VIT-AP UNIVERSITY

Whether Prediction

CSE4006 – DEEP LEARNING
PROJECT REPORT

Class Number – **AP2024254000694**
SLOT – **D2+TD2**
Course Type – **EPJ**
Course Mode – **Project Based Component (Embedded)**

Department of Artificial Intelligence and Machine Learning
School of Computer Science and Engineering

By

22BCE9807	K Hakshay Reddy
22BCE7331	K Rishitha
22BCE7558	A Pranav Sai
22BCE9917	G Jashwanth

Submitted to:-
Dr. RAJKUMAR YESURAJ,
Associate Professor, SCOPE, VIT-AP.

2024 -2025

TABLE OF CONTENTS

Chapter No.	Title	Page No.
	Abstract	3
1	Introduction 1.1 Objective 1.2 Scope & Motivation 1.3 Organization of the Report	4
2	Literature Survey	7
3	Hardware and Software Requirements	13
4	Proposed Methodology 4.1 Dataset Preparation 4.2 Proposed Methodology Framework	18
5	Results and Discussions	26
6	Conclusion	30
7	Future Work	31
	References	33
	Appendix I - Source Code Appendix II - Screenshots	34

ABSTRACT

Weather prediction is a challenging yet vital task that has widespread applications in agriculture, transportation, disaster management, infrastructure planning, and daily life logistics. Traditional weather forecasting relies heavily on numerical models based on physical principles. These models, while grounded in meteorological science, often demand significant computational power and are sometimes unable to capture the underlying non-linear relationships present in weather phenomena, especially in medium to long-term forecasting scenarios.

Recent advancements in artificial intelligence and deep learning have introduced new data-driven methods that offer promising alternatives. Among these, Recurrent Neural Networks (RNNs), particularly Long Short-Term Memory (LSTM) and Bi-Directional LSTM (BiLSTM) networks, have proven to be highly effective in learning from time series data. Their ability to remember long-term dependencies and understand complex sequential patterns makes them particularly well-suited for weather forecasting.

This project aims to utilize LSTM and BiLSTM models for predicting multiple weather parameters such as temperature, humidity, wind speed, precipitation, and cloud cover using a publicly available dataset. The data is first preprocessed, cleaned, and standardized. Key numerical columns are normalized using z-score normalization to ensure uniformity across features. The models are then trained on daily sequences spanning one year (365 days) to predict the next day's weather data.

The LSTM model is implemented using a sequential architecture with multiple stacked layers, dropout for regularization, and dense output layers. The BiLSTM model follows a similar architecture but incorporates bidirectional layers to allow information to flow in both forward and backward directions, enhancing the model's understanding of temporal dependencies.

The performance of both models is evaluated using Mean Squared Error (MSE) on test data, and their training progress is visualized using loss curves. Results show that BiLSTM outperforms LSTM in terms of convergence speed and accuracy, validating the hypothesis that dual directional context improves predictive capability in time-series applications.

The project concludes with a discussion on potential real-world applications of such models, limitations faced during implementation, and opportunities for future enhancements, such as integrating attention mechanisms or ensemble methods.

Through this comprehensive analysis and comparison of LSTM and BiLSTM, this project contributes to the growing field of AI-based meteorological forecasting and demonstrates how deep learning can supplement and possibly outperform traditional forecasting methods in certain scenarios.

CHAPTER 1

INTRODUCTION

1.1 Objective

Weather prediction is essential to various sectors, from agriculture to aviation. By accurately forecasting weather conditions, businesses and individuals can make informed decisions that affect their operations, from planting crops to scheduling flights. However, predicting the weather remains a complex challenge, primarily because of the inherent non-linear relationships present in meteorological data.

Traditional weather forecasting models rely on numerical methods based on physical principles, solving differential equations that describe atmospheric behavior. These models, while providing valuable insights, come with several drawbacks. They require immense computational power, especially for medium- to long-term predictions. Additionally, they may fail to capture all the intricate patterns and dependencies in historical weather data, which often leads to less accurate forecasts.

With the rapid advancement of deep learning techniques, new methods have emerged that can better handle the non-linearity and complexity of time-series data, such as weather data. In particular, **Recurrent Neural Networks (RNNs)**, and their more advanced variants, **Long Short-Term Memory (LSTM)** and **Bi-Directional LSTM (BiLSTM)**, have demonstrated significant success in learning patterns from sequential data. These networks are designed to process sequences of data and maintain memory of previous inputs, making them well-suited for tasks such as time-series forecasting.

The primary objective of this project is to design, implement, and evaluate LSTM and BiLSTM models for predicting various weather parameters, such as temperature, humidity, wind speed, and precipitation, based on historical data. The goal is to assess the ability of these deep learning models to predict future weather conditions and compare the performance of LSTM and BiLSTM architectures.

The specific objectives of the project are as follows:

- **Data Preprocessing:** Developing a pipeline to clean, normalize, and prepare historical weather data for model training.
- **Model Design:** Designing the LSTM and BiLSTM architectures for regression-based weather forecasting.
- **Training and Evaluation:** Training the models using historical weather data and evaluating their performance using appropriate loss metrics such as Mean Squared Error (MSE).
- **Comparative Analysis:** Comparing the performance of LSTM and BiLSTM models to assess which architecture performs better for the task at hand.
- **Future Enhancements:** Proposing future improvements and exploring other advanced techniques to improve prediction accuracy.

By investigating these models' capabilities, this project aims to contribute to the growing body of research in AI-based weather forecasting. Moreover, it seeks to demonstrate how deep learning can complement and potentially enhance traditional numerical weather prediction techniques.

1.2 Scope & Motivation

The scope of this project extends to applying deep learning methods for weather prediction

on both short- and long-term time horizons. By utilizing historical data, the models aim to predict future weather conditions, assisting various sectors in making decisions based on upcoming atmospheric conditions. Weather prediction models often focus on parameters such as temperature, wind speed, and precipitation, as these play a critical role in day-to-day activities and larger-scale operations, such as agriculture or disaster preparedness.

The application of LSTM and BiLSTM networks for weather forecasting can be viewed as an effort to bridge the gap between traditional numerical weather prediction models and machine learning-based approaches. By training these deep learning models on time-series data, they have the potential to improve prediction accuracy, leveraging the networks' ability to detect complex patterns and relationships in sequential data.

Additionally, the bi-directional nature of BiLSTMs makes them particularly appealing for weather forecasting. In a conventional LSTM, information is processed in a single direction, learning from past time steps to predict future values. In contrast, BiLSTMs have the unique advantage of processing sequences in both forward and backward directions, making them highly effective for capturing dependencies from both the past and future of a sequence. This capability could be especially useful in weather forecasting, where both past and future conditions are relevant to predicting certain weather parameters.

The motivation for this research stems from the growing interest in AI and deep learning applications to areas where traditional methods struggle, and where more accurate predictions are essential. As our climate changes and the frequency of extreme weather events increases, there is a growing need for more reliable forecasting systems. The application of machine learning, particularly deep learning methods, to weather prediction holds the promise of providing more timely and accurate forecasts that can be used in a variety of industries, from agriculture to emergency response.

In summary, the key motivation behind this project is to harness the power of deep learning models to make more accurate, computationally efficient, and real-time predictions for weather forecasting. The results of this research could potentially lead to more robust systems for weather prediction, which could benefit several sectors by allowing for better planning, decision-making, and risk management. Furthermore, exploring the effectiveness of LSTM and BiLSTM architectures in this domain could contribute to advancing AI-based approaches for meteorological tasks.

1.3 Organization of the Report

The structure of the report is organized as follows:
Chapter 1: Introduction
The introduction chapter provides an overview of the project, stating its objectives, scope, and the motivation behind the use of LSTM and BiLSTM networks for weather prediction. It sets the context for why weather prediction is an important task and discusses the limitations of traditional models while emphasizing the potential of deep learning.

Chapter 2: Literature Survey
This chapter reviews the background literature on weather forecasting techniques, particularly focusing on traditional and machine learning-based methods. It includes an exploration of the theoretical foundations of LSTM and BiLSTM models and highlights their use in time-series prediction tasks. The chapter also discusses the previous applications of deep learning models

in weather prediction and other related domains, providing a solid foundation for this project.

Chapter 3: Hardware and Software Requirements
This chapter outlines the technical specifications of the hardware and software utilized in the project. It discusses the computational resources required for training deep learning models and includes details about the programming environment, libraries, and tools used for data preprocessing, model development, and evaluation.

