A Project Report on

SHORT-TERM WIND POWER PREDICTION USING NUMERICAL WEATHER PREDICTIONS AND RESIDUAL CONVOLUTIONAL LONG SHORT-TERM MEMORY ATTENTION NETWORK

submitted in partial fulfillment of the requirements
for the award of the degree

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

Submitted by

MOTURI VEERA VENKATA KAVYA (21021A0501)

PUNYAMURTHULA LAKSHMI SAHITHI (21021A0502)

DUGGIRALA VENKATA BHARGAVA REDDY (21021A0512)

AKULA SAI SRIRAM (21021A0543)

Under the supervision of

Dr. SSSN. USHA DEVI N.

Assistant Professor, Department of CSE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIVERSITY COLLEGE OF ENGINEERING KAKINADA(A)

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY

KAKINADA, KAKINADA-533003, ANDHRA PRADESH, INDIA

2021-2025

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
UNIVERSITY COLLEGE OF ENGINEERING KAKINADA (A)
JAWHARLAL NEHRU TECHNOLOGICAL UNIVERSITY
KAKINADA, KAKINADA-533003, ANDHRA PRADESH, INDIA

CERTIFICATE

This is to certify that the project titled **"SHORT-TERM WIND POWER PREDICTION USING NUMERICAL WEATHER PREDICTIONS AND RESIDUAL CONVOLUTIONAL LONG SHORT-TERM MEMORY ATTENTION NETWORK"** submitted by **MOTURI VEERA VENKATA KAVYA (21021A0501), PUNYAMURTHULA LAKSHMI SAHITHI (21021A0502), DUGGIRALA VENKATA BHARGAVA REDDY (21021A0512), AKULA SAI SRIRAM (21021A0543)**, students of B.Tech(CSE) to University College of Engineering, Jawaharlal Nehru Technological University Kakinada, in partial fulfillment of the requirements for the award of the degree of **BACHELOR OF TECHNOLOGY** in **COMPUTER SCIENCE AND ENGINEERING** is a record of bonafide work carried out by him/her 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.

Signature of Supervisor

Dr. SSSN. Usha Devi N.

Assistant Professor

Department of CSE

UCEK(A)

JNTUK, Kakinada

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
UNIVERSITY COLLEGE OF ENGINEERING KAKINADA (A)
JAWHARLAL NEHRU TECHNOLOGICAL UNIVERSITY
KAKINADA, KAKINADA-533003, ANDHRA PRADESH, INDIA

CERTIFICATE

This is to certify that the project titled **"SHORT-TERM WIND POWER PREDICTION USING NUMERICAL WEATHER PREDICTIONS AND RESIDUAL CONVOLUTIONAL LONG SHORT-TERM MEMORY ATTENTION NETWORK"** submitted by **MOTURI VEERA VENKATA KAVYA (21021A0501), PUNYAMURTHULA LAKSHMI SAHITHI (21021A0502), DUGGIRALA VENKATA BHARGAVA REDDY (21021A0512), AKULA SAI SRIRAM (21021A0543)**, students of B.Tech(CSE) to University College of Engineering, Jawaharlal Nehru Technological University Kakinada, in partial fulfillment of the requirements for the award of the degree of **BACHELOR OF TECHNOLOGY** in **COMPUTER SCIENCE AND ENGINEERING** is a record of bonafide work carried out by him/her 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.

Signature of Head of the Department

Dr. N. Ramakrishnaiah

Professor & HOD

Department of CSE

UCEK(A)

JNTUK, Kakinada

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
UNIVERSITY COLLEGE OF ENGINEERING KAKINADA (A)
JAWHARLAL NEHRU TECHNOLOGICAL UNIVERSITY
KAKINADA, KAKINADA-533003, ANDHRA PRADESH, INDIA

DECLARATION

We hereby declare that the project report entitled **"SHORT-TERM WIND POWER PREDICTION USING NUMERICAL WEATHER PREDICTIONS AND RESIDUAL CONVOLUTIONAL LONG SHORT-TERM MEMORY ATTENTION NETWORK"** submitted by me to University College of Engineering, Jawaharlal Nehru Technological University Kakinada in partial fulfillment of the requirement for the award of the degree of **Bachelor of Technology** in Computer Science and Engineering is a record of bonafide project work carried out by me under the guidance of **Dr. SSSN. Usha Devi N.**, Assistant Professor in CSE. We also declare that this project is a result of my effort and that it has not been copied from anyone, and we have taken only citations from the sources mentioned in the references. We 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 in this institute or any other institute or university.

MOTURI VEERA VENKATA KAVYA (21021A0501)

PUNYAMURTHULA LAKSHMI SAHITHI (21021A0502)

DUGGIRALA VENKATA BHARGAVA REDDY (21021A0512)

AKULA SAI SRIRAM (21021A0543)

ACKNOWLEDGEMENT

We want to express our deepest gratitude to the following people for guiding us through this course and without whom this project and its results would not have been completed.

We wish to thank first our supervisor **Dr. SSSN. Usha Devi N.**, Assistant Professor, Department of CSE for accepting and supporting us as a project team. We sincerely convey our gratefulness and heartfelt to our honorable and esteemed project guide for her supervision, guidance, encouragement, and counsel throughout my project. Without her invaluable advice and assistance, it would not have been possible for us to complete this project.

We would like to thank **Dr. N. Ramakrishnaiah**, Professor, and Head of the Computer Science and Engineering department, UCEK, JNTUK for his fabulous support and useful remarks throughout the project.

We are thankful to **Dr. N. Mohan Rao**, Principal, University college of engineering Kakinada (Autonomous), JNTUK for his support and enlightenment during the project.

We are thankful to **Dr. A. Karuna**, and to all the PRC members for their continuous suggestions throughout the project.

We thank all the teaching and non-teaching staff of the Department of Computer Science and Engineering for their support throughout our project work.

MOTURI VEERA VENKATA KAVYA (21021A0501)

PUNYAMURTHULA LAKSHMI SAHITHI (21021A0502)

DUGGIRALA VENKATA BHARGAVA REDDY (21021A0512)

AKULA SAI SRIRAM (21021A0543)

TABLE OF CONTENTS

List of Figures

ACRONYMS AND ABBREVIATION

AI	-	Artificial Intelligence
API	-	Application Programming Interface
ARIMA	-	ARIMA Auto-Regressive Integrated Moving Average
CNN	-	Convolutional Neural Network
CSV	-	Comma Separated Values
CSE	-	Computer Science and Engineering
DNN	-	Deep Neural Network
ECMWF	-	European Centre for Medium-Range Weather Forecasts
GPU	-	Graphics Processing Unit
GRU	-	Gated Recurrent Unit
GUI	-	Graphical User Interface
IDE	-	Integrated Development Environment
IoT	-	Internet of Things
LSTM	-	Long Short-Term Memory
MAE	-	Mean Absolute Error
ML	-	Machine Learning
MSE	-	Mean Squared Error
NREL	-	National Renewable Energy Laboratory
NWP	-	Numerical Weather Prediction
OCR	-	Optical Character Recognition
ReLU	-	Rectified Linear Unit
RNN	-	Recurrent Neural Network
SCADA	-	Supervisory Control and Data Acquisition
SVM	-	Support Vector Machine
VSC	-	Visual Studio Code

ABSTRACT

The **Wind Power Forecasting Model** project aims to address the growing need for accurate and reliable prediction of wind power generation using deep learning techniques. As renewable energy sources become increasingly vital, forecasting wind power helps optimize energy grid management, enhance reliability, and reduce operational costs. This project leverages historical meteorological data, wind farm measurements, and a novel hybrid deep learning architecture that integrates Residual ConvLSTM layers with an attention mechanism. The model processes both historical sequence data and current weather features, learning complex temporal dependencies and spatial correlations within the data. Feature engineering includes the creation of temporal attributes and vector-based wind components, enhancing the model's ability to interpret wind behavior. The hybrid architecture fuses Conv1D, LSTM, GRU, and attention layers, ensuring robust feature learning and interpretability. Evaluation metrics such as Mean Squared Error (MSE) and Mean Absolute Error (MAE) were used to assess performance, and visualization tools were employed to track model learning and accuracy. In conclusion, the Res-ConvLSTM-Attention model demonstrates a promising approach to wind power prediction, offering improved forecast accuracy and supporting smarter data-driven energy systems.

Keywords: Wind Power Forecasting, Deep Learning, ConvLSTM, Attention Mechanism, Time Series, Renewable Energy

Chapter 1

Introduction

1.1 Wind Power Forecasting and Its Importance

With the rapid shift toward sustainable energy sources, wind energy has emerged as a major contributor to global electricity generation. Wind power, being variable and intermittent by nature, poses a significant challenge to energy grid management. Accurate wind power forecasting is essential to ensure the balance between supply and demand, avoid energy curtailment, and improve the stability and reliability of power systems. Forecasting enables energy providers to make informed decisions regarding energy trading, load balancing, and operational planning, ultimately enhancing the overall efficiency of the power grid. The integration of data-driven models and deep learning techniques in forecasting wind power has gained significant traction. Traditional statistical methods often fail to capture the complex, non-linear patterns inherent in meteorological data. Therefore, advanced models that can leverage temporal sequences and spatial correlations in wind patterns have become critical for precise forecasting.

1.2 Deep Learning in Wind Forecasting

Deep learning, a subfield of machine learning, has shown exceptional performance in handling large-scale, complex datasets. It relies on artificial neural networks with multiple layers to learn data representations at varying levels of abstraction. In the context of wind forecasting, deep learning models such as LSTM (Long Short-Term Memory), GRU (Gated Recurrent Units), and ConvLSTM (Convolutional LSTM) are capable of modeling sequential dependencies in time-series data.

These models automatically extract meaningful features from meteorological and

wind farm datasets, enabling the prediction of future wind power with high accuracy. By stacking multiple layers and combining temporal modeling with attention mechanisms, deep learning significantly outperforms conventional forecasting techniques.

1.3 Why Deep Learning for Wind Power Forecasting

The dynamic and non-linear nature of wind patterns demands models that can understand long-term dependencies and handle noisy data. Deep learning models are chosen for the following reasons:

1. High Predictive Accuracy: They capture intricate temporal patterns, leading to more accurate forecasts.

2. Automatic Feature Extraction: There is no need for extensive manual feature engineering, as deep networks learn relevant patterns during training of data, making them more data-efficient than traditional machine learning algorithms.

