, 10, 3965. https://doi.org/10.3390/app10113965
7. Chen, R.; Zhang, W.; Wang, X. Machine Learning in Tropical Cyclone Forecast Modeling: A Review. Atmosphere 2020, 11, 676. https://doi.org/10.3390/atmos11070676
8. Tong, B. & Wang, X. & Fu, J.Y. & Chan, P.W. & He, Yuncheng. (2022). Short-term prediction of the intensity and track of tropical cyclone via ConvLSTM model. Journal of Wind Engineering and Industrial Aerodynamics. 226. 105026. 10.1016/j.jweia.2022.105026.
9. Akila Rajini Selvaraj, TamilPavai Gurusamy,An optimal model using single-dimensional CAE-IRNN based SPOA for cyclone track prediction,Expert Systems with Applications,Volume 230,2023,120437,ISSN 0957-4174,https://doi.org/10.1016/j.eswa.2023.120437
10. Pérez-Alarcón, Albenis & Coll-Hidalgo, Patricia & Trigo, Ricardo & Nieto, Raquel & Gimeno, Luis. (2024). CyTRACK: An open-source and user-friendly python toolbox for detecting and tracking cyclones. Environmental Modelling & Software. 176. 106027. 10.1016/j.envsoft.2024.106027.
11. Tiantian Wu, Zhongdong Duan,A new and efficient method for tropical cyclone detection and tracking in gridded datasets,Weather and Climate Extremes,Volume 42,2023,100626,ISSN 2212-0947,https://doi.org/10.1016/j.wace.2023.100626.
12. Enz, B. M., Engelmann, J. P., and Lohmann, U., “Use of threshold parameter variation for tropical cyclone tracking”,Geoscientific Model Development, vol. 16, no. 17, pp. 5093–5112, 2023. doi:10.5194/gmd-16-5093-2023.
13. Paul, Tushar & Ranganayakulu, Dhanalakshmi. (2023). A Study Of Cyclone Tracking Mechanism Using Deep Learning Techniques. 1-5. 10.1109/ICSCAN58655.2023.10395440

14. Wang, Yuanfei & Zhang, Wei & Fu, Wen. (2011). Back Propogation(BP)-neural network for tropical cyclone track forecast. Proceedings - 2011 19th International Conference on Geoinformatics, Geoinformatics 2011. 1-4. 10.1109/GeoInformatics.2011.5981095.
15. Kumar, Sandeep & Biswas, Koushik & Pandey, Ashish. (2021). Track Prediction of Tropical Cyclones Using Long Short-Term Memory Network. 0251-0257. 10.1109/CCWC51732.2021.9376027.
16. An assessment of long-term changes in mortalities due to extreme weather events in India: A study of 50 years' data, 1970–2019,Weather and Climate Extremes,Volume

Appendix
Code :

```
import pandas as pd

# Read the original CSV file
df = pd.read_csv("/content/drive/MyDrive/ibtracs.NI.list.v04r00.csv")

# Combine column names with descriptions
column_names_with_desc = []
for col, desc in zip(df.columns, df.iloc[0]):
    # Check if the second row is empty for this column
    if pd.isnull(df.iloc[1][col]):
        column_names_with_desc.append(col)
    else:
        column_names_with_desc.append(f"{col}_{desc}")

# Remove the first row (description row) from the DataFrame
df = df.iloc[1:]

# Assign the updated column names
df.columns = column_names_with_desc

print(df.columns)

df_ni = df[df['BASIN_ '] == 'NI']
df=df_ni
print(df_ni)

# Define prefixes to delete
prefixes_to_delete = ['WMO', 'USA', 'TOKYO', 'CMA', 'HKO', 'REUNION', 'BOM', 'NADI', 'WELLINGTON', 'DS', 'TD',
'NEUMANN', 'MLC','NEWDELHI','IFLAG']

# Filter columns to keep
columns_to_keep = [col for col in df.columns if not any(col.startswith(prefix) for prefix in prefixes_to_delete)]

# Create a new DataFrame with the columns to keep
df_filtered = df[columns_to_keep]

print(df_filtered)
# Filter rows where a specific column's value is different from "not_named"
df_filtered['SEASON_Year'] = df_filtered['SEASON_Year'].astype(int)
df_filtered = df_filtered[df_filtered['SEASON_Year'] >= 1900]
import math
import pandas as pd

def angleFromCoordinate(lat1, lon1, lat2, lon2):
    dLon = (lon2 - lon1)
    y = math.sin(dLon) * math.cos(lat2)
    x = math.cos(lat1) * math.sin(lat2) - math.sin(lat1) * math.cos(lat2) * math.cos(dLon)
    brng = math.atan2(y, x)
    brng = math.degrees(brng)
    brng = (brng + 360) % 360
    brng = 360 - brng  # count degrees clockwise - remove to make counter-clockwise
    return brng

def calculate_angles(df):
    angles = []
    prev_lat = None
    prev_lon = None
    prev_sid = None

    for index, row in df.iterrows():
        if prev_lat is None or prev_lon is None or row['NAME_ '] != prev_sid:
            angles.append(0.0)
        else:
            angle = angleFromCoordinate(prev_lat, prev_lon, row['LAT_degrees_north'], row['LON_degrees_east'])
            angles.append(angle)
```

```
            prev_lat = row['LAT_degrees_north']