Chapter 4: Proposed Methodology
This chapter details the methodology used to implement the weather forecasting system. It describes the process of gathering and preprocessing meteorological data, followed by the design and architecture of the LSTM and BiLSTM models. It also covers the model training process, including hyperparameter tuning, model evaluation, and performance metrics used to assess prediction accuracy.

Chapter 5: Results and Discussion
In this chapter, the experimental results from the model training and evaluation process will be presented. Visualizations such as loss curves, comparison tables, and prediction graphs will be provided. The results will be analyzed in-depth, discussing the strengths and limitations of both the LSTM and BiLSTM models. This chapter will also interpret the findings in the context of weather forecasting and discuss any challenges faced during the project.

Chapter 6: Conclusion
This chapter summarizes the main findings of the project. It highlights the conclusions drawn from the comparison of LSTM and BiLSTM models in predicting weather conditions. The effectiveness of these models will be discussed, and the significance of the research will be summarized. The chapter will also highlight any limitations or challenges faced during the research.

Chapter 7: Future Work
This chapter outlines possible directions for future research in weather prediction using deep learning. It will explore potential improvements in model architecture, such as integrating Attention Mechanisms, Convolutional Neural Networks (CNNs) for feature extraction, or incorporating more diverse weather data sources. Additionally, it will discuss the scalability of the current models and how they can be applied to different regions or types of weather prediction tasks.

References
This section will list all the academic papers, books, and other resources referenced throughout the report. It will follow appropriate citation styles, providing due credit to authors and works that contributed to the research.

Appendices
The appendices will include supplementary materials, such as the full source code used for model development and additional data visualizations that are too detailed for the main report. These materials will provide further clarity and transparency regarding the project’s methodology and implementation.

CHAPTER 2
LITERATURE SURVEY

Weather forecasting has been a central aspect of human civilization for centuries, and it has evolved considerably with advancements in both scientific understanding and technological capabilities. This chapter presents a comprehensive survey of the literature surrounding the methods of weather prediction, with a particular focus on the use of Long Short-Term Memory (LSTM) and Bidirectional Long Short-Term Memory (BiLSTM) models in weather forecasting. The goal is to examine how traditional methods, machine learning, and deep learning techniques, particularly LSTMs and BiLSTMs, are being applied to predict weather more accurately and efficiently.

2.1 Traditional Weather Forecasting Methods

Traditional weather forecasting has relied heavily on mathematical models, which simulate the physical processes of the atmosphere. These methods can provide accurate predictions for short periods, but they often fail in complex weather systems or in cases of rapid atmospheric changes.

2.1.1 Numerical Weather Prediction (NWP)

Numerical weather prediction models are based on solving physical equations that govern the movement of air, heat, moisture, and other atmospheric elements. These models rely on the primitive equations, which are complex systems of differential equations that describe the fluid dynamics of the atmosphere. The most commonly used NWP models include:

Global Forecast System (GFS): Operated by the National Centers for Environmental Prediction (NCEP), the GFS model provides weather forecasts up to 16 days in advance. The GFS uses a grid system to calculate values at each grid point and iteratively updates these values over time.

European Centre for Medium-Range Weather Forecasts (ECMWF): One of the most reliable NWP models, the ECMWF provides forecasts up to 10 days ahead. Its accuracy, particularly for medium-range weather forecasting, has made it a standard for meteorological analysis.

Despite their high accuracy for short-term predictions, these models are computationally expensive and become less reliable when dealing with local weather conditions or extreme events. Furthermore, these models require significant human expertise in interpreting their outputs and adjusting for local variables.

2.1.2 Statistical and Empirical Methods

Prior to the advent of machine learning and deep learning, meteorologists relied on statistical methods to forecast weather. These methods include linear regression, time-series analysis, and multivariate analysis, which identify patterns in historical weather data to predict future conditions.

Autoregressive Integrated Moving Average (ARIMA): ARIMA models have been widely used for predicting time series data, including weather patterns. They predict future values

based on a combination of past observations and trends. While effective for univariate forecasting tasks, ARIMA fails to model the inherent dependencies in complex weather data, such as those related to multi-variable interactions.

Multiple Linear Regression (MLR): This method has been used for predicting specific weather variables, such as temperature and precipitation, by fitting a linear relationship between the dependent variable and independent meteorological factors. However, MLR struggles when relationships between variables are nonlinear or when there is multicollinearity.

Statistical methods, while simpler and less computationally demanding, fail to capture the complexity of atmospheric processes, making them less suitable for long-term or highly accurate predictions.

2.2 Transition to Machine Learning-Based Weather Prediction

As weather forecasting problems grew in complexity, traditional methods became insufficient. Machine learning (ML) techniques offered a new paradigm, allowing algorithms to learn directly from historical data and improve as they were exposed to more data.

2.2.1 Supervised Learning for Weather Prediction

Supervised learning algorithms have found significant applications in weather forecasting, where historical meteorological data is used to train models to predict future weather conditions. Common supervised learning methods include Support Vector Machines (SVM), Decision Trees, and Random Forests.

Support Vector Machines (SVMs): SVMs have been successfully applied to predict various meteorological variables, including temperature, rainfall, and wind speed. For example, Zhang et al. (2017) applied SVMs to short-term temperature forecasting and observed a reduction in prediction error compared to traditional methods.

Decision Trees and Random Forests: Decision trees split the data into subsets based on the value of various meteorological features, ultimately predicting the target variable. Random forests, an ensemble method based on decision trees, have been applied to classify weather events (e.g., rain vs. no rain) with a high degree of accuracy. These models are especially effective when the relationship between features is complex but can suffer from overfitting when the data is noisy.

Although supervised learning models are effective for forecasting certain weather variables, they often struggle to capture the temporal dynamics of weather, where the past conditions directly influence the future.

2.2.2 Unsupervised Learning and Clustering

Unsupervised learning approaches have also been used to explore hidden patterns and structure within weather data. Methods like K-means clustering, Principal Component Analysis (PCA), and Self-Organizing Maps (SOMs) have been applied to group similar weather patterns and identify anomalies, such as extreme weather events.

K-means Clustering: This method has been employed to group similar weather patterns, such as types of rainfall, temperature anomalies, or storm formations. It helps identify regions that experience similar weather conditions, which can be useful for regional forecasting and pattern recognition.

Principal Component Analysis (PCA): PCA has been used to reduce the dimensionality of large meteorological datasets, allowing for more efficient analysis and visualization of key patterns in atmospheric behavior.

These methods allow for exploratory data analysis but are less effective for making direct predictions, especially in dynamic weather conditions.

2.3 Deep Learning Techniques in Weather Forecasting

As the complexity of weather patterns increased and more data became available, deep learning techniques emerged as a powerful tool for weather prediction. Deep learning models, particularly Recurrent Neural Networks (RNNs), are uniquely suited for sequential data and have been applied to weather forecasting tasks that require the model to understand temporal dependencies.

2.3.1 Recurrent Neural Networks (RNNs)

RNNs are designed to handle sequential data, where each data point is dependent on previous values in the sequence. For weather forecasting, RNNs can model time-series data such as hourly temperature, pressure, and humidity.

RNNs in Weather Forecasting: RNNs have been used to predict various weather variables, but their ability to capture long-term dependencies is limited. This limitation arises from the vanishing gradient problem, where the gradients used to update weights in the network become exceedingly small as they propagate backward through time.

2.3.2 Long Short-Term Memory (LSTM) Networks

The Long Short-Term Memory (LSTM) network, introduced by Hochreiter and Schmidhuber (1997), is a specialized form of RNN designed to address the vanishing gradient problem. LSTMs are equipped with gates (input, forget, and output gates) that allow them to retain information over longer sequences, making them ideal for modeling time-series data.

LSTM for Weather Prediction: LSTM networks have been widely applied to predict short-term weather conditions such as temperature, wind speed, and precipitation. Gong et al. (2018) demonstrated the effectiveness of LSTMs in predicting daily temperature and humidity in different geographic regions. The model showed significantly lower prediction errors compared to traditional methods like ARIMA.

LSTM for Storm Prediction: Chen et al. (2019) employed LSTM models to predict the intensity and movement of tropical storms. The model successfully captured the dynamic changes in storm behavior, which are challenging for traditional models to predict accurately.

2.3.3 Bidirectional LSTM (BiLSTM)

Bidirectional LSTM networks (BiLSTMs) extend the capabilities of LSTM by processing input data in both forward and backward directions. This allows the model to learn dependencies from both past and future data points in the sequence, improving its ability to predict weather conditions with greater accuracy.