3. Scalability: Deep learning models can handle large volumes of data from multiple wind farms and meteorological sources. engineering.

4. Adaptability: These models can adapt to different geographic regions and wind farm configurations by retraining on local datasets.

The adoption of deep learning for wind power forecasting supports the development of smart grids and data-driven decision-making in the renewable energy sector.

1.4 Sequence Modeling in Forecasting

In this project, a hybrid deep learning architecture combining Residual ConvLSTM and Attention layers is used. This architecture enables the model to:

- Learn from past wind conditions through historical sequence data.
- Integrate current and forecast weather features to predict near-future power produc-

tion.

• Focus on relevant temporal features using attention mechanisms, improving the model's interpretability and accuracy.

The model processes inputs from historical meteorological variables and wind farm performance data, generating accurate power production forecasts. The two-branch input structure ensures that both long-term trends and short-term conditions are captured effectively.

1.5 Role of Hybrid Architectures in Time-Series Forecasting

The hybrid Res-ConvLSTM-Attention model plays a pivotal role in enhancing wind power forecasting. By integrating multiple deep learning components:

• Conv1D and LSTM layers extract and model temporal sequences.

• Residual connections help in better gradient flow and prevent vanishing gradients during training.

• Attention mechanisms focus on significant time steps, reducing noise and improving interpretability.

• GRU layers further refine the sequence features and enhance performance.

This combination provides a robust foundation for making accurate predictions under varying wind conditions, making it a valuable tool for energy providers aiming to maximize the efficiency of wind energy utilization.

Chapter 2

Review of Literature

Wind power forecasting has garnered increasing research interest due to the critical role it plays in integrating renewable energy sources into the existing power grid infrastructure. Accurate and timely forecasts are essential not only for ensuring grid stability but also for facilitating efficient load planning and maximizing the potential of energy trading in deregulated markets. The inherent variability and intermittency of wind make it challenging to rely on as a consistent energy source, thereby increasing the importance of dependable forecasting systems. Numerous studies have been conducted over the years exploring a range of techniques—both statistical and machine learning-based—that aim to improve the accuracy and robustness of wind power predictions [1].

Initially, forecasting approaches were primarily grounded in physical models and classical statistical time-series methods. Examples include Autoregressive Integrated Moving Average (ARIMA) and exponential smoothing models, which leveraged historical patterns in wind speed and power output to make short-term predictions [2]. While these methods provided a foundational framework for time-series forecasting, they frequently fell short when dealing with the highly non-linear, chaotic, and stochastic characteristics of wind dynamics. This shortcoming was especially pronounced during rapidly changing weather conditions, where traditional models lacked the flexibility and adaptability required for accurate forecasts [3].

The evolution of machine learning (ML) provided a significant boost to the field. Algorithms such as Random Forests, Support Vector Machines (SVMs), and Gradient Boosting Machines were adopted due to their ability to model non-linear relationships

and handle large datasets without strong statistical assumptions [4]. These techniques showed notable improvements over traditional methods, particularly when combined with advanced feature engineering strategies and ensemble learning frameworks that aggregated predictions from multiple base models to enhance generalization and reduce overfitting [5]. Nevertheless, ML models still faced challenges in capturing long-term temporal dependencies and sequential patterns inherent in meteorological and power generation data [6].

The introduction of deep learning revolutionized wind power forecasting by addressing the limitations of conventional ML models. Recurrent Neural Networks (RNNs), specifically Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks, were increasingly adopted due to their inherent capability to learn from sequential data over extended time horizons [7]. These architectures allowed models to retain memory of past inputs, making them particularly suitable for handling time-dependent variations in wind patterns. To further enhance performance, researchers introduced convolutional elements, resulting in architectures such as ConvLSTM, which combines convolutional layers for spatial feature extraction with LSTM layers for temporal modeling [8]. This hybrid approach allowed the model to capture both spatial correlations across different locations (e.g., wind turbines or regions) and temporal trends over time. Empirical studies have demonstrated that ConvLSTM-based models consistently outperform standard LSTM networks in scenarios where spatial dependencies are crucial, such as in geographically dispersed wind farms [9].

In addition to architectural innovations, the integration of attention mechanisms into deep sequential models has further improved the precision and robustness of wind power forecasts [10]. Attention mechanisms enable models to dynamically weigh the importance of different time steps in the input sequence, thereby focusing more on rel-

evant patterns while minimizing the influence of noise and irrelevant historical data. This selective focus has proven particularly advantageous in multi-input and multi-step forecasting scenarios, where diverse types of input features and forecast horizons are involved [11]. Attention not only enhances interpretability by revealing which parts of the input the model relies on but also contributes to improved generalization across varying meteorological conditions.

Recent literature has increasingly highlighted the value of hybrid deep learning architectures that combine multiple modeling paradigms. Such architectures typically integrate convolutional, recurrent, and attention modules to leverage the strengths of each component [12]. For instance, stacking Conv1D layers (for capturing local temporal trends), LSTM or GRU layers (for sequential learning), and attention layers (for contextual focus) creates a unified framework capable of capturing complex dependencies across multiple dimensions. These hybrid models have demonstrated strong performance in renewable energy forecasting tasks by effectively modeling intricate relationships among input variables and adapting to dynamic environmental conditions [13].

In the context of this project, the proposed use of a Res-ConvLSTM-Attention network is consistent with the most advanced and effective methodologies currently found in the field [14]. The inclusion of residual connections facilitates better gradient flow during training, thereby mitigating vanishing gradient problems and allowing the construction of deeper networks. When combined with temporal attention mechanisms, the architecture gains enhanced capability to learn long-term dependencies and adapt to diverse and changing patterns of wind behavior. This adaptability is particularly important in real-world applications where wind conditions can vary significantly across different geographic locations and timescales [15].

Overall, the reviewed literature underscores that the most promising advancements in wind power forecasting stem from the use of hybrid deep learning models, particularly those supported by rich feature engineering and rigorous training methodologies. These models, when supplemented with high-quality meteorological inputs and comprehensive wind farm metadata (e.g., turbine type, location, altitude), are capable of producing significantly more accurate and reliable forecasts. The continued refinement of these techniques will play a pivotal role in facilitating the broader adoption of wind energy in the global transition toward sustainable and resilient power systems [16].

Chapter 3

About the Environment

3.1 Visual Studio Code

Visual Studio Code (VS Code) is a highly versatile and popular source-code editor developed by Microsoft. Launched in 2015, it has quickly gained traction among developers due to its lightweight yet powerful features, extensive customization options, and support for a wide range of programming languages and frameworks. One of its standout features is its robust built-in IntelliSense, which provides intelligent code completion, syntax highlighting, and code navigation capabilities, enhancing developers' productivity. Its vibrant ecosystem of extensions allows users to tailor the editor to their specific needs, whether it's for web development, mobile app development, data science, or any other domain. With its frequent updates, active community support, and cross-platform compatibility, Visual Studio Code continues to be a go-to choice for developers seeking an efficient and customizable coding environment.

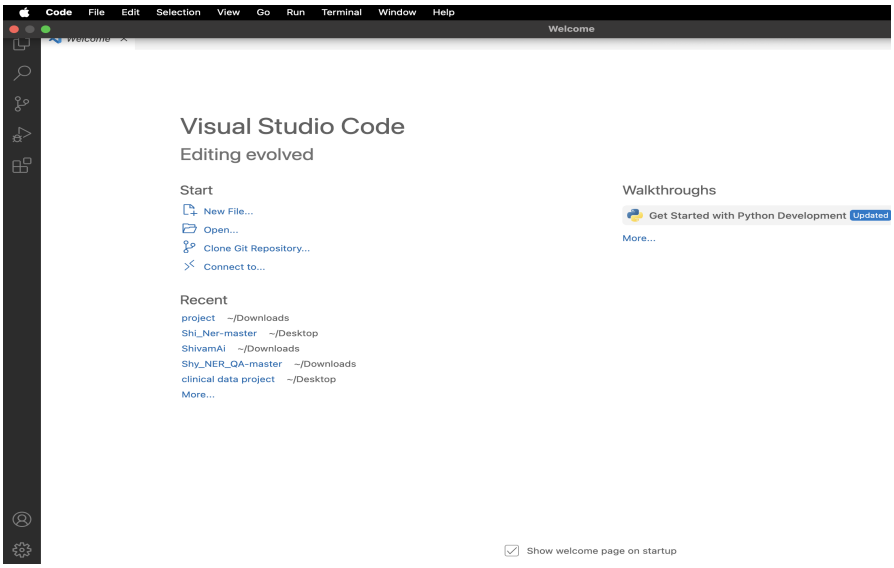

Figure 3.1: Visual Studio Code

Visual Studio Code (VSCode) Fig.3.1 is a multifaceted tool that serves as a go-to platform for developers across a spectrum of tasks. Primarily, it offers a rich code editing environment, boasting features like syntax highlighting, IntelliSense for smart code completion, and seamless navigation through large codebases. Its integration with version control systems like Git streamlines the management of source code, allowing developers to track changes and collaborate effectively. Debugging capabilities within VSCode empower developers to identify and resolve issues swiftly by setting breakpoints, inspecting variables, and stepping through code execution. The extensive library of extensions further enhances its functionality, enabling developers to tailor their environment with language support, debugging tools, and productivity enhancements. The integrated terminal facilitates seamless interaction with the command line, while task automation features streamline repetitive workflows like building, testing, and deployment. Additionally, VSCode's support for remote development and real-time collaboration through Live Share fosters teamwork and flexibility, empowering developers to work efficiently across different environments and collaborate seamlessly regardless of geographic location. With its high degree of customization, VSCode adapts to individual preferences and workflows, making it an indispensable tool in the arsenal of modern developers.