            prev_lon = row['LON_degrees_east']
            prev_sid = row['NAME_ ']

        df['ANGLE'] = angles
        return df

    # Apply angle calculation function to the filtered DataFrame
    df_filtered = calculate_angles(df_filtered)

    # Display the DataFrame (no need to save)
    df_filtered.head(5)
    import math
    import pandas as pd

    def haversine(lat1, lon1, lat2, lon2):
        R = 6371.0  # Radius of the Earth in kilometers

        # Convert latitude and longitude from degrees to radians
        lat1_rad = math.radians(lat1)
        lon1_rad = math.radians(lon1)
        lat2_rad = math.radians(lat2)
        lon2_rad = math.radians(lon2)

        # Calculate the change in coordinates
        dlon = lon2_rad - lon1_rad
        dlat = lat2_rad - lat1_rad

        # Apply Haversine formula
        a = math.sin(dlat / 2)**2 + math.cos(lat1_rad) * math.cos(lat2_rad) * math.sin(dlon / 2)**2
        c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))

        # Calculate the distance
        distance = R * c
        return distance

    def calculate_distances(df):
        distances = []
        prev_lat = None
        prev_lon = None
        prev_sid = None

        for index, row in df.iterrows():
            if prev_lat is None or prev_lon is None or row['NAME_ '] != prev_sid:
                distances.append(0.0)
            else:
                distance = haversine(prev_lat, prev_lon, row['LAT_degrees_north'], row['LON_degrees_east'])
                distances.append(distance)
            prev_lat = row['LAT_degrees_north']
            prev_lon = row['LON_degrees_east']
            prev_sid = row['NAME_ ']

        df['DISTANCE_km'] = distances
        return df

    # Apply distance calculation function to the DataFrame
    df_filtered = calculate_distances(df_filtered)

    # Display the DataFrame with distances
    print(df_filtered)
    import pandas as pd

    # Assuming df_filtered is your DataFrame containing the relevant data

    # Convert ISO_TIME_ column to datetime objects with the specified format
    df_filtered['ISO_TIME_ '] = pd.to_datetime(df_filtered['ISO_TIME_ '], format='%d-%m-%Y %H:%M')

    # Sort DataFrame by 'NAME_' and 'ISO_TIME_'
    df_sorted = df_filtered.sort_values(by=['SID_ ', 'ISO_TIME_ '])
```

```
# Initialize a variable to store the first instance of ISO_TIME_ for each NAME_
first_instance_time = None

# Initialize a variable to store the previous NAME_
previous_name = None

# Initialize an empty list to store the time differences
time_differences = []

# Loop through each row in the sorted DataFrame
for index, row in df_sorted.iterrows():
    # Check if the NAME_ has changed
    if row['SID_ '] != previous_name:
        # If so, update the first_instance_time and previous_name
        first_instance_time = row['ISO_TIME_ ']
        previous_name = row['SID_ ']

    # Calculate the time difference in hours from the first instance time
    time_difference_hours = (row['ISO_TIME_ '] - first_instance_time).total_seconds() / 3600

    # Append the time difference to the list
    time_differences.append(time_difference_hours)

# Add the time differences as a new column to the DataFrame
df_sorted['TIME_DIFFERENCE_hours'] = time_differences

# Display the DataFrame
print(df_sorted)
df_sorted.to_csv('processed_data.csv')

import pandas as pd

# Convert ISO_TIME_ column to datetime objects with the specified format
df_sorted['ISO_TIME_ '] = pd.to_datetime(df_sorted['ISO_TIME_ '], format='%Y-%m-%d %H:%M:%S')

# Extract temporal features
df_sorted['Hour_of_the_Day'] = df_sorted['ISO_TIME_ '].dt.hour
df_sorted['Day_of_the_Week'] = df_sorted['ISO_TIME_ '].dt.dayofweek
df_sorted['Month'] = df_sorted['ISO_TIME_ '].dt.month
df_sorted['Season'] = df_sorted['ISO_TIME_ '].dt.month // 4  # Divide months into 4 seasons


# Calculate time elapsed since the start of each cyclone event
df_sorted['Time_Since_Start'] = df_sorted.groupby('SID_ ')['ISO_TIME_ '].transform(lambda x: (x - x.min()).dt.total_seconds() / 3600)

# Display the updated DataFrame
df_sorted.head()

import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense

# Convert selected features to numpy arrays
X = df_sorted[selected_features].values

# Standardize the input features
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Define the target variables
y_lat = df_sorted['LAT_degrees_north'].values
y_lon = df_sorted['LON_degrees_east'].values