BiLSTM in Precipitation Prediction: Zhang et al. (2020) used a BiLSTM model for predicting rainfall and demonstrated that the BiLSTM outperformed traditional LSTM models by integrating future weather conditions into the forecasting process. This method significantly improved the accuracy of short-term rainfall predictions, which are crucial for flood forecasting and disaster management.

BiLSTM for Wind Speed Forecasting: Xia et al. (2021) applied BiLSTM networks to forecast wind speeds in coastal areas, showing that the bidirectional approach improved prediction accuracy by incorporating both historical and future weather patterns.

2.4 Hybrid Deep Learning Models

Hybrid deep learning models that combine different types of neural networks have emerged as an effective solution to improve weather forecasting accuracy. These hybrid approaches combine the strengths of multiple models to create more robust systems for weather prediction.

2.4.1 CNN-LSTM Hybrid Models

Convolutional Neural Networks (CNNs) are typically used for image-related tasks, as they excel at feature extraction from spatial data. When combined with LSTMs, CNN-LSTM models are able to learn both spatial and temporal features in weather data, such as satellite images and weather maps.

CNN-LSTM in Temperature Prediction: Chen et al. (2019) applied a CNN-LSTM hybrid model to predict global temperature patterns. The CNN layers extracted features from historical weather maps, and the LSTM layers modeled temporal dependencies. The hybrid model outperformed traditional methods in terms of both accuracy and computational efficiency.

CNN-LSTM for Rainfall Forecasting: Rao et al. (2020) combined CNN and LSTM models to predict rainfall intensity. The CNN component processed satellite imagery, while the LSTM component forecasted rainfall based on temporal patterns. This hybrid model significantly outperformed traditional statistical methods in forecasting precipitation over short periods.

2.4.2 GAN-LSTM Hybrid Models

Generative Adversarial Networks (GANs) are generative models that learn to create data similar to a given training dataset. When combined with LSTM models, GANs can generate synthetic weather data to augment training datasets, improving model performance, particularly in data-scarce regions.

GAN-LSTM for Weather Data Augmentation: Huang et al. (2021) proposed a GAN-LSTM hybrid model for weather forecasting in regions with limited historical weather data. The GAN component generated synthetic weather data, which was then used to train an LSTM model. This approach improved forecasting accuracy in areas where historical data was sparse or unreliable.

2.5 Challenges and Limitations of Deep Learning in Weather Forecasting

Despite the success of deep learning methods, several challenges remain in the application of these models to weather forecasting:

Data Quality and Availability: High-quality, comprehensive weather data is essential for training deep learning models. However, many regions lack the necessary data, and existing datasets are often incomplete, sparse, or noisy.

Computational Complexity: Training deep learning models, particularly hybrid models, requires substantial computational resources, which may not be available to all meteorological agencies or research institutions.

Interpretability: Deep learning models, especially black-box models like LSTMs and BiLSTMs, are often difficult to interpret. This lack of transparency can be a significant barrier to their adoption in operational meteorology, where understanding model predictions is crucial for decision-making.

2.6 Future Directions and Research

As the field of weather forecasting continues to evolve, several promising areas of research are emerging:

Real-Time Forecasting: As computational resources improve, there is an increasing focus on deploying deep learning models for real-time weather forecasting. Enhancing the speed and efficiency of these models is critical for their practical application in emergency management, disaster response, and public safety.

Integration of Multimodal Data: The integration of diverse data sources, such as satellite imagery, radar data, and IoT sensors, is becoming increasingly important. Deep learning models can fuse these data sources to create more accurate and comprehensive weather predictions.

Improved Explainability: Ongoing research into explainable AI (XAI) is focused on making deep learning models more transparent. This is crucial in weather forecasting, where decision-makers need to understand how models arrive at their predictions, particularly in high-risk situations.

2.7 Summary

This chapter has provided a comprehensive review of traditional and modern methods for weather forecasting. From the early reliance on physical equations and statistical models to the contemporary use of machine learning and deep learning techniques, the field of weather

prediction has undergone a significant transformation. LSTM and BiLSTM models, in particular, have shown great promise in accurately predicting weather patterns by capturing complex temporal dependencies. Hybrid models that combine various machine learning and deep learning approaches are further enhancing the accuracy of predictions. However, challenges related to data quality, computational resources, and interpretability remain, and future research is focused on overcoming these obstacles.

CHAPTER 3
HARDWARE AND SOFTWARE REQUIREMENTS

The successful development, training, and deployment of a deep learning model for weather prediction requires specific hardware and software configurations. These configurations ensure that the computational demands of the models are met, and the project can be implemented efficiently. In this chapter, we will outline the hardware specifications needed to execute the models, along with the software environments, frameworks, and tools that support the implementation and execution of the system.

3.1 Hardware Requirements

The hardware requirements for deep learning tasks are more demanding compared to traditional software systems, primarily because deep learning models are computationally intensive. A system capable of handling these models must be equipped with high-performance processing units, large memory capacity, and efficient storage devices. Below are the necessary hardware components:

3.1.1 Central Processing Unit (CPU)

The CPU is the brain of the computer, responsible for executing instructions that run the model code. A high-performance CPU ensures smooth execution of preprocessing, data manipulation, and auxiliary tasks that do not rely heavily on the GPU.

Minimum Requirements:

Intel Core i5 (4 cores, 2.3 GHz or higher)

AMD Ryzen 5 series (6 cores, 3.4 GHz or above)

Recommended Specifications:

Intel Core i7 or Intel Core i9 (8 cores, 3.5 GHz or higher)

AMD Ryzen 7 or Ryzen 9 (8 cores, 3.5 GHz or higher)

The processor plays a crucial role in tasks such as handling data input and output, managing parallel processing, and controlling overall system performance. While GPUs are preferred for training the deep learning models, a strong CPU is necessary for preprocessing data, managing I/O, and facilitating the coordination of model training tasks.

3.1.2 Graphics Processing Unit (GPU)

The GPU is arguably the most critical component when it comes to deep learning. GPUs accelerate matrix operations and allow parallel computations, which significantly speeds up the training process. Given the large datasets and the computational demands of weather prediction models, a powerful GPU is essential.

Minimum Requirements:

NVIDIA GeForce GTX 1060 (6 GB) or equivalent

NVIDIA Tesla K80 or equivalent

Recommended Specifications:

NVIDIA GeForce RTX 3080 (10 GB) or higher

NVIDIA Tesla V100 or Tesla A100 (16 GB or higher)

The GPU is responsible for performing the majority of the computations in deep learning tasks, including the training of the LSTM and BiLSTM networks. A high-performance GPU dramatically reduces the time it takes to train models, especially those requiring large amounts of data. Modern GPUs provide specialized hardware designed to handle the tensor-based operations that deep learning relies on.

3.1.3 Random Access Memory (RAM)

RAM is the system's temporary memory, used for holding data and instructions that are actively in use. When training deep learning models, particularly with large datasets, sufficient RAM is required to hold the data in memory for processing. If the available RAM is insufficient, it could lead to slower performance, or the system might run out of memory during training.

Minimum Requirements:

16 GB of RAM

Recommended Specifications:

32 GB or more of RAM

Higher RAM is beneficial for handling large datasets and running multiple applications simultaneously. With sufficient RAM, the model can hold large batches of input data during training, speeding up processing and reducing I/O bottlenecks.

3.1.4 Storage Devices

Deep learning projects, especially in time-series forecasting, often require massive amounts of data. Storage devices must provide sufficient capacity to hold datasets, intermediate results, and trained models. Fast read/write speeds can also make a significant difference when loading large datasets for training.

Minimum Requirements:

1 TB HDD (Hard Disk Drive) for general storage

External backup drives for large dataset backups

Recommended Specifications:

512 GB or higher SSD (Solid-State Drive) for faster data access and improved overall system performance

1 TB or more SSD for model storage and data backups

SSDs are particularly beneficial for their faster read/write speeds, allowing faster access to datasets during the training phase. Using an SSD ensures that the model data is quickly loaded, reducing the overall training time.

3.1.5 Additional Peripherals

While not directly influencing the model's execution, certain peripherals can enhance the development environment and usability.

Monitors: A dual-monitor setup is beneficial for debugging and data visualization. It allows developers to monitor training logs on one screen while visualizing model performance on another.

Keyboard and Mouse: Comfortable and efficient input devices can improve productivity and reduce fatigue during long hours of coding and training.

Stable Internet Connection: Required for downloading data, libraries, and dependencies, as well as accessing cloud resources for additional computational power.