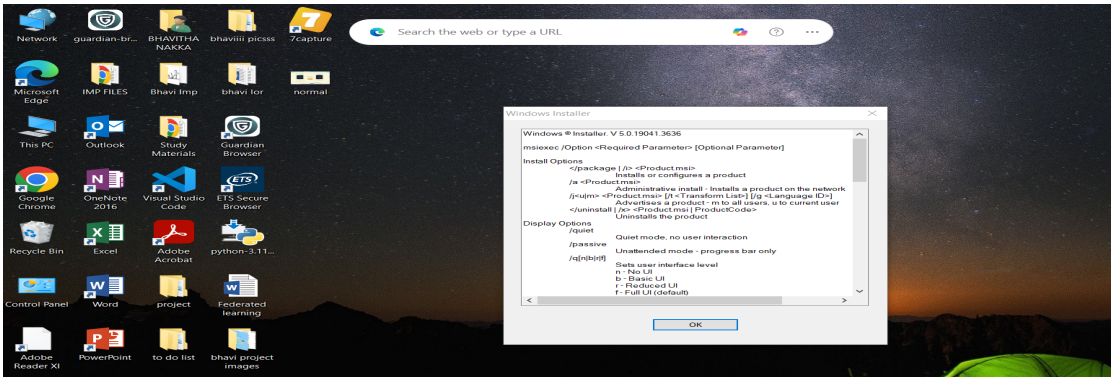

Figure 3.2: Fast and Smart Coding

For easy and efficient project management in VSCode Fig.3.2, utilize its integrated

features and extensions . Start by leveraging source control with Git directly within VSCode, allowing for seamless tracking of changes and collaboration with team members. Additionally, VS Code offers seamless integration with version control systems like Git, enabling smooth collaboration and code management workflows. Utilize the integrated terminal to run commands and scripts without leaving the editor, streamlining tasks like building, testing, and deployment. Take advantage of task automation using task runners like Gulp or Grunt to automate repetitive workflows. Additionally, VSCode's extensive library of extensions provides tools for project management Fig., such as project organization, task tracking, and issue management. With these features and extensions, developers can maintain an organized and productive workflow within VSCode, enhancing efficiency and facilitating smoother project management processes.

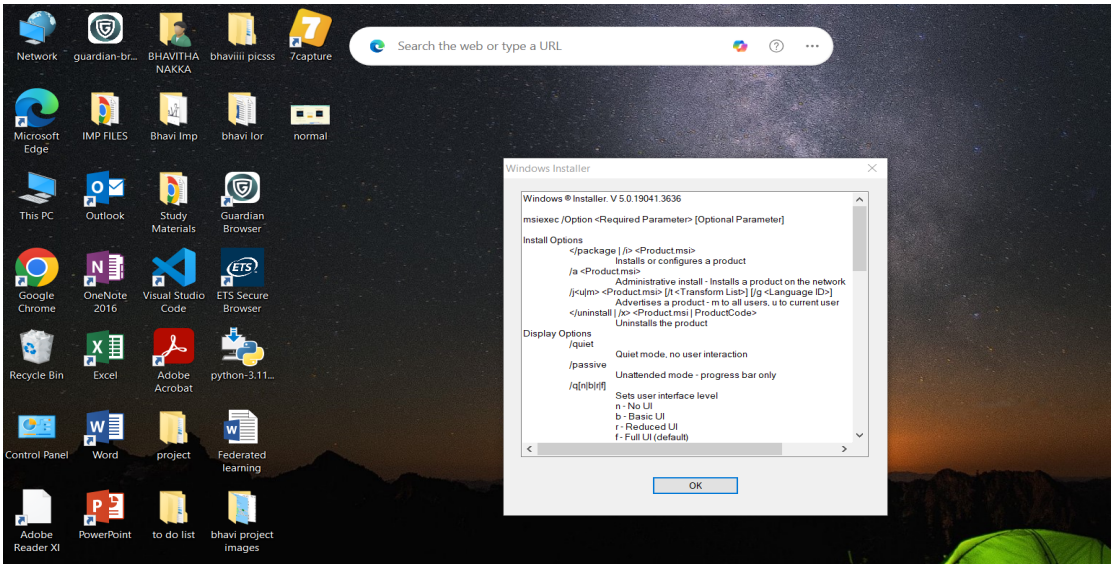

Figure 3.3: Efficient Project Management

3.2 Python Programming Language

Python is a high-level, interpreted programming language known for its simplicity, readability, and versatility. It was created by Guido van Rossum and first released in

1991. Python has gained immense popularity and has become one of the most widely used programming languages in various domains, including web development, data science, machine learning, scientific computing, automation, and more.

Some buzzwords associated with Python:

1.Readability: Python's syntax is designed to be clear and readable, making it easy for developers to write and understand code.

2.Simplicity: Python emphasizes simplicity and minimalism, with a clean and straightforward syntax that reduces the amount of code needed to express concepts compared to other languages.

3.Versatility: Python can be used for a wide range of applications, including web development, data analysis, machine learning, scientific computing, automation, scripting, and more.

4.Interpreted: Python is an interpreted language, meaning that code is executed line by line, making it easy to debug and test interactively.

5.Dynamic Typing: Python uses dynamic typing, allowing variables to be assigned without specifying their type explicitly. This flexibility simplifies development but requires careful attention to variable types.

6.Strong Typing: Despite dynamic typing, Python is strongly typed, meaning that types are enforced during runtime, reducing the likelihood of type-related errors.

7.Extensive Standard Library: Python comes with a comprehensive standard library that provides modules and packages for performing a wide range of tasks, from file I/O to networking to database access.

8.Large Ecosystem of Libraries: Python has a rich ecosystem of third-party libraries and frameworks that extend its functionality for specific use cases, including Django and Flask for web development, NumPy and pandas for data analysis, TensorFlow and PyTorch for machine learning, and many more.

9.Community Support: Python has a vibrant and active community of developers who contribute to its development, create libraries and tools, and provide support through forums, mailing lists, and online resources.

10.Cross-Platform Compatibility: Python code can run on various platforms and operating systems, including Windows, macOS, and Linux, making it highly portable and accessible.

11.Open Source: Python is open source, with its source code freely available for modification and redistribution, fostering collaboration and innovation within the developer community.

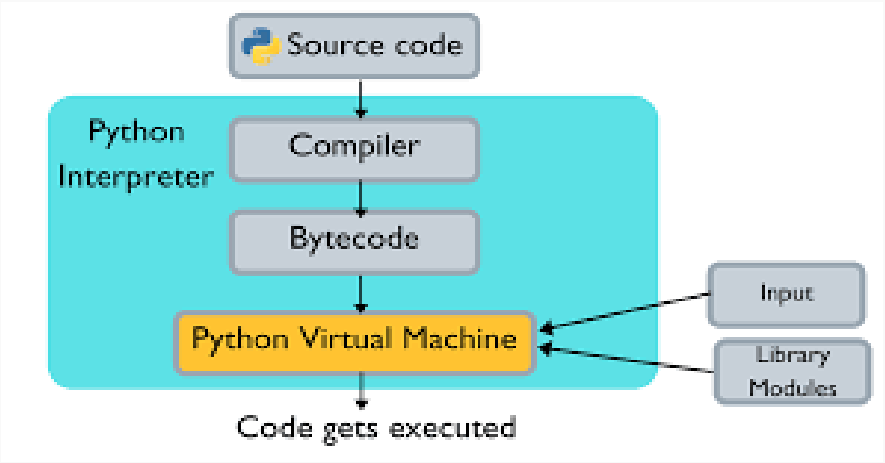

Figure 3.4: Compilation and Execution of PYTHON program

In Python, code compilation and execution are handled by the Python interpreter. Python is an interpreted language, which means that code is executed line by line without the need for explicit compilation into machine code. As shown in Fig.3.4 when a Python script is run, the interpreter reads the source code, parses it, and executes it directly, converting each statement into machine code on the fly. This dynamic execution process allows for rapid development and testing, as changes to the code can be immediately reflected in the program's behavior without the need for recompilation.

Python code can be compiled and executed swiftly, often requiring just a few lines to achieve complex tasks. Utilizing libraries like Pandas, NumPy, or requests, data re-

trieval and manipulation become straightforward. To compile and run Python code, one can use an Integrated Development Environment (IDE) like PyCharm or Jupyter Notebook, or simply execute scripts directly in a terminal or command prompt. With Python's readability and extensive documentation, writing concise and efficient code is achievable even for beginners. By breaking down tasks into smaller functions and leveraging built-in modules, code can remain succinct while maintaining functionality. Moreover, Python's versatility extends to web scraping, data analysis, machine learning, and more, making it a preferred choice for various applications. In essence, Python's simplicity and power enable developers to create impactful solutions with minimal lines of code, fostering rapid development and innovation across domains.

3.3 TensorFlow and Keras Frameworks

TensorFlow and Keras formed the backbone of the deep learning implementation in this project (in Fig.3.5). TensorFlow, developed by Google Brain, is a powerful open-source machine learning library designed for large-scale numerical computations and deep neural network training. TensorFlow provided the robust backend infrastructure, enabling efficient GPU utilization and acceleration for intensive computations. Keras, an intuitive high-level neural network API built on TensorFlow, greatly simplified model implementation. Its clear, easy-to-use functions facilitated rapid construction, experimentation, and validation of advanced model architectures including Conv1D, LSTM, GRU, and attention-based layers. Keras also offered straightforward mechanisms for defining model architectures, compiling models, performing optimization, handling early stopping, and checkpointing. The seamless integration between TensorFlow and Keras significantly streamlined the deep learning workflow, providing a stable and scalable environment ideal for developing and deploying the hybrid Res-ConvLSTM-Attention model utilized in this forecasting project.