# Split the data into train and test sets
X_train, X_test, y_lat_train, y_lat_test, y_lon_train, y_lon_test = train_test_split(X, y_lat, y_lon, test_size=0.2, random_state=42)
```

```
# Reshape input for CNN
X_train_cnn = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test_cnn = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))

# Define the CNN model with 4 layers for latitude prediction
model_lat_cnn = Sequential([
    Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(X_train_cnn.shape[1], X_train_cnn.shape[2])),
    MaxPooling1D(pool_size=2),
    Conv1D(filters=32, kernel_size=3, activation='relu'),
    MaxPooling1D(pool_size=2),
    Flatten(),
    Dense(100, activation='relu'),
    Dense(50, activation='relu'),
    Dense(1)
])

# Compile the latitude CNN model
model_lat_cnn.compile(optimizer='adam', loss='mse')

# Train the latitude CNN model
history_lat_cnn = model_lat_cnn.fit(X_train_cnn, y_lat_train, epochs=50, batch_size=32)
history_lat_cnn_dict = history_lat_cnn.history

# Evaluate the latitude CNN model on the test set
test_loss_lat_cnn = model_lat_cnn.evaluate(X_test_cnn, y_lat_test, verbose=0)
print('Latitude CNN Test Loss:', test_loss_lat_cnn)

# Define the CNN model with 4 layers for longitude prediction
model_lon_cnn = Sequential([
    Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(X_train_cnn.shape[1], X_train_cnn.shape[2])),
    MaxPooling1D(pool_size=2),
    Conv1D(filters=32, kernel_size=3, activation='relu'),
    MaxPooling1D(pool_size=2),
    Flatten(),
    Dense(100, activation='relu'),
    Dense(50, activation='relu'),
    Dense(1)
])

# Compile the longitude CNN model
model_lon_cnn.compile(optimizer='adam', loss='mse')

# Train the longitude CNN model
history_lon_cnn = model_lon_cnn.fit(X_train_cnn, y_lon_train, epochs=50, batch_size=32)
history_lon_cnn_dict = history_lon_cnn.history

# Evaluate the longitude CNN model on the test set
test_loss_lon_cnn = model_lon_cnn.evaluate(X_test_cnn, y_lon_test, verbose=0)
print('Longitude CNN Test Loss:', test_loss_lon_cnn)
from sklearn.metrics import mean_squared_error

# Calculate RMSE for latitude CNN
y_lat_pred_cnn = model_lat_cnn.predict(X_test_cnn)
rmse_lat_cnn = np.sqrt(mean_squared_error(y_lat_test, y_lat_pred_cnn))
print('Latitude CNN Test RMSE:', rmse_lat_cnn)

# Calculate RMSE for longitude CNN
y_lon_pred_cnn = model_lon_cnn.predict(X_test_cnn)
rmse_lon_cnn = np.sqrt(mean_squared_error(y_lon_test, y_lon_pred_cnn))
print('Longitude CNN Test RMSE:', rmse_lon_cnn)

# Create a copy of df_sorted to avoid modifying the original DataFrame
df_predicted = df_sorted.copy()
# Reshape input for prediction
X_predict_cnn = X.reshape((X.shape[0], X.shape[1], 1))

# Predict latitude for all instances
y_lat_predicted = model_lat_cnn.predict(X_predict_cnn)

# Predict longitude for all instances
```

```
y_lon_predicted = model_lon_cnn.predict(X_predict_cnn)

# Append predicted latitude and longitude to df_predicted DataFrame
df_predicted['CNN_latitude'] = y_lat_predicted
df_predicted['CNN_longitude'] = y_lon_predicted

# Print the updated DataFrame
df_predicted.head(10)

#similarly all the other were done and the comparision is

import matplotlib.pyplot as plt
from cartopy import crs as ccrs
from cartopy.feature import NaturalEarthFeature

# Define the coordinates
original_lons =michuang_df['LON_degrees_east'].values
original_lats = michuang_df['LAT_degrees_north'].values
predicted_lons =michuang_df['CNN_longitude'].values
predicted_lats =michuang_df['CNN_latitude'].values

map_proj = ccrs.PlateCarree()

# Create the plot
fig, ax = plt.subplots(subplot_kw={"projection": map_proj})

# Plot the original track points
ax.plot(original_lons, original_lats, 'ro-', color='red', linewidth=2, transform=map_proj, label='Original Track')

# Plot the predicted track points
ax.plot(predicted_lons, predicted_lats, 'bo-', color='blue', linewidth=2, transform=map_proj, label='CNN - Predicted Track')


# Add land feature
land_feature = NaturalEarthFeature('physical', 'land', '10m', edgecolor='black')
ax.add_feature(land_feature, facecolor='lightgray')

# Add gridlines
gl = ax.gridlines(draw_labels=True, linestyle='--')
gl.top_labels = False
gl.right_labels = False

# Set the plot extent and title
ax.set_xlim(75, 86)
ax.set_ylim(8, 17)
ax.set_title('Cyclone Michuang Track (Original vs. CNN Predicted Track)')

# Add legend
plt.legend()
plt.savefig('/content/plotssaved/CNN.png')
# Show the plot
plt.show()
```