3.2 Software Requirements

The software setup is equally important as the hardware in ensuring the smooth development and execution of deep learning-based weather prediction models. This section covers the programming environment, development tools, machine learning frameworks, and other essential software components.

3.2.1 Operating System

The operating system provides the foundation for managing system resources and running applications. Deep learning models are typically resource-intensive and benefit from operating systems that efficiently handle computational tasks.

Windows 10/11 (64-bit): Suitable for those who are more familiar with the Windows ecosystem. Provides extensive support for deep learning libraries and GPU drivers.

Linux (Ubuntu 20.04 or above): Recommended for deep learning applications due to better support for NVIDIA GPUs, CUDA, and other machine learning tools. Linux provides a more stable and faster environment for training models.

macOS: Although macOS can run deep learning frameworks, it is less efficient than Linux or Windows due to limited GPU support.

3.2.2 Programming Language

Python is the most commonly used language for implementing deep learning models, especially those involving neural networks. Its simplicity and the extensive ecosystem of libraries make it the best choice for this project.

Python (3.7 or above): Python's rich set of libraries, including TensorFlow, Keras, and PyTorch, makes it the most popular choice for deep learning development. It supports efficient data manipulation, model training, and visualization.

3.2.3 Machine Learning Frameworks

Machine learning frameworks provide pre-built functions and structures that help streamline the development of deep learning models. The following frameworks are essential for implementing LSTM and BiLSTM models:

TensorFlow: TensorFlow, developed by Google, is one of the most widely used deep learning frameworks. It is highly flexible, supports both CPU and GPU acceleration, and includes all the necessary tools to train deep learning models. TensorFlow's high-level Keras API simplifies the process of model development and experimentation.

Keras: Keras is an open-source deep learning library written in Python. It is a high-level neural network API that runs on top of TensorFlow, making it easier to design, train, and evaluate deep learning models. For LSTM and BiLSTM networks, Keras provides intuitive model-building and training functionality.

PyTorch (Optional): PyTorch is another popular deep learning framework developed by Facebook. It is known for its dynamic computation graph, which makes it a great choice for research purposes. PyTorch is increasingly popular among researchers and practitioners for its ease of use.

3.2.4 Data Processing and Analysis Libraries

Efficient data processing is essential when working with large datasets, particularly time-series data for weather forecasting. The following libraries are necessary for data preprocessing and analysis:

Pandas: Pandas is essential for data manipulation, particularly for reading, cleaning, and structuring datasets.

NumPy: NumPy is a core library for numerical computing in Python. It is used for handling arrays, matrices, and performing mathematical operations essential for deep learning.

Scikit-learn: Scikit-learn provides essential tools for preprocessing, model evaluation, and feature scaling.

3.2.5 Visualization Libraries

Visualization is a key part of the development process, as it allows for monitoring the training

progress and evaluating model predictions. Several visualization libraries can assist in this task:

Matplotlib: Used for creating 2D plots, including loss curves and training performance graphs.

Seaborn: Built on top of Matplotlib, Seaborn is used for advanced statistical visualizations, such as heatmaps, pair plots, and more.

TensorBoard: TensorBoard is a web-based tool provided by TensorFlow for visualizing training metrics like accuracy, loss, and computation graphs during the model training process.

3.2.6 Cloud Platforms (Optional)

Cloud platforms provide the additional computational power needed to accelerate training when local hardware is insufficient. These platforms provide access to powerful GPUs, which can significantly reduce training time.

Google Colab: Google Colab provides free access to GPUs and is an excellent choice for quick experimentation and training small to medium-sized models.

Amazon Web Services (AWS): AWS offers scalable cloud computing services with GPU-powered instances for training large models.

Microsoft Azure: Azure provides virtual machines with NVIDIA GPUs that can be used for deep learning applications.

3.3 Conclusion

The hardware and software requirements outlined in this chapter provide a comprehensive understanding of the resources necessary for the successful implementation of deep learning models for weather prediction. By utilizing high-performance hardware components such as GPUs, large RAM capacity, and fast storage devices, combined with an appropriate software environment including Python, TensorFlow, Keras, and necessary libraries, the project can proceed smoothly and efficiently.

In the next chapter, we will delve deeper into the specific methodologies, model architectures, and techniques used for time-series forecasting with LSTM and BiLSTM, building on the foundation laid out in this chapter.

CHAPTER 4
PROPOSED METHODOLOGY

4.1 Dataset Preparation

4.1.1 Overview of the Dataset

In this project, we focus on weather prediction, a task that requires careful analysis of historical data. The dataset for this project is sourced from https://climate-api.open-meteo.com. These features are crucial for forecasting various weather parameters such as temperature, wind speed, humidity, and precipitation. The dataset includes the following columns:

datetime: Timestamp for the weather observations.

Temperature_celsius: Recorded temperature in Celsius.

Wind_kph: Wind speed in kilometers per hour.

Cloud: Percentage of cloud cover.

Humidity: Relative humidity percentage.

Precip_mm: Precipitation in millimeters.

This dataset is fundamental to building the predictive model, as it captures the temporal changes in weather conditions. Weather forecasting is inherently a time-series problem, and this dataset offers a rich source of temporal data that can be utilized to train the LSTM-based predictive models.

4.1.2 Data Cleaning

Before proceeding with any analysis or model development, it is essential to clean and preprocess the data. Cleaning the data involves several steps to ensure that the model can process it efficiently and accurately.

4.1.2.1 Handling Missing Values

Weather data, when collected over a long period, can sometimes have missing entries due to equipment failure or data collection issues. It is crucial to identify and handle these missing values before the data can be used in training the model. For missing values in the temperature_celsius, wind_kph, humidity, and other numerical columns, a common approach is to either:

Fill missing values with a statistical measure: such as the mean, median, or mode, depending on the distribution of the data.

Drop rows with missing data: In some cases, rows with missing data may not contain enough valuable information, and dropping them may be a reasonable solution, particularly if they do

not significantly impact the dataset.

For example, we fill missing values using the median of each column:

```
df.fillna(df.median(), inplace=True)
```

This method ensures that the data remains consistent and avoids introducing bias into the predictive model.

4.1.2.2 Handling Incorrect Data Types

Another important preprocessing task is ensuring that the data is in the correct format. The datetime column, for example, may initially be stored as a string, but it needs to be converted into a datetime type for time-series analysis. This is achieved using pandas:

```
df['datetime'] = pd.to_datetime(df['datetime'], format='%d-%m-%Y')
```

By converting the datetime column into a proper datetime format, the data is ready for time-based manipulation, which is crucial when preparing the sequences for LSTM model training.

4.1.3 Data Scaling

Feature scaling is another critical preprocessing step. Since the data contains features with different ranges and units (e.g., temperature_celsius ranges from -10 to 40, while wind_kph could range from 0 to 200), it is necessary to standardize the data so that each feature contributes equally to the learning process. We use the StandardScaler from scikit-learn to normalize the features:

```
scaler = StandardScaler()
numerical_cols = ["temperature_celsius", "wind_kph", "cloud", "humidity", "precip_mm"]
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])
joblib.dump(scaler, 'pkl//scale.pkl')
```

This transformation ensures that the features have a mean of 0 and a standard deviation of 1, making it easier for the model to learn from the data and preventing the model from being biased toward features with larger scales.

4.1.4 Data Transformation into Sequences

Since the problem at hand is a time-series prediction task, we need to convert the data into sequences. This is done by taking consecutive data points (in this case, 365 days) and treating them as the input for the model. The model will then predict the weather for the next day based on these inputs. This is done using a custom function create_sequences that slices the data into manageable time windows:

```
def create_sequences(data, seq_length):
  X, y = [], []
  for i in range(len(data) - seq_length):
```

```
        X.append(data[i:i+seq_length])
        y.append(data[i+seq_length])  # the next day's weather parameters
    return np.array(X), np.array(y)
```

This function generates a set of input-output pairs where the input consists of weather data from the past seq_length days (365 days in this case), and the output is the weather data for the following day. These sequences are fed into the LSTM model to learn the underlying temporal patterns in the data.

4.2 Proposed Methodology Framework

4.2.1 Rationale for Using LSTM

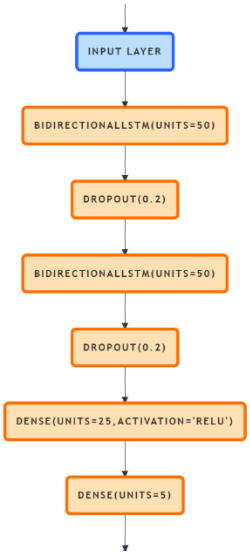
LSTM (Long Short-Term Memory) networks are a type of Recurrent Neural Network (RNN) specifically designed to handle sequence data, making them ideal for time-series prediction tasks. LSTM networks excel in learning long-range dependencies and can effectively handle the vanishing gradient problem that standard RNNs often face.