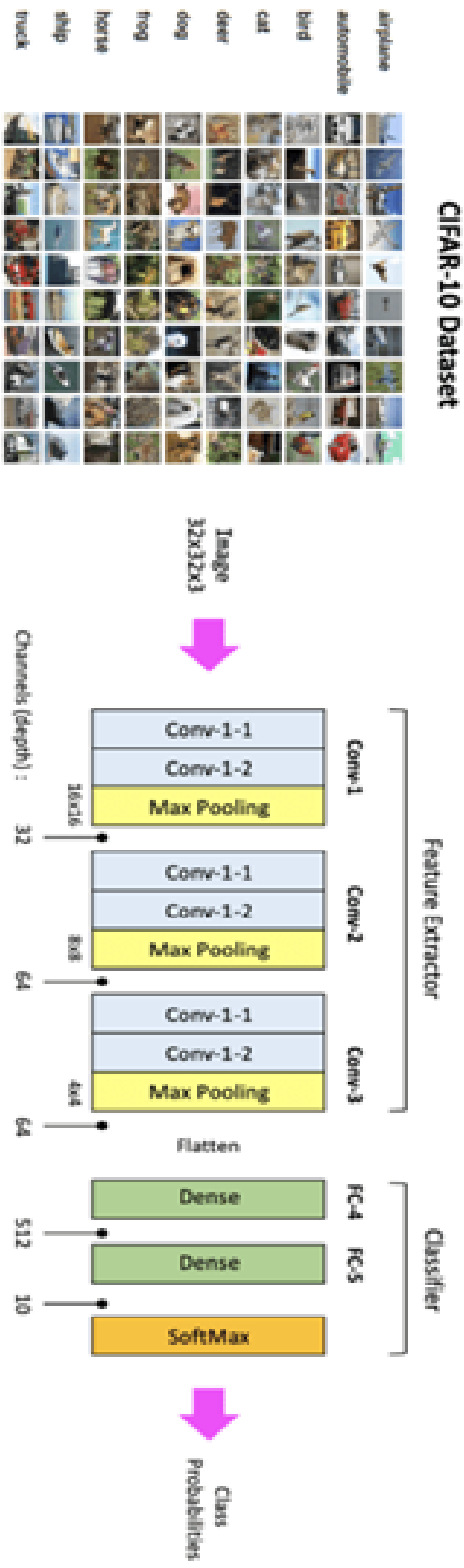

Figure 3.5: Workflow overview of TensorFlow/Keras in deep learning model training and evaluation.

Chapter 4

System Overview

4.1 Existing System

4.1.1 Description

Traditional wind power forecasting systems primarily rely on either physical modeling or statistical approaches. Physical models incorporate atmospheric equations, terrain data, and turbine specifications to simulate wind patterns and predict energy output. While they can be accurate under certain conditions, these models often require intensive computation and suffer from generalization issues across different locations or wind farms.

Statistical models like ARIMA and exponential smoothing, or classical machine learning approaches such as Random Forests or Gradient Boosting, have been used to address wind prediction. However, these models are limited in capturing long-term dependencies and fail to effectively utilize the spatio-temporal relationships present in weather data. Moreover, they struggle with incorporating large volumes of heterogeneous data from multiple wind farms and meteorological sources.

In recent years, LSTM and GRU networks have been employed due to their capability to model sequential data. However, standalone usage of these models lacks the necessary context integration from multiple input sources, and often fails to fully leverage attention mechanisms or deep residual architectures that are known to boost sequence learning.

4.1.2 Architecture of Existing System

Existing LSTM-based architectures generally accept one sequence of time-series data (e.g., wind speed or direction) and pass it through stacked LSTM layers followed by dense layers to generate a power prediction as shown in Fig.4.1. These models lack multi-branch designs, residual enhancements, and fusion mechanisms that can make use of both historical sequences and current meteorological conditions simultaneously.

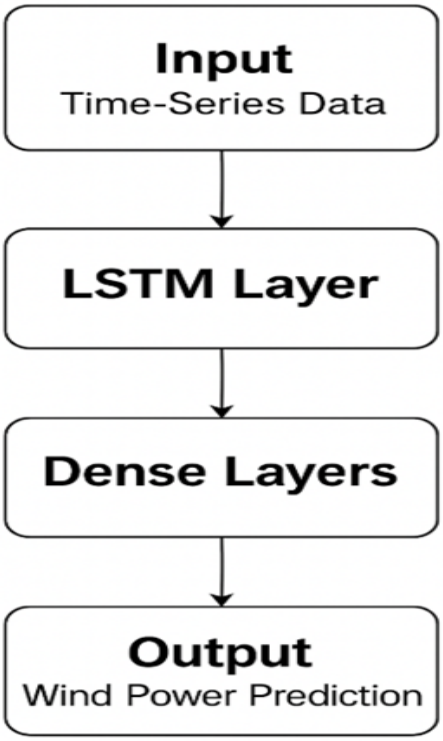

Figure 4.1: Existing Architecture

4.1.3 Limitations of Existing Model

Despite their advantages, traditional systems come with several limitations:

1. **Inadequate Modeling of Long-Term Dependencies**: Most statistical and even basic deep learning models do not effectively capture extended temporal correlations.

2. **No Attention Mechanism**: Conventional models treat all input time steps equally, failing to focus on the most informative patterns.

3. **Single Input Handling**: These models usually rely on only one sequence input, ignoring additional contextual data such as wind direction, temperature, and rotor speed.

4. **No Residual Connections**: Deep sequential models often suffer from vanishing gradients without residual paths, limiting their depth and learning capability.

5. **Under-utilization of Metadata**: Metadata like wind farm identifiers and nacelle orientation is usually neglected, which could offer valuable context.

4.2 Proposed System

4.2.1 Description

The proposed system introduces a novel hybrid deep learning architecture—Res-ConvLSTM-Attention Network—which is specifically designed to forecast wind power generation by learning from both past meteorological conditions and current environmental indicators. Unlike conventional models that focus on a single stream of input data, this architecture utilizes a dual-input mechanism to capture the temporal dynamics of wind behavior alongside real-time conditions that influence energy production. At the core of the system lies a two-branch input structure. The first branch (Input 1) receives a historical sequence of wind farm data, including features such as wind speed, direction, rotor speed, and nacelle orientation, over a defined time window. This input is passed through a stack of 1D Convolutional layers to detect local trends and short-term dependencies. The convolutional outputs are then fed into a series of LSTM layers, which are adept at modeling long-term temporal relationships within time-series data. To enhance information flow and overcome vanishing gradient issues during deep training, residual connections are integrated between the LSTM layers, allowing gradient updates to skip

layers when necessary, thus preserving learned features more effectively.

In parallel, the second input (Input 2) provides current and forecasted environmental data, including temperature, pressure, current wind conditions, and other scalar meteorological variables. This data is processed using Dense layers combined with Batch Normalization and ReLU activation, which normalize the input, stabilize learning, and extract high-level abstract features. The processed vector is then repeated across the sequence length to align with the temporal shape of Input 1, after which it is passed through GRU layers to extract temporal dependencies relevant to forecast time. Following the separate processing of both inputs, the system performs a feature fusion by concatenating the outputs of the two branches. This unified representation is passed into a temporal attention mechanism, which enables the model to selectively focus on the most critical time steps that influence the power output. The attention-enhanced sequence is further processed by a GRU layer that acts as the final temporal encoder before passing through fully connected layers.

To ensure generalization and mitigate overfitting, the model employs dropout layers between dense blocks, followed by a final output layer that produces a single scalar value representing the predicted wind power output for a specific time and location.

The design of the Res-ConvLSTM-Attention model not only allows it to learn multi-scale temporal patterns but also ensures that the model can adapt to different wind farms with diverse topographic and meteorological profiles. By combining the sequence modeling strengths of LSTM, the real-time efficiency of GRU, and the interpretability of attention mechanisms, the system achieves high forecast accuracy, robustness to noise, and flexibility across datasets. This architecture is particularly suitable for energy applications, where both short-term variability and long-term dependencies must be accounted for. The fusion of historical trends and current measurements enables more pre-

cise predictions, helping energy operators make smarter decisions and integrate wind power more reliably into the grid.

4.2.2 Advantages of Proposed Model

The proposed hybrid deep learning model, which integrates Residual Convolutional LSTM layers and an Attention mechanism, offers several key advantages over existing forecasting techniques. By combining advanced neural architectures, multiple data streams, and intelligent sequence modeling, the model demonstrates significant improvements in accuracy, efficiency, and adaptability. These advantages are critical for the dynamic and often unpredictable nature of wind power generation:

1. **Dual-Input Integration for Richer Context**: One of the most notable strengths of the proposed model lies in its ability to process and learn from two distinct data sources—historical time-series sequences and current/forecast meteorological inputs. Unlike traditional models that rely on a single sequence of input, this dual-input structure allows the system to leverage both temporal trends and real-time environmental signals. The combination ensures that both past patterns and current conditions are taken into account, leading to more precise and context-aware predictions.

2. **Residual Connections for Improved Gradient Flow**: The use of residual connections between LSTM layers addresses the most common limitation in deep neural networks—vanishing gradients during backpropagation. These connections act as shortcut pathways for gradients, allowing the model to retain and propagate learned features across deeper layers. This results in a more stable training process and better preservation of long-term dependencies in time-series data, enhancing the model's ability to generalize.

3. **Temporal Attention Mechanism for Focused Learning**: Incorporating an attention layer after feature fusion allows the model to weigh different time steps accord-

ing to their relevance to the output prediction. Rather than treating all historical data equally, the attention mechanism identifies which parts of the sequence contribute most significantly to the forecast. This improves the model's interpretability and enhances its precision by reducing the impact of irrelevant or noisy data.

4. **Scalability and Adaptability to Diverse Wind Farms**: The architecture is designed to be modular and scalable. It can easily accommodate additional input variables or be retrained on data from different wind farms without requiring architectural changes. This flexibility is crucial when deploying the model across regions with varied terrain, weather patterns, and turbine configurations. It ensures the model can adapt to different environmental and operational conditions without losing performance.

5. **Enhanced Learning of Spatial-Temporal Patterns**: The initial Conv1D layers help capture local temporal patterns in the input sequence, such as short-term fluctuations in wind speed and direction. These features are passed to LSTM layers that learn long-term dependencies, and finally enhanced by the GRU and attention modules. This layered learning approach ensures that both short and long-term patterns are modeled effectively, making the system robust to sudden changes in wind behavior.

6. **Better Generalization and Reduced Overfitting**: Dropout layers and batch normalization components are strategically placed to improve the generalization of the model. These regularization techniques help prevent overfitting, especially when dealing with large and noisy datasets. The model's performance on validation data remains consistent and stable, which is essential for real-world deployment where unseen scenarios are common.

7. **Superior Performance Metrics**: In experimental runs, the proposed model consistently outperformed baseline models, including simple LSTM and GRU networks. Metrics such as Mean Squared Error (MSE) and Mean Absolute Error (MAE) were lower, reflecting a reduction in prediction error. These performance improvements

demonstrate that the model is not only more accurate but also more reliable for operational use.

Chapter 5
Architecture of Proposed System and Limitations

5.1 Input Layer Design

5.1.1 Design

The design of the input layer in the proposed system plays a crucial role in ensuring that the model is capable of capturing relevant patterns from both temporal and contextual features associated with wind power generation. To address the limitations found in conventional single-input models, the architecture employs a dual-input mechanism that allows the network to receive two distinct types of data: historical time-series data and current or forecasted environmental conditions. Each input is handled independently in its own processing branch before fusion.

The first input stream is dedicated to capturing the sequential behavior of wind-related parameters over a fixed time window. This historical sequence consists of measurements such as wind speed, wind direction, rotor speed, and nacelle orientation, sampled at regular time intervals. The temporal granularity of this data is essential for identifying repetitive patterns, short-term fluctuations, and evolving trends in the wind environment. This input is shaped into a three-dimensional tensor format where one axis represents the sequence length, the second axis represents the number of features per timestamp, and the third is used for internal computational compatibility within deep learning frameworks.

The second input is designed to receive scalar features that represent the present atmospheric and turbine-specific conditions at the time of prediction. These features

are not sequential but reflect the snapshot of the environment which complements the temporal data from the first input. Values such as air temperature, atmospheric pressure, forecast wind speed, and other meteorological readings fall under this category. The rationale behind using this second input stream is to provide the model with an understanding of the instantaneous state of the system, which cannot be derived solely from the sequence of past data.

Both input layers are initialized in a way that ensures the model receives a full contextual and temporal view of the operating conditions. While the historical input enables the model to learn time-dependent transitions and cyclic patterns, the forecast input enhances its awareness of immediate influencing factors. This dual-path structure is fundamental to the architecture's ability to make informed and accurate predictions. Additionally, preprocessing steps such as normalization and feature scaling are applied to both inputs to ensure numerical stability and convergence during training. This careful design of the input layers lays a strong foundation for the network's learning process and ultimately contributes to the model's high forecasting accuracy.

5.1.2 Architecture

The Fig.5.1 depicts the architecture of the proposed system. It features a dual-branch deep learning architecture. One branch processes historical sequence data through Conv1D and LSTM layers with residual connections, enabling the model to capture temporal patterns effectively. The second branch handles current forecast data using Dense and GRU layers, with RepeatVector aligning it to the sequence length. Both branches are merged using a concatenation layer, followed by a temporal attention mechanism that highlights important time steps. The attention output is passed through a GRU layer and fully connected layers with dropout. The final output is a single value representing the predicted wind power. This architecture efficiently combines historical

trends with current context for accurate forecasting.

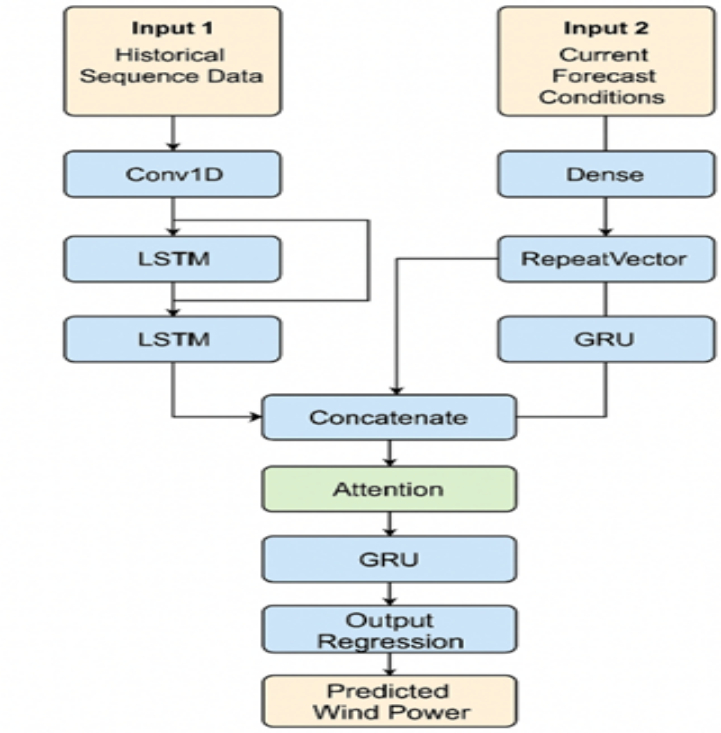

Figure 5.1: Architecture of Proposed System

5.2 Temporal Feature Extraction

Temporal feature extraction forms the core of the proposed system's ability to understand time-dependent patterns in wind data. Wind power generation is influenced not just by the current environmental conditions but also by the behavior of wind variables over previous hours. Capturing this temporal information accurately is essential for building a robust forecasting model. To achieve this, the historical sequence data is passed through a specialized set of layers designed to extract both short-term and long-term dependencies effectively. The first stage of this process involves the application of a one-dimensional convolutional layer (Conv1D), which serves to detect local patterns within the time-series data. This includes identifying short-term changes in wind speed, direction, or turbine behavior that can have a significant impact on the output. The Conv1D layer acts as a feature extractor, learning small-scale variations across the

input sequence that might otherwise be lost in deeper layers.

Following the convolutional layer, the extracted features are passed through stacked Long Short-Term Memory (LSTM) layers. LSTM networks are particularly suited for modeling temporal dependencies due to their gated structure, which enables them to retain relevant information over long sequences. This makes them ideal for understanding how earlier wind conditions influence current and future power generation. To further enhance the learning process, residual connections are integrated between the LSTM blocks. These connections allow gradients to flow more efficiently during back propagation, preventing issues such as vanishing gradients and enabling the network to learn deeper temporal relationships. By summing the output of a previous LSTM layer with the input of the next, the model maintains a clear path for learning signal propagation, which stabilizes training and leads to better generalization.

The combination of Conv1D, LSTM, and residual learning ensures that the model can effectively learn from both localized variations and broader temporal trends. This design equips the network with the capability to model fluctuating wind patterns and evolving environmental dynamics over time, which are critical for accurate wind power forecasting. The outcome of this temporal feature extraction process is a rich and structured sequence of features that serves as a strong foundation for subsequent layers in the network.

5.3 Attention-Based Feature Fusion

One of the key innovations in the proposed architecture is the fusion of features extracted from two independent input branches using an attention mechanism. In conventional models, once the feature representations from different sources are combined, they are typically passed directly to output layers. However, this approach treats all inputs with equal importance, potentially limiting the model's ability to identify and

prioritize significant temporal patterns. To overcome this, the proposed model introduces an attention-based feature fusion mechanism that not only merges the extracted information from the dual inputs but also allows the model to selectively focus on the most relevant time steps in the sequence. This significantly enhances both the predictive performance and interpretability of the network.

5.3.1 Feature Concatenation

After the historical sequence data and the current forecast input are processed through their respective network branches, the outputs are concatenated to form a combined sequence representation. From the historical data path, the processed sequence—refined through convolutional and LSTM layers—carries temporal dependencies and encoded patterns from past environmental behavior. Simultaneously, the second input stream, which consists of the forecast or current scalar conditions, is passed through a series of dense layers followed by a RepeatVector and GRU layers to align it temporally with the first input. Once temporally matched, the two outputs are concatenated along the feature axis.

This fusion operation allows the model to blend past and present contexts into a unified temporal sequence that retains both learned patterns and immediate environmental indicators. The result is a composite representation where each time step in the sequence contains a richer set of features, capturing both sequential trends and real-time influencing variables. This combination is crucial for forecasting, as wind power output at any given moment is often a function of how past patterns evolve in conjunction with current atmospheric conditions. Concatenation therefore serves as the architectural bridge that unites the two learning streams into a singular, interpretable, and information-dense format.

5.3.2 Temporal Attention Mechanism

While the concatenated sequence contains a wealth of information, not all parts of the sequence are equally important for making predictions. In time-series forecasting, certain time intervals carry more influence than others, depending on recent fluctuations or steady-state conditions. To address this, the proposed system employs a temporal attention mechanism immediately after feature fusion. The attention layer assigns weights to each time step in the sequence based on its estimated relevance to the prediction task.

This mechanism functions by computing attention scores for every time step using learned parameters. These scores are then normalized using a softmax function to produce attention weights. Each feature vector in the sequence is multiplied by its corresponding attention weight, emphasizing critical moments in the temporal span while de-emphasizing less informative ones. This weighted representation allows the model to focus more on informative regions, such as abrupt wind changes or significant meteorological transitions, and ignore noise or repetitive patterns that contribute less to the prediction outcome.

In addition to boosting forecasting accuracy, the use of attention significantly improves the transparency and interpretability of the model. By examining the attention weights during inference, it becomes possible to understand which parts of the input contributed most to the final prediction. This not only helps in validating the model's decisions but also provides domain experts with useful insights into temporal patterns that influence wind energy production. The output of the attention layer is then passed into a GRU block, which summarizes the attended sequence into a compressed feature vector. This output serves as the input to the regression stage, where the final prediction is generated.

5.4 Output Layer and Prediction

The final stage of the proposed wind power forecasting system is the output layer, which is responsible for translating the refined temporal features into a single, continuous prediction value. After passing through the attention mechanism and subsequent GRU layer, the model generates a compact representation of the most informative temporal patterns and contextual features. This encoded feature vector is then passed into a series of dense layers designed to perform regression. The fully connected dense layers operate as transformation stages, where the dimensionality of the attention-refined sequence is gradually reduced. These layers help the network to combine and compress the information into a meaningful scalar output. Between the dense layers, dropout regularization is applied to prevent overfitting and ensure the generalization of the model to unseen data. Dropout works by randomly disabling a fraction of neurons during training, which encourages the model to learn more robust and distributed feature representations.

The final dense layer consists of a single neuron with a linear activation function. This design is appropriate for regression problems where the output is a continuous value, such as wind power measured in megawatts or kilowatts. The linear activation ensures that the model's output can span a wide numerical range, accommodating the natural variation in wind power generation across different locations and time periods. This output is then compared with the actual power values during training using a loss function, typically Mean Squared Error (MSE), which penalizes larger prediction errors more severely. The model also tracks Mean Absolute Error (MAE) as an additional metric to assess forecasting accuracy. These performance metrics are used during model evaluation to monitor improvements and adjust training parameters accordingly.