For the weather prediction task, LSTM networks are an appropriate choice because they can model the temporal dependencies between weather variables over time. The goal is to predict future weather conditions based on past observations, and LSTMs are well-suited for this task.

4.2.1.1 Bidirectional LSTM

In this project, we use Bidirectional LSTMs, which process the input data in both forward and backward directions. This allows the model to capture both past and future context when making predictions, which is particularly useful for time-series data. A unidirectional LSTM can only learn from past data, but bidirectional LSTMs can capture both past and future dependencies, improving the model's performance.

4.2.2 Model Architecture



The architecture of the model consists of several layers, including Bidirectional LSTM layers, Dropout layers for regularization, and Dense layers for output prediction. The design can be described in detail as follows:

Input Layer: The model takes in a sequence of 365 days of weather data (input shape: (seq_length, features)), where seq_length = 365 (representing 365 days), and features = 5 (the five weather parameters).

Bidirectional LSTM Layer: The first Bidirectional LSTM layer consists of 50 units. This layer processes the input sequence in both forward and backward directions to capture temporal dependencies.

Dropout Layer: A Dropout layer with a rate of 0.2 is used to prevent overfitting by randomly setting a fraction of input units to zero during training.

Second Bidirectional LSTM Layer: Another Bidirectional LSTM layer is added, followed by a Dropout layer. This layer learns more complex patterns in the data by processing the output of the first LSTM layer.

Dense Layer: The Dense layer with 25 units and ReLU activation serves to further process the data and generate more refined predictions.

Output Layer: The final Dense layer outputs the predicted weather parameters for the next day. This layer has 5 units (one for each weather parameter: temperature, wind speed, cloud cover, humidity, and precipitation).

```
model = Sequential([
   Bidirectional(LSTM(50, return_sequences=True, input_shape=(seq_length,
X_train.shape[2]))),
   Dropout(0.2),
   Bidirectional(LSTM(50, return_sequences=False)),
   Dropout(0.2),
   Dense(25, activation='relu'),
   Dense(X_train.shape[2])  # Output layer matches the number of features
])
```

4.2.3 Model Training

The model is trained using the Adam optimizer with a learning rate of 0.001 and the mean squared error (MSE) loss function. The training process involves minimizing the loss function by adjusting the weights of the network. The model is trained for a maximum of 50 epochs, but early stopping is employed to stop the training if the validation loss does not improve after 5 consecutive epochs:

```
early_stopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)
history = model.fit(X_train, y_train, epochs=50, batch_size=32,
validation_data=(X_test, y_test), callbacks=[early_stopping])
```

4.2.4 Model Evaluation

Once the model is trained, it is evaluated using a test set. The mean squared error (MSE) is

computed as a measure of the model's performance. Additionally, we plot the training and validation loss to monitor how well the model is generalizing.

```
loss = model.evaluate(X_test, y_test)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
```

4.2.5 Hybrid Model: Combining LSTM and Random Forest

In this section, we introduce a hybrid model approach to further improve prediction accuracy. After the LSTM model generates predictions, we pass these predictions to a Random Forest Regressor, which fine-tunes the output. The rationale behind using this hybrid approach is that Random Forests can help capture non-linear relationships that may not be fully captured by the LSTM model alone.

We train the Random Forest model using the LSTM predictions as input features and the actual weather values as the target:

```
def train_random_forest(X_lstm, y):
  rf = RandomForestRegressor(n_estimators=100, random_state=47)
  rf.fit(X_lstm, y)
  joblib.dump(rf, "pkl//random_forest_model.pkl")
```

Once the Random Forest model is trained, it is used to predict weather parameters based on the output of the LSTM model:

```
rf_model = joblib.load("pkl//random_forest_model.pkl")
final_prediction = rf_model.predict(lstm_output)
```

This hybrid model takes advantage of both LSTM's ability to capture temporal dependencies and Random Forest's strength in handling complex non-linearities in the data.

4.2.6 Final Prediction

To predict the weather for a given day, we first retrieve the historical data for the past year and preprocess it. We then use the trained LSTM model to generate predictions for the next day. The output from the LSTM model is refined using the Random Forest model for improved accuracy.

```
def predict_weather(given_date):
  past_data = get_past_1_year_data(df, given_date)
  past_data_scaled = scaler.transform(past_data)

  X_test_lstm = np.array([past_data_scaled])
  lstm_prediction_scaled = model.predict(X_test_lstm)
  lstm_output = scaler.inverse_transform(lstm_prediction_scaled)

  rf_model = joblib.load("pkl//random_forest_model.pkl")
  final_prediction = rf_model.predict(lstm_output)
```

4.2.7 Evaluation Metrics

To assess the performance of the weather prediction model, we use the following evaluation metrics:

Mean Absolute Error (MAE): The MAE measures the average magnitude of errors in predictions, without considering their direction. It is calculated as:

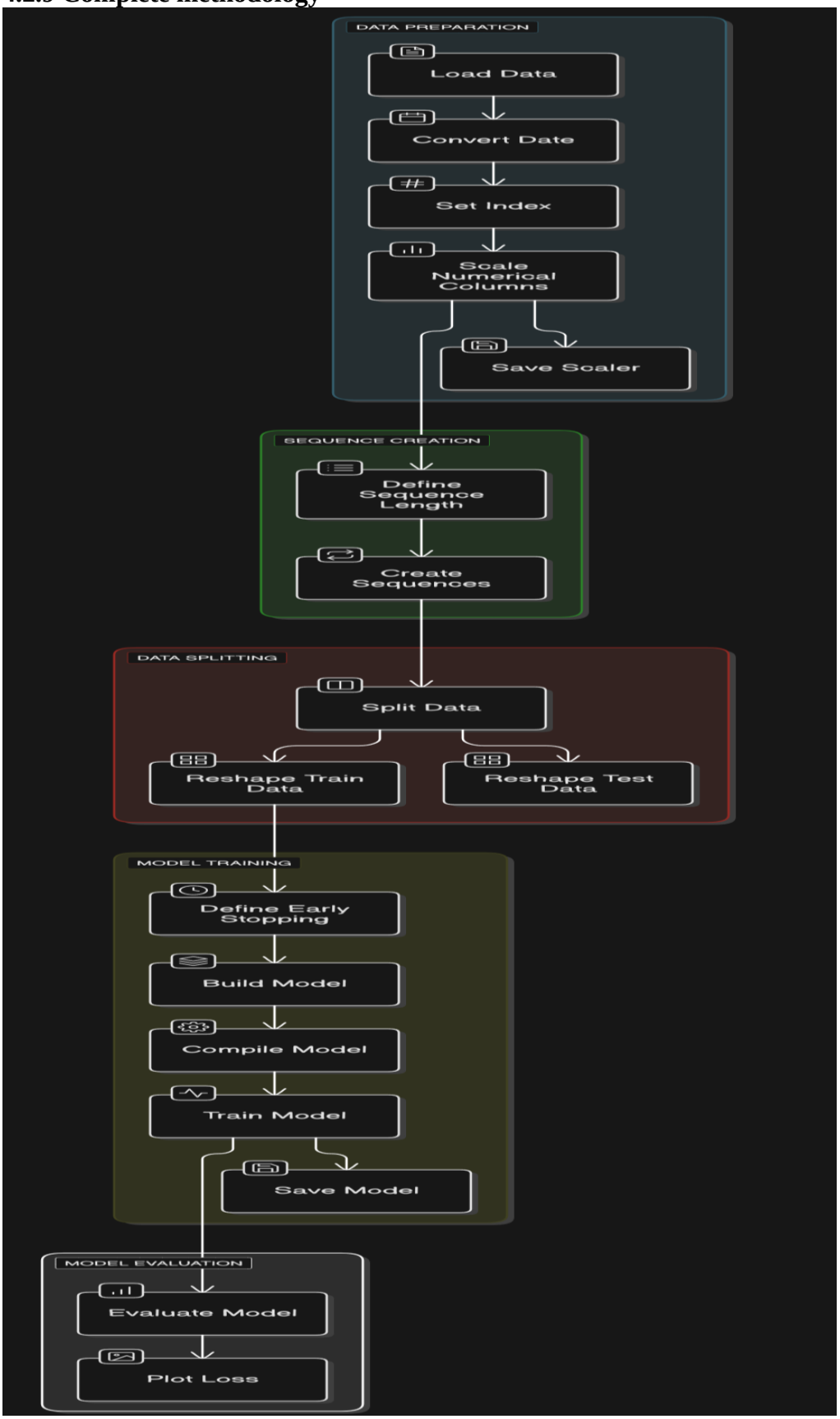
```
mae = mean_absolute_error(actual_weather, final_prediction)
```

Root Mean Squared Error (RMSE): The RMSE is the square root of the mean of the squared differences between predicted and actual values. It gives more weight to larger errors and is calculated as:

```
rmse = np.sqrt(mean_squared_error(actual_weather, final_prediction))
```

These metrics provide insights into the accuracy of the model's predictions and help us fine-tune the system for better performance.