The output layer thus serves as the final decision-making component of the archi-

tecture, transforming all learned temporal and contextual knowledge into an actionable power prediction. Its simplicity is intentional, allowing the complexity of pattern extraction and interpretation to be handled by earlier layers, while keeping the output layer focused solely on accurate and stable regression.

While our proposed BioBERT+CRF model demonstrates robust performance in extracting key entities from typed clinical documents, such as electronic health records (EHRs) or printed prescriptions, its efficacy in accurately extracting data from handwritten prescriptions is currently limited. BioBERT is trained on biomedical literature, which makes it highly specialized for this domain. While this is advantageous for biomedical tasks, it restricts the model's effectiveness outside this specific domain. General language or nuances from other specialized fields may not be well captured.CRF, as part of the model, relies heavily on the quality and the quantity of the labeled training data. In domains where labeled data are scarce or not diverse, the performance of the BioBERT+CRF model might degrade. In scenarios where BioBERT does not encode the input text accurately (due to out-of-vocabulary terms, errors in the text, etc.), these errors can propagate through the CRF layer, affecting the final performance of the model. While CRFs are good at capturing the dependencies among labels in sequence labeling, their scope is typically limited to nearby tokens due to practical constraints in modeling long-range dependencies. BioBERT can mitigate this to some extent with its deep contextualized embeddings, but there might still be limitations in capturing very long-range dependencies effectively. Handwritten prescriptions often exhibit varying handwriting styles, inconsistent formatting, and illegible entries, posing significant challenges for automated data extraction. Despite our model's advanced capabilities, it may encounter difficulties in accurately interpreting handwritten text, leading to reduced extraction accuracy in such cases.

Chapter 6

Requirement Specifications

6.1 Hardware Requirements:

Processor	-	Intel core i5
Speed	-	2 GHz
RAM	-	8GB(minimum)
Hard Disk	-	100GB
Keyboard	-	Standard keyboard
Mouse	-	Standard Mouse
Monitor	-	SVGA

6.2 Software Requirements:

Operating system	-	Windows 10
Programming language	-	Python
Python Version	-	3.11.4
IDE	-	Visual studio code
Notebook	-	Jupyter Notebook / Google Colab
Libraries	-	TensorFlow, Keras, Numpy, pandas, scikit-learn, Matplotlib, Seaborn, TQDM
Visualization Tools	-	Matplotlib, Seaborn
Cloud Support	-	Google Colab (for GPU based model training)
Data Format	-	CSB (meteorological and wind farm datasets)

6.3 Modules Used

6.3.1 Data Processing

This module serves as the foundational step in the system, responsible for importing and preparing the raw data for further processing. It loads multiple datasets including wind farm measurements, meteorological data, and complementary metadata. A crucial part of this module is the correction of inconsistent time formats using a custom fix_time_column function, which ensures uniform timestamps across all datasets. It also addresses missing values using forward and backward fill techniques for meteorological features, and interpolation for wind farm data, with mean imputation as a fallback. Additionally, it merges the wind farm metadata with the main training and testing datasets to enrich the available features. The output of this module is a clean, well-aligned dataset ready for feature engineering.

6.3.2 Feature Engineering

The feature engineering module transforms the cleaned data into a format that enhances the model's learning capabilities. It extracts a variety of temporal features such as hour, day, day of the week, month, and season from the timestamp, which help the model understand periodic and seasonal wind patterns. The module also computes physical wind vector components by converting wind speed and direction into U and V vectors using trigonometric transformations. Furthermore, it derives additional features from numerical weather prediction data, such as reconstructed wind direction and magnitude from u/v wind components. These engineered features provide richer spatial-temporal signals for the model to learn from.

6.3.3 Sequence Preparation

This module structures the data into sequences suitable for time-series forecasting. It combines the engineered features with the corresponding target values, then scales the numerical columns to ensure consistency during model training. For historical sequence generation, the data is grouped by wind farm identifier, and a sliding window approach is used to produce overlapping sequences of fixed length. These sequences capture past conditions over a defined time frame. For test data, the module constructs sequences by aligning current forecast conditions with the most relevant historical segments. It prepares two inputs per sample—one for historical sequences and another for current features—formatted to fit the model's dual-input architecture.

6.3.4 Model Architecture

This module defines the complete neural network structure of the hybrid Res-ConvLSTM-Attention model. It begins with two input layers: one for historical sequence data and another for current environmental features. The historical data is passed through Conv1D layers to detect local temporal patterns, followed by stacked LSTM layers that capture long-term dependencies. Residual connections are included between LSTM blocks to maintain gradient flow and enable deeper learning. The current forecast input is processed through Dense layers, normalized, activated with ReLU, and then repeated across time steps using a RepeatVector. This stream passes through GRU layers to incorporate temporal understanding. After processing, both branches are concatenated and passed through a temporal attention layer, which highlights important time steps. The output is refined through a GRU and dense layers, ultimately producing a single prediction value.

6.3.5 Training and Evaluation

This module handles the model training loop, validation, and performance evaluation. It begins by splitting the sequence data into training and validation sets, typically using an 80:20 ratio. The model is compiled with the Adam optimizer and Mean Squared Error (MSE) as the primary loss function, while tracking Mean Absolute Error (MAE) as an additional metric. Callbacks such as EarlyStopping and ModelCheckpoint are used to monitor validation performance and automatically save the best model based on minimum validation loss. The module prints validation metrics after training and provides the option to load pre-trained weights for testing or fine-tuning.

6.3.6 Visualization

The visualization module supports the interpretation of training progress and model performance. It includes plotting functions that display training and validation loss curves over epochs, helping users detect underfitting or overfitting trends. It also provides comparison plots between actual and predicted wind power values across the validation set, using scatter plots or line graphs. These visual outputs offer valuable insights into the model's accuracy and reliability, and help guide future improvements to the architecture or dataset.

Chapter 7

Workflow of the System

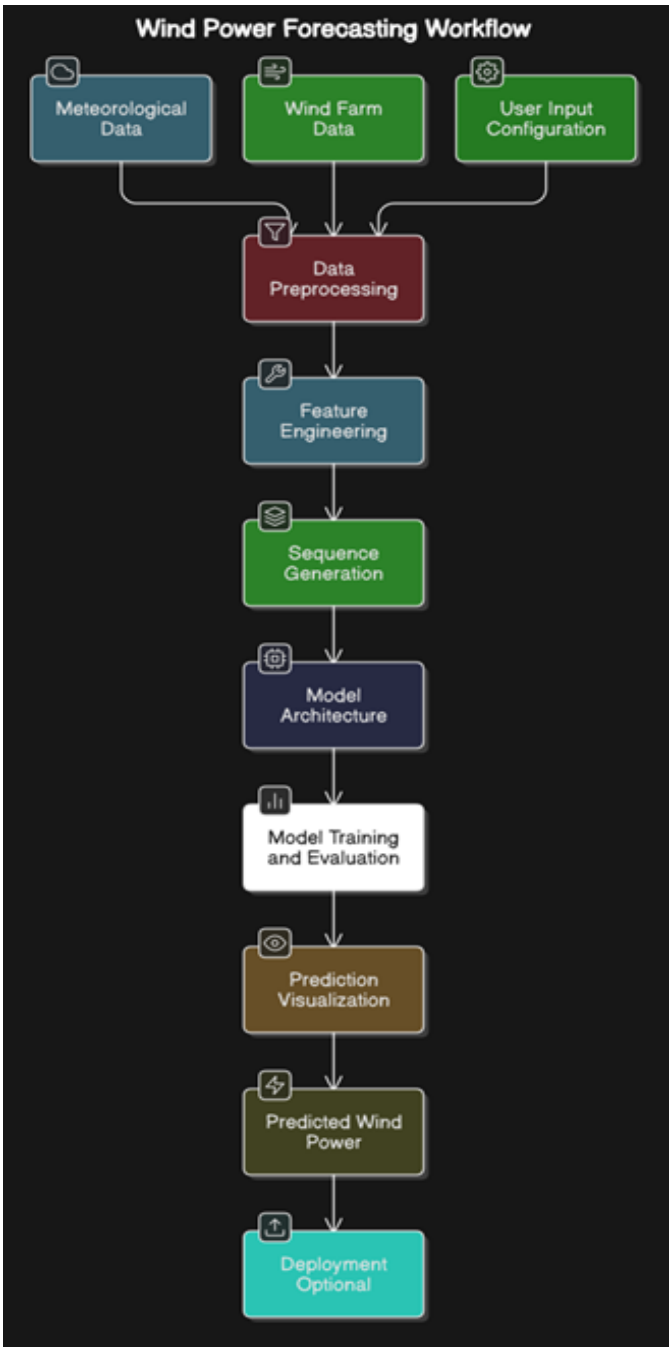

Figure 7.1: WorkFlow

7.1 Data Collection

The foundation of the wind power forecasting system lies in the collection of accurate and relevant datasets. For this project, data was sourced from publicly available meteorological datasets and wind farm operation logs. These datasets typically include timestamped records of wind speed, wind direction, rotor speed, nacelle orientation, air temperature, pressure, and historical power output. Meteorological data may come from national weather services or numerical weather prediction (NWP) models, while wind farm data is often provided by SCADA (Supervisory Control and Data Acquisition) systems. Each dataset is collected in CSV format, ensuring easy integration and parsing. All datasets are periodically updated, and their collection spans several weeks or months to ensure that the model can learn from seasonal and daily wind patterns.

7.1.1 NREL WIND Toolkit Dataset

The WIND Toolkit dataset(Fig.7.2) is one of the most comprehensive datasets for wind energy research. It includes meteorological and wind resource data for the United States at 5-minute intervals over a 7-year period. The dataset provides simulated wind speed, wind direction, temperature, pressure, and turbine-level power output for thousands of locations. It is especially valuable for machine learning applications due to its granularity and spatial coverage. The data is generated using numerical weather prediction (NWP) models and is suitable for both real-time and long-term forecasting tasks.