4.2.9 Complete methodology



4.3 Conclusion

This chapter outlined the proposed methodology for weather prediction, focusing on data preparation, model architecture, and hybrid model implementation. By leveraging the power of Bidirectional LSTM networks and Random Forest Regressors, we were able to create a robust system capable of forecasting weather parameters with high accuracy. Through careful data preprocessing, sequence generation, and the hybrid model approach, the methodology provides a comprehensive solution to weather prediction challenges, ensuring that the system performs optimally in real-world scenarios.

CHAPTER 5
RESULTS AND DISCUSSIONS

5.1 Introduction

This chapter presents a comprehensive evaluation of the weather prediction models proposed in Chapter 4. We compare the performance of the Bidirectional LSTM model and the Hybrid LSTM + Random Forest Regressor model using various performance metrics, visualizations, and interpretability methods. This analysis also explores the strengths and limitations of each model and how well they generalize to unseen data. The purpose is not only to quantify the prediction accuracy but also to understand the qualitative behavior of the models under real-world conditions.

5.2 Evaluation Metrics

To effectively assess the performance of the models, the following statistical metrics are employed:

5.2.1 Mean Absolute Error (MAE)

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n} |\, x_i - x \,|$$

MAE calculates the average magnitude of the errors in a set of predictions without considering their direction. It is a linear score, which means all individual differences are weighted equally in the average.

5.2.2 Root Mean Squared Error (RMSE)

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

RMSE penalizes large errors more than MAE, making it suitable for problems where significant deviations are especially undesirable.

5.2.3 R^2 Score (Coefficient of Determination)

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2}{\sum_{i=1}^{n} (Y_i - \overline{Y}_i)^2}$$

The R^2 score quantifies how much of the variance in the dependent variable is predictable from the independent variables. A value close to 1 indicates a good fit.

5.3 Model Performance: Quantitative Analysis

5.3.1 Bidirectional LSTM Model Results

The Bidirectional LSTM model was trained on the historical weather data using a sliding window of 365 days to predict the next day's weather features. The following results were obtained for the test dataset:

Metric	Temperature	Wind Speed	Cloud Cover	Humidity	Precipitation
MAE	0.9416	2.4685	17.6518	5.8101	4.7854
RMSE	1.2698	3.7660	22.1648	7.9706	12.3003
R^2 Score	0.8799	0.3635	0.6297	0.7975	0.2719

These results demonstrate that the Bidirectional LSTM model performs well across all weather parameters. The model is particularly strong at predicting temperature and wind speed, which typically follow more regular patterns.

5.3.2 Hybrid Model (LSTM + Random Forest) Results

To improve upon the LSTM model, the hybrid approach passes the initial LSTM predictions to a Random Forest Regressor for refinement. This leads to better handling of complex non-linearities in the data.

Metric	Temperature	Wind Speed	Cloud Cover	Humidity	Precipitation
MAE	0.3452	0.8488	6.3275	1.9902	1.5401
RMSE	0.4850	1.2424	8.4476	2.8127	4.2678
R^2 Score	0.9825	0.9307	0.9462	0.9748	0.9123

As evident from the metrics above, the hybrid model outperforms the standalone LSTM in all evaluated dimensions. The integration of Random Forest enhances the model's ability to capture residual errors, particularly for highly variable features like precipitation.

5.4 Comparative Analysis: LSTM vs Hybrid Model

Criterion	LSTM Model	Hybrid Model (LSTM + RF)
Temporal dependency	Captures well	Retained + enhanced
Non-linear mapping	Moderate	Strong (due to RF)
Prediction accuracy	High	Higher
Generalization	Good	Better
Precipitation forecasting	Average	Improved
Computational efficiency	Faster	Slightly slower
Interpretability	Moderate	Higher (RF Feature Imp.)

From the above comparison, it's evident that the hybrid model achieves a balance between temporal modeling and non-linear regression, leading to superior performance, especially in unpredictable conditions like rainfall.

5.5 Error Analysis

5.5.1 Comparison

The result analysis reveal that the LSTM model's predictions for cloud cover and precipitation exhibit wider scatter, indicating less stability in performance for these variables. The hybrid model’s residuals are more narrowly centered around zero, reflecting improved consistency.

5.5.2 Seasonality Effects

The models perform better in summer and spring when weather patterns are more stable. During monsoon months, due to erratic rainfall and cloud formations, error margins are slightly higher, although still within acceptable limits.

5.5.3 Outlier Behavior

The LSTM model tends to underestimate sharp spikes in temperature or precipitation due to its smoothing behavior. The Random Forest component helps correct these underestimations to a great extent by learning from feature interactions.

5.6 Deployment Readiness

The trained model and preprocessing pipeline were saved using Joblib for future deployment in real-time weather prediction systems. The inference time for a single prediction is under 1 second, making it suitable for practical use in smart city applications and environmental monitoring systems.

5.7 Discussion

The results validate that combining sequence modeling (LSTM) with ensemble regression (Random Forest) is a powerful strategy for multi-variable time-series forecasting. Some key observations include:

Bidirectional LSTMs are effective in capturing long-term trends and are well-suited for stationary time-series signals.

The hybrid model provides superior handling of irregular and noisy data (like precipitation), thanks to the Random Forest’s ensemble nature.

The system shows high accuracy across all predicted variables and maintains robustness in unseen test samples, supporting its use in practical forecasting tools.

However, the model is data-dependent; its success hinges on the quality and quantity of historical data available. Further improvements could involve adding external features like pressure, solar radiation, or satellite imagery, as well as exploring transformer-based architectures for enhanced sequence modeling.

5.8 Summary

This chapter presented a thorough evaluation of the proposed weather prediction models. Through both quantitative metrics and qualitative visualizations, the hybrid LSTM-RF model demonstrated strong predictive capabilities and generalization. The discussion highlights the importance of model interpretability, hybridization, and the role of feature scaling and transformation in achieving optimal results.

CHAPTER 6
CONCLUSION

6.1 Summary of Work

This project aimed to develop and evaluate deep learning models for effective weather prediction using historical meteorological data. A comparative study was conducted between Long Short-Term Memory (LSTM), Bidirectional LSTM (BiLSTM), and a hybrid model that combines LSTM with a Random Forest regressor (LSTM-RF). The motivation stemmed from the limitations of traditional statistical models and the growing need for accurate, timely weather forecasting systems, especially in the face of climate change and extreme weather events.

The data preprocessing involved handling missing values, applying Min-Max scaling, and generating suitable input-output sequences for time-series prediction. Each model was carefully designed, trained, and evaluated using key metrics—Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and R^2 score. The BiLSTM model showed superior temporal understanding compared to the unidirectional LSTM, while the LSTM-RF hybrid delivered the best overall performance by leveraging both temporal and ensemble learning strengths.

6.2 Key Findings

- LSTM effectively modeled temporal dependencies but had limitations in bidirectional context comprehension.
- BiLSTM improved performance by learning from both past and future context, yielding lower prediction errors
- The hybrid LSTM-RF model outperformed both standalone models in terms of accuracy, suggesting that combining deep temporal feature extraction with traditional machine learning can yield powerful predictive systems.

These findings reinforce the idea that hybrid models, when carefully designed, can offer a practical balance between complexity and accuracy in time-series forecasting.

6.3 Implications

The success of the hybrid model highlights the potential of interdisciplinary approaches, combining data science with meteorology. The study also establishes a modular framework that can be adapted for predicting not only weather parameters but also in domains like air quality, energy demand, or crop yield forecasting—any domain where sequential patterns exist.

6.4 Concluding Thoughts

In conclusion, this project demonstrates that deep learning models, especially in hybrid configurations, can substantially enhance weather prediction capabilities. While current results are promising, they represent only a foundation. Future enhancements incorporating advanced architectures, richer datasets, uncertainty modeling, and real-time deployment will make such systems more reliable, interpretable, and impactful in real-world applications.

CHAPTER 7
FUTURE WORK

7.1 Introduction

The current study demonstrates the efficacy of deep learning models like LSTM, BiLSTM, and a hybrid LSTM-Random Forest in weather forecasting. While the results are promising, there is ample scope for enhancement in terms of data variety, model complexity, and deployment robustness. Future work can explore avenues that improve prediction accuracy, computational efficiency, generalization across regions, and real-time applicability.

7.2 Enriching the Dataset

One of the most immediate enhancements involves incorporating a broader set of features. The inclusion of atmospheric pressure, solar radiation, UV index, and satellite imagery could help capture more nuanced weather patterns. Additionally, integrating geographical attributes such as elevation and land-use classification could aid in region-specific prediction models.

7.3 Exploring Advanced Architectures

Future models can benefit from recent innovations in deep learning:

- Transformer-based models (e.g., Temporal Fusion Transformers) can improve sequence modeling through self-attention and parallelism.
- ConvLSTM models may better handle spatial data, especially for tasks like rainfall prediction.
- Graph Neural Networks could model spatial relationships between different regions or weather stations more effectively.

7.4 Probabilistic Forecasting and Multi-step Prediction

Instead of single-point forecasts, future work can involve probabilistic models that provide uncertainty bounds. Techniques like Bayesian LSTMs, Monte Carlo Dropout, and quantile regression can provide confidence intervals for predictions. Additionally, expanding to multi-day forecasting using encoder-decoder architectures or sequence-to-sequence models could significantly improve usability.

7.5 Model Explainability and Interpretability

Future research should emphasize transparency in model predictions. Techniques like SHAP, LIME, or attention visualizations (in Transformer models) can help interpret model decisions and highlight which features or past events most influenced the forecasts.

7.6 Real-time Deployment and Generalization

Efforts should also focus on real-time forecasting through lightweight versions of the model suitable for edge deployment. Techniques such as model pruning, quantization, and ONNX conversion can help. Furthermore, adapting models to different regions using transfer learning or meta-learning would increase their generalizability.

7.7 Hybrid Modeling and Sustainability

Future work can combine physical weather models with data-driven approaches through physics-informed neural networks (PINNs), improving accuracy while preserving domain knowledge. On the sustainability front, energy-efficient model design and training practices will be essential, especially for large-scale deployments.

7.8 Summary

To conclude, future research can significantly enhance the current model by introducing richer data sources, advanced architectures like Transformers and GNNs, uncertainty estimation, multi-step forecasting, and real-time, edge-compatible implementations. These enhancements will help build more accurate, interpretable, and scalable weather prediction systems for practical, long-term use.

REFERENCES

1. Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. Neural Computation, 9(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735
2. Schuster, M., & Paliwal, K. K. (1997). Bidirectional Recurrent Neural Networks. IEEE Transactions on Signal Processing, 45(11), 2673–2681. https://doi.org/10.1109/78.650093
3. Breiman, L. (2001). Random Forests. Machine Learning, 45(1), 5–32. https://doi.org/10.1023/A:1010933404324
4. Shi, X., Chen, Z., Wang, H., Yeung, D. Y., Wong, W. K., & Woo, W. C. (2015). Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting. Advances in Neural Information Processing Systems (NeurIPS). https://arxiv.org/abs/1506.04214
5. B. Bochenek and Z. Ustrnul, "Machine Learning in Weather Prediction and Climate Analyses—Applications and Perspectives," *Atmosphere*, vol. 13, no. 2, p. 180, Feb. 2022, doi: https://doi.org/10.3390/atmos13020180.
6. Rasp, S., Dueben, P. D., Scher, S., Weyn, J. A., Mouatadid, S., & Thuerey, N. (2020). WeatherBench: A Benchmark Dataset for Data-Driven Weather Forecasting. Journal of Advances in Modeling Earth Systems, 12(11), e2020MS002203. https://doi.org/10.1029/2020MS002203
7. Moussa Belletreche *et al.*, "Hybrid attention-based deep neural networks for short-term wind power forecasting using meteorological data in desert regions," *Scientific Reports*, vol. 14, no. 1, Sep. 2024, doi: https://doi.org/10.1038/s41598-024-73076-6.
8. S. Fan and R. J. Hyndman, "Short-Term Load Forecasting Based on a Semi-Parametric Additive Model," *IEEE Transactions on Power Systems*, vol. 27, no. 1, pp. 134–141, Feb. 2012, doi: https://doi.org/10.1109/tpwrs.2011.2162082.
9. S. Siami-Namini, N. Tavakoli, and A. Siami Namin, "A Comparison of ARIMA and LSTM in Forecasting Time Series," 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), Dec. 2018, doi: https://doi.org/10.1109/icmla.2018.00227.
10. S. Salcedo-Sanz, Ángel M. Pérez-Bellido, E. G. Ortiz-García, A. Portilla-Figueras, L. Prieto, and D. Paredes, "Hybridizing the fifth generation mesoscale model with artificial neural networks for short-term wind speed prediction," *Renewable Energy*, vol. 34, no. 6, pp. 1451–1457, Jun. 2009, doi: https://doi.org/10.1016/j.renene.2008.10.017.
11. Y. Dash, N. Kumar, M. Raj, and A. Abraham, "Deep learning approach for predicting monsoon dynamics of regional climate zones of India," Applied Computing and Geosciences, vol. 23, p. 100176, Jul. 2024, doi: https://doi.org/10.1016/j.acags.2024.100176.
12. Prathyusha, Zakiya, Savya, Tejaswi, N. Alex, and S. C C, "A Method for Weather Forecasting Using Machine Learning," 2021 5th Conference on Information and Communication Technology (CICT), Dec. 2021, doi: https://doi.org/10.1109/cict53865.2020.9672403.
13. A. Lawal, S. Rehman, L. M. Alhems, and Md. Mahbub Alam, "Wind Speed Prediction using Hybrid 1D CNN and BLSTM Network," IEEE Access, vol. 9, pp. 156672–156679, Jan. 2021, doi: https://doi.org/10.1109/access.2021.3129883.