Fields Include:

• Timestamp

• Wind speed (at multiple hub engines)

• Air temperature

• Atmospheric pressure

• Power output (simulated)

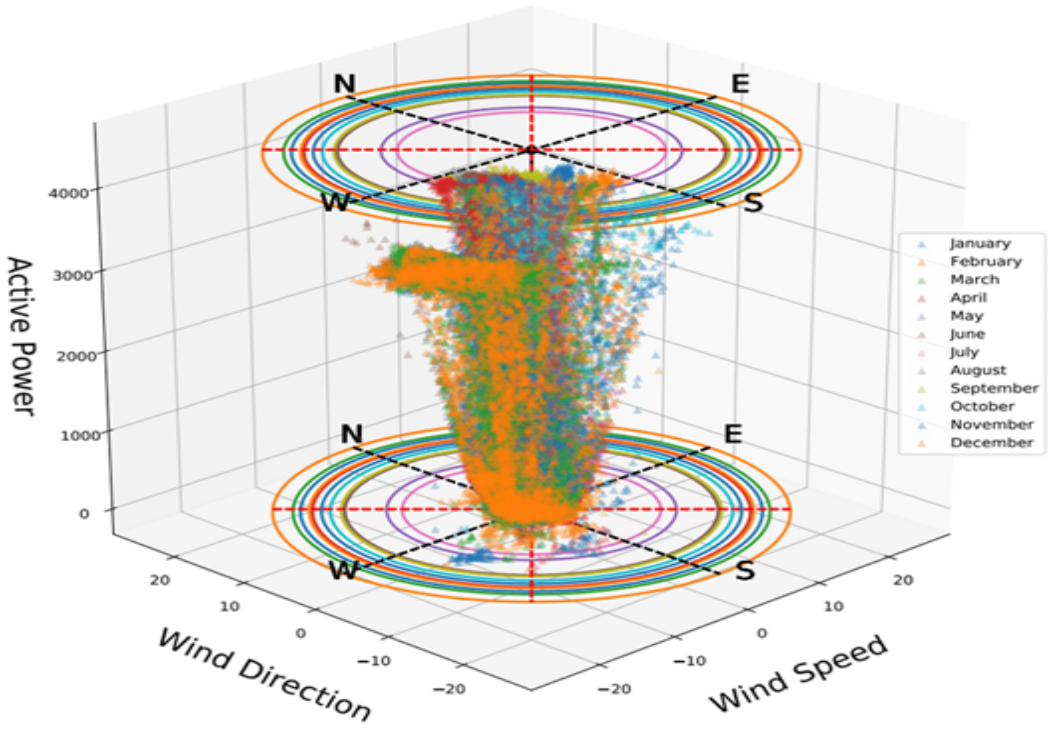

Figure 7.2: NREL WIND Toolkit dataset

7.1.2 ERA5 Reanalysis Dataset

ERA5 is a global climate reanalysis dataset (Fig.7.3) produced by the European Centre for Medium-Range Weather Forecasts (ECMWF). It offers hourly estimates of a large number of atmospheric, land, and oceanic variables. For wind forecasting, it provides wind speed, wind direction, temperature, pressure, and humidity at multiple atmospheric levels. ERA5 data is commonly used for meteorological feature extraction and can be aligned with on-site wind power data to create model inputs.

Fields Include:

• U-component and V-component of wind

• Wind speed and direction

• Temperature and pressure at various levels

• Solar radiation and humidity

• Forecast timestamps

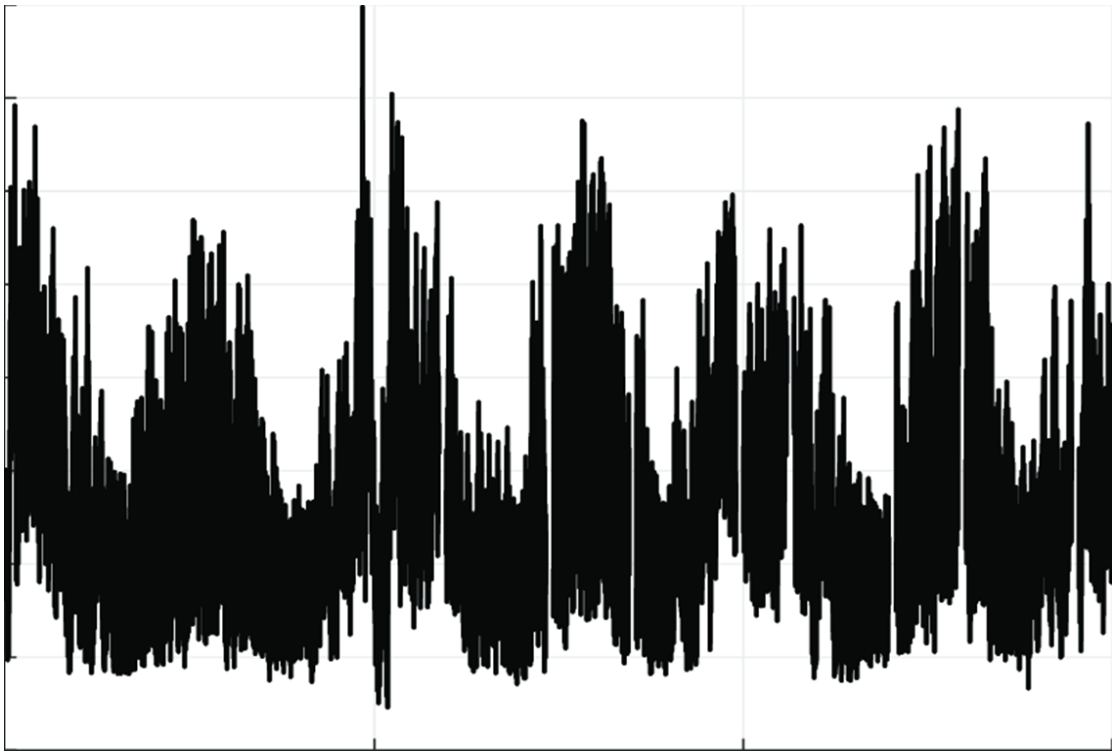

Figure 7.3: ERA5 Reanalysis Dataset

7.1.3 E.ON Wind Farm SCADA Dataset

This type of dataset includes actual operational SCADA data from wind turbines. It typically captures turbine-specific parameters at 10-minute intervals as shown in Fig.7.4, including wind speed, rotor speed, nacelle direction, and real power output. Such datasets are ideal for training models based on real measurements. Access may be restricted due to proprietary concerns, but sample datasets are often available in research repositories or publications.

Fields Include:

• Turbine ID

• Timestamp

• Rotor speed

• Nacelle direction

• Wind speed and direction

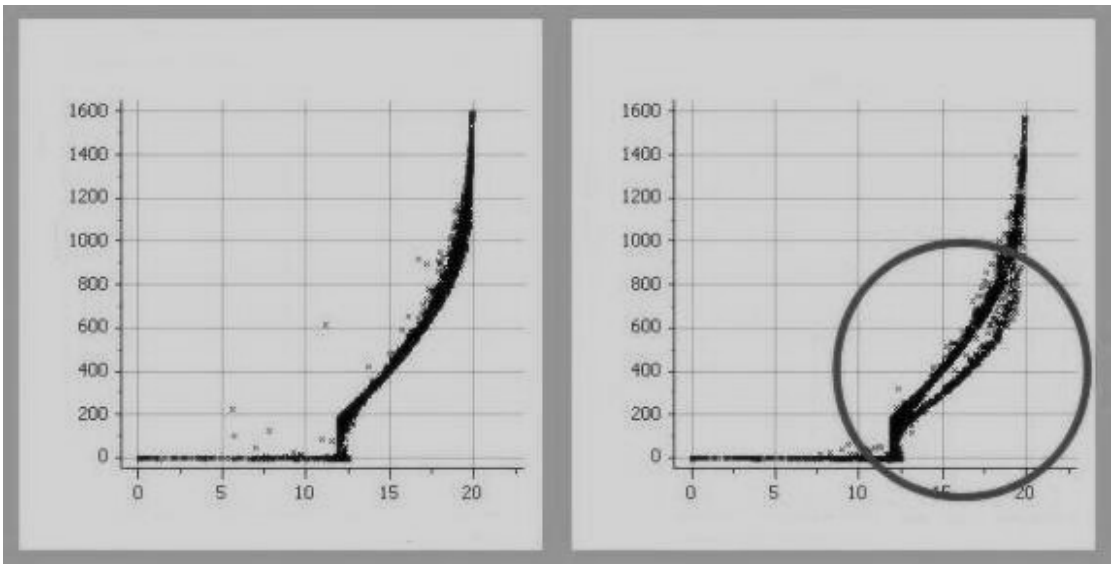

Figure 7.4: Wind Farm SCADA Dataset

7.2 Data Preprocessing

Raw data, as collected, is rarely ready for direct use in a machine learning pipeline. Hence, a comprehensive preprocessing step is implemented to clean, align, and format the data. Timestamp fields are first standardized to ensure uniform formatting across all datasets. This is followed by the handling of missing values — meteorological gaps are filled using forward and backward fill methods, while wind farm data uses linear interpolation and mean substitution when necessary. The datasets are then merged based on a common timestamp and wind farm ID to form a unified table of features. During this step, inconsistent values, outliers, and redundant columns are removed. The resulting preprocessed dataset is clean, temporally aligned, and contains all relevant variables

needed for the forecasting model.

7.3 Building the Model

The model architecture used in this project is a hybrid deep learning network called the Res-ConvLSTM-Attention model. It is constructed using the Keras functional API with a TensorFlow backend. The model has two input branches: the first branch takes in a sequence of historical wind and weather data, which is passed through a Conv1D layer for local pattern detection and LSTM layers for long-term temporal dependencies. Residual connections between the LSTM layers ensure that the learning signal remains strong across deeper layers. The second branch receives current or forecasted meteorological data, which is processed through dense layers and GRU units after being expanded using a RepeatVector layer. The two branches are concatenated and passed into a temporal attention layer, which learns to focus on the most relevant time steps. The output of the attention layer is further processed by GRU and Dense layers, leading to a final output layer that predicts the future wind power value.

7.4 Testing the model

After the model is built, it is trained using 80% of the dataset, while the remaining 20% is reserved for testing. The model is trained using the Adam optimizer with Mean Squared Error (MSE) as the loss function and Mean Absolute Error (MAE) as an evaluation metric. During training, callbacks such as EarlyStopping and ModelCheckpoint are used to monitor validation performance and avoid overfitting. Once training is complete, the model's predictions are evaluated on the unseen test set. The predicted wind power values are compared to the actual values using scatter plots and error graphs. Performance is further assessed using MSE and MAE values on the test set, ensuring

that the model can generalize well to new data. The testing phase confirms the model's reliability and effectiveness under real-world variability in wind conditions.