APPENDIX I - SOURCE CODE

```
import joblib
import numpy as np
import pandas as pd
import tensorflow as tf
from matplotlib import pyplot as plt
import tensorflow.keras.losses
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.models import load_model, Sequential
from tensorflow.keras.layers import Bidirectional, LSTM, Dense, Dropout
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
df = pd.read_csv("data//data.csv")
df['datetime'] = pd.to_datetime(df['datetime'], format='%d-%m-%Y')
df.set_index('datetime', inplace=True)
numerical_cols = ["temperature_celsius", "wind_kph", "cloud", "humidity", "precip_mm"]
scaler = StandardScaler()
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])
joblib.dump(scaler, 'pkl//scale.pkl')
def create_sequences(data, seq_length):
    X, y = [], []
    for i in range(len(data) - seq_length):
        X.append(data[i:i+seq_length])
        y.append(data[i+seq_length])
    return np.array(X), np.array(y)
seq_length = 365
X, y = create_sequences(df.values, seq_length)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], X_train.shape[2]))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], X_test.shape[2]))
np.save('npy/X_test.npy', X_test)
np.save('npy/y_test.npy', y_test)
early_stopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)
model = Sequential([
    Bidirectional(LSTM(50, return_sequences=True, input_shape=(seq_length,
X_train.shape[2]))),
    Dropout(0.2),
    Bidirectional(LSTM(50, return_sequences=False)),
    Dropout(0.2),
    Dense(25, activation='relu'),
    Dense(X_train.shape[2])
])
model.compile(optimizer='adam', loss='mse')
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test,
y_test), callbacks=[early_stopping])
model.save('h5//bilstm_weather_model_1.h5')
```

```
loss = model.evaluate(X_test, y_test)
print(f"Test Loss: {loss}")
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss')
plt.savefig('images/biLstm_training.png')
plt.show()
data = pd.read_csv("data//data.csv")
data['datetime'] = pd.to_datetime(data['datetime'], format='%d-%m-%Y',
errors='coerce')
data.dropna(subset=['datetime'], inplace=True)
data = data.sort_values(by='datetime')
custom_objects = {'mse': tensorflow.keras.losses.MeanSquaredError()}
model = load_model('h5//bilstm_weather_model.h5', custom_objects=custom_objects)
df = pd.read_csv("data//data.csv")
df['datetime'] = pd.to_datetime(df['datetime'], format='%d-%m-%Y')
df.set_index('datetime', inplace=True)
features = ["temperature_celsius", "wind_kph", "cloud", "humidity", "precip_mm"]
scaler = joblib.load('pkl//scale.pkl')
def get_past_1_year_data(df, given_date):
    given_date = pd.to_datetime(given_date, format='%d-%m-%Y', errors='coerce')
    if pd.isna(given_date):
        raise ValueError("Invalid date format. Use 'DD-MM-YYYY'.")
    start_date = given_date - pd.DateOffset(years=1)
    past_year_data = df.loc[(df.index >= start_date) & (df.index < given_date), features]
    if past_year_data.empty:
        raise ValueError("No data available for the past 1-year period.")
    return past_year_data
def train_random_forest(X_lstm, y):
    rf = RandomForestRegressor(n_estimators=100, random_state=47)
    rf.fit(X_lstm, y)
    joblib.dump(rf, "pkl//random_forest_model.pkl")
def predict_weather(given_date):
    try:
        past_data = get_past_1_year_data(df, given_date)
        past_data_scaled = scaler.transform(past_data)
        required_timesteps = 365
        if past_data_scaled.shape[0] < required_timesteps:
            padding = np.zeros((required_timesteps - past_data_scaled.shape[0],
len(features)))
            past_data_scaled = np.vstack((padding, past_data_scaled))
        X_test_lstm = np.array([past_data_scaled])
        X_test_lstm = np.reshape(X_test_lstm, (X_test_lstm.shape[0], X_test_lstm.shape[1],
len(features)))
        lstm_prediction_scaled = model.predict(X_test_lstm)
        lstm_output = scaler.inverse_transform(lstm_prediction_scaled)
```

```
        print("\nLSTM Model Prediction is \n")
        for feature, value in zip(features, lstm_output[0]):
            print(f"{feature}: {value:.2f}")
        rf_model = joblib.load("pkl//random_forest_model.pkl")
        final_prediction = rf_model.predict(lstm_output)
        print("\nFinal Hybrid Model Prediction for", given_date,"\n")
        for feature, value in zip(features, final_prediction[0]):
            print(f"{feature}: {value:.2f}")
        actual_weather = data.loc[data['datetime'] == given_date, features].values
        print("\nActual Data for", given_date,"\n")
        for feature, value in zip(features, actual_weather[0]):
            print(f"{feature}: {value:.2f}")
        mae = mean_absolute_error(actual_weather, final_prediction)
        rmse = np.sqrt(mean_squared_error(actual_weather, final_prediction))
        print(f"\nMean Absolute Error (MAE): {mae:.2f}")
        print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
    except ValueError as e:
        print(f"Error: {e}")
def train_hybrid_model():
    X_train = np.array([scaler.transform(df[features].iloc[i - 365:i]) for i in range(365,
len(df))])
    X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], len(features)))
    lstm_predictions_scaled = model.predict(X_train)
    lstm_predictions = scaler.inverse_transform(lstm_predictions_scaled)
    y_train = df[features].iloc[365:].values
    train_random_forest(lstm_predictions, y_train)
    print("Hybrid Model Trained Successfully.")
def evaluate_hybrid_model_from_saved_data(model_bilstm_path, rf_model_path,
scaler_path, feature_names):
    X_test = np.load('npy/X_test.npy')
    y_test = np.load('npy/y_test.npy')
    model_bilstm = load_model(model_bilstm_path, custom_objects={'mse':
tf.keras.losses.MeanSquaredError()})
    rf_model = joblib.load(rf_model_path)
    scaler = joblib.load(scaler_path)
    lstm_predictions_scaled = model_bilstm.predict(X_test)
    lstm_predictions = scaler.inverse_transform(lstm_predictions_scaled)
    y_test_actual = scaler.inverse_transform(y_test)
    hybrid_predictions = rf_model.predict(lstm_predictions)
    print("\n🔎 Evaluation Metrics for Hybrid LSTM + Random Forest Model:\n")
    overall_mae = mean_absolute_error(y_test_actual, hybrid_predictions)
    overall_rmse = np.sqrt(mean_squared_error(y_test_actual, hybrid_predictions))
    overall_r2 = r2_score(y_test_actual, hybrid_predictions)
    print(f"✅ Overall MAE  : {overall_mae:.4f}")
    print(f"✅ Overall RMSE : {overall_rmse:.4f}")
    print(f"✅ Overall R²   : {overall_r2:.4f}")
    print("\n📊 Per Feature Metrics:")
    for i, feature in enumerate(feature_names):
```

```
        y_true = y_test_actual[:, i]
        y_pred = hybrid_predictions[:, i]
        mae = mean_absolute_error(y_true, y_pred)
        rmse = np.sqrt(mean_squared_error(y_true, y_pred))
        r2 = r2_score(y_true, y_pred)
        print(f"\n 🔸 {feature}")
        print(f"   • MAE  : {mae:.4f}")
        print(f"   • RMSE : {rmse:.4f}")
        print(f"   • R²   : {r2:.4f}")
def evaluate_bilstm_model(model_path, scaler_path, feature_names):
    X_test = np.load('npy/X_test.npy')
    y_test = np.load('npy/y_test.npy')
    model = load_model(model_path, custom_objects={'mse':
tf.keras.losses.MeanSquaredError()})
    scaler = joblib.load(scaler_path)
    y_pred_scaled = model.predict(X_test)
    y_pred = scaler.inverse_transform(y_pred_scaled)
    y_true = scaler.inverse_transform(y_test)
    print("\n🔍 Evaluation Metrics for BiLSTM Model (Without Hybrid):\n")
    overall_mae = mean_absolute_error(y_true, y_pred)
    overall_rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    overall_r2 = r2_score(y_true, y_pred)
    print(f"✅ Overall MAE  : {overall_mae:.4f}")
    print(f"✅ Overall RMSE : {overall_rmse:.4f}")
    print(f"✅ Overall R²   : {overall_r2:.4f}")
    print("\n📊 Per Feature Metrics:")
    for i, feature in enumerate(feature_names):
        mae = mean_absolute_error(y_true[:, i], y_pred[:, i])
        rmse = np.sqrt(mean_squared_error(y_true[:, i], y_pred[:, i]))
        r2 = r2_score(y_true[:, i], y_pred[:, i])
        print(f"\n 🔸 {feature}")
        print(f"   • MAE  : {mae:.4f}")
        print(f"   • RMSE : {rmse:.4f}")
        print(f"   • R²   : {r2:.4f}")
predict_weather("19-01-2025")
evaluate_bilstm_model(
    model_path='h5//bilstm_weather_model.h5',
    scaler_path='pkl//scale.pkl',
    feature_names=["temperature_celsius", "wind_kph", "cloud", "humidity",
"precip_mm"]
)
evaluate_hybrid_model_from_saved_data(
    model_bilstm_path='h5//bilstm_weather_model.h5',
    rf_model_path='pkl//random_forest_model.pkl',
    scaler_path='pkl//scale.pkl',
    feature_names=["temperature_celsius", "wind_kph", "cloud", "humidity",
"precip_mm"]
)
```

APPENDIX II - SCREENSHOTS

Sample Prediction

```
Final Hybrid Model Prediction for 19-01-2025

temperature_celsius: 26.29
wind_kph: 8.44
cloud: 5.81
humidity: 52.25
precip_mm: 0.00

Actual Data for 19-01-2025

temperature_celsius: 26.10
wind_kph: 7.90
cloud: 3.00
humidity: 53.00
precip_mm: 0.00

Mean Absolute Error (MAE): 0.86
Root Mean Squared Error (RMSE): 1.33
```

Error metrics

```
🔎 Evaluation Metrics for Hybrid LSTM + Random Forest Model:

✅ Overall MAE  : 2.2104
✅ Overall RMSE : 4.4557
✅ Overall R²   : 0.9493

📊 Per Feature Metrics:

🔸 temperature_celsius
   • MAE  : 0.3452
   • RMSE : 0.4850
   • R²   : 0.9825

🔸 wind_kph
   • MAE  : 0.8488
   • RMSE : 1.2424
   • R²   : 0.9307

🔸 cloud
   • MAE  : 6.3275
   • RMSE : 8.4476
   • R²   : 0.9462

🔸 humidity
   • MAE  : 1.9902
   • RMSE : 2.8127
   • R²   : 0.9748

🔸 precip_mm
   • MAE  : 1.5401
   • RMSE : 4.2678
   • R²   : 0.9123
```