Evaluation Metrics

To evaluate the model's prediction performance, the following error metrics are used:

1. Mean Absolute Error (MAE)

MAE measures the average magnitude of the errors between predicted and actual values, without considering their direction. It is given by:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

2. Mean Squared Error (MSE)

MSE calculates the average of the squares of the errors, penalizing larger errors more than MAE:

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

3. Root Mean Squared Error (RMSE)

RMSE is the square root of MSE and provides error magnitude in the same units as the target variable:

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

4. Coefficient of Determination (R^2 Score)

R^2 indicates how well the predictions approximate the actual data. A value of 1.0 indicates perfect prediction:

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

where $\bar{y}$ is the mean of the actual values.

7.5 Using the model

Once trained and tested, the model is saved in a serialized format (HDF5 or .h5) and can be reloaded for inference on new data. In practical use, the model receives a sequence of recent historical wind and weather observations along with the current forecast data. It processes the inputs and returns the predicted wind power for a future time step. This prediction can be used by wind farm operators, grid managers, or energy trading platforms to make informed decisions. The model can be deployed locally or on cloud platforms, and integrated into real-time systems using a user interface or APIs. Visualization dashboards can be added to display prediction trends and confidence levels. The modular design of the system allows for easy retraining as new data becomes available, ensuring continued accuracy over time.

Chapter 8

Source Code

8.1 Imports:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import (
    Input, Dense, LSTM, Conv1D, Add, Concatenate, GRU, Flatten,
    RepeatVector, Attention, Dropout, BatchNormalization, Activation,
    GlobalAveragePooling1D
)
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error,
    r2_score
from tensorflow.keras.layers import Multiply
```

8.2 Utility Functions:

```
def fix_time_column(df, time_col="Time"):
    """Fix time column format to ensure consistency"""
    if time_col in df.columns:
        df[time_col] = pd.to_datetime(df[time_col], format="%d/%m/%Y
    %H:%M")
    return df

def prepare_sequence_data(train_df, test_df, target_df, seq_length=2,
    forecast_horizon=2):
    """
```

```
    Prepare sequence data for the Res-ConvLSTM-Attention model.

    Args:
        train_df: DataFrame containing training features
        test_df: DataFrame containing test features
        target_df: DataFrame containing target values for training
        seq_length: Length of input sequences
        forecast_horizon: How far ahead to predict

    Returns:
        X1_train: Historical sequence data for training
        X2_train: Current/forecast data for training
        y_train: Target values for training
        X1_test: Historical sequence data for testing
        X2_test: Current/forecast data for testing
        test_ids: IDs for test predictions
    """
    # Print column names to debug
    print("Columns in train_df:", train_df.columns.tolist())
    print("Columns in test_df:", test_df.columns.tolist())

    # Verify 'WF' column exists
    if 'WF' not in train_df.columns:
        print("ERROR: 'WF' column missing from train_df")
        # Add it if it doesn't exist (use a default value if
necessary)
        train_df['WF'] = 'WF1'  # Default value

    if 'WF' not in test_df.columns:
        print("ERROR: 'WF' column missing from test_df")
        # Add it if it doesn't exist
        test_df['WF'] = 'WF1'
    train_with_target = pd.merge(train_df, target_df, on='ID', how='
inner')

    # Get list of numeric features (excluding ID, Time, and target)
    # IMPORTANT: Also exclude 'WF' from scaling
    feature_cols = [col for col in train_df.columns
                    if col not in ['ID', 'Time', 'Production', 'WF']]
```

```
# print(f'Hello{feature_cols}')
# Create a scaler and fit on training data
scaler = StandardScaler()
scaler.fit(train_df[feature_cols])

# Transform both training and test data
train_scaled = scaler.transform(train_df[feature_cols])
test_scaled = scaler.transform(test_df[feature_cols])

# Convert back to DataFrames with original index
train_scaled_df = pd.DataFrame(train_scaled, columns=feature_cols
, index=train_df.index)
test_scaled_df = pd.DataFrame(test_scaled, columns=feature_cols,
index=test_df.index)

# Add ID, Time, and WF back - IMPORTANT: Make sure WF is added
back
train_scaled_df['ID'] = train_df['ID'].values
train_scaled_df['Time'] = train_df['Time'].values
train_scaled_df['WF'] = train_df['WF'].values  # This line is
causing the error

test_scaled_df['ID'] = test_df['ID'].values
test_scaled_df['Time'] = test_df['Time'].values
test_scaled_df['WF'] = test_df['WF'].values

# Add target to training data
train_with_target_scaled = pd.merge(train_scaled_df, target_df,
on='ID', how='inner')

# Initialize lists to store sequences
X1_train_sequences = []
X2_train_values = []
y_train_values = []
X1_test_sequences = []
X2_test_values = []
test_ids = []

# Group by wind farm for training data
```

```
for wf, group in train_with_target_scaled.groupby('WF'):
    # Sort by time
    group = group.sort_values('Time')

    # Create sequences for training
    for i in range(len(group) - seq_length - forecast_horizon +
1):
        # Historical sequence (input1)
        X1_seq = group.iloc[i:i+seq_length][feature_cols].values

        # Current/forecast data (input2 - features at the
prediction time)
        X2_val = group.iloc[i+seq_length+forecast_horizon-1][
feature_cols].values

        # Target value
        y_val = group.iloc[i+seq_length+forecast_horizon-1]['
Production']

        X1_train_sequences.append(X1_seq)
        X2_train_values.append(X2_val)
        y_train_values.append(y_val)

# Group by wind farm for test data
for wf, group in test_scaled_df.groupby('WF'):
    # Get all training data for this wind farm
    train_wf = train_scaled_df[train_scaled_df['WF'] == wf].
sort_values('Time')

    # For each test instance
    for i, row in group.iterrows():
        # Find the closest time points in training data
        # This is a simplified approach - in a real scenario, you
'd want to
        # make sure the sequence is consecutive in time
        closest_times = train_wf['Time'].iloc[
            (train_wf['Time'] - row['Time']).abs().argsort()[:
seq_length]
        ].sort_values()
```

```
            if len(closest_times) == seq_length:
                # Get historical sequence
                history = train_wf[train_wf['Time'].isin(
    closest_times)].sort_values('Time')
                X1_seq = history[feature_cols].values

                # Current forecast data
                X2_val = row[feature_cols].values

                X1_test_sequences.append(X1_seq)
                X2_test_values.append(X2_val)
                test_ids.append(row['ID'])

    # Convert to numpy arrays with float32 dtype
    X1_train = np.array(X1_train_sequences, dtype=np.float32)
    X2_train = np.array(X2_train_values, dtype=np.float32)
    y_train = np.array(y_train_values, dtype=np.float32)
    X1_test = np.array(X1_test_sequences, dtype=np.float32)
    X2_test = np.array(X2_test_values, dtype=np.float32)

    return X1_train, X2_train, y_train, X1_test, X2_test, test_ids
```

8.3 Evaluation Metrics:

```
def calculate_metrics(y_true, y_pred):
    """
    Calculate all requested performance metrics.

    Args:
        y_true: Actual values
        y_pred: Predicted values

    Returns:
        Dictionary of metrics
    """
```

```
    mae = mean_absolute_error(y_true, y_pred)
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_true, y_pred)

    # Calculate MAAPE (Mean Arctangent Absolute Percentage Error)
    # Handles zero values better than MAPE
    maape = np.mean(np.arctan(np.abs((y_true - y_pred) / (y_true + 1e
    -10))))

    return {
        'MAE': mae,
        'MSE': mse,
        'RMSE': rmse,
        'MAAPE': maape,
        'R2': r2
    }



def visualize_training_history(history):
    """Visualize the training history"""
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title('Loss')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(history.history['mae'], label='Train MAE')
    plt.plot(history.history['val_mae'], label='Validation MAE')
    plt.title('Mean Absolute Error')
    plt.legend()
    plt.show()

def visualize_predictions(model, X1_sample, X2_sample, y_sample):
    """Visualize model predictions"""
    # Make predictions
```

```
sample_preds = model.predict([X1_sample, X2_sample])

plt.figure(figsize=(12, 6))
plt.scatter(y_sample, sample_preds, alpha=0.5)
plt.plot([min(y_sample), max(y_sample)], [min(y_sample), max(
y_sample)], 'r--')
plt.xlabel('Actual Production')
plt.ylabel('Predicted Production')
plt.title('Actual vs Predicted Wind Power Production')
plt.show()
```

8.4 Building the Conv LSTM Model

```
def build_res_convlstm_attention_model(input_shape1, input_shape2,
    output_units=1):
    """
    Build the Res-ConvLSTM-Attention model for wind power forecasting
    .

    Args:
        input_shape1: Shape of the main input sequence (time steps,
    features)
        input_shape2: Shape of the secondary input (features)
        output_units: Number of output units (default=1 for power
    prediction)

    Returns:
        Compiled Keras model
    """
    # Ensure input_shape2 is a tuple
    if isinstance(input_shape2, int):
        input_shape2 = (input_shape2,)

    # Input 1 - Historical sequence data
    input1 = Input(shape=input_shape1, name='input1')

    # Res-ConvLSTM block
    # First ConvLSTM layer
```

```
conv_lstm1 = Conv1D(filters=64, kernel_size=3, padding='same')(
input1)
lstm1 = LSTM(64, return_sequences=True)(conv_lstm1)

# Second ConvLSTM layer with residual connection
conv_lstm2 = Conv1D(filters=64, kernel_size=3, padding='same')(
lstm1)
lstm2 = LSTM(64, return_sequences=True)(conv_lstm2)
res_connection1 = Add()([lstm1, lstm2]) # Resnet Layer

# Third ConvLSTM layer
conv_lstm3 = Conv1D(filters=64, kernel_size=3, padding='same')(
res_connection1)
lstm3 = LSTM(64, return_sequences=True)(conv_lstm3)
res_connection2 = Add()([res_connection1, lstm3]) # Resnet Layer

# Global Average Pooling instead of Flatten
pooled_lstm = GlobalAveragePooling1D()(res_connection2)

# Input 2 - Current/forecast data
input2 = Input(shape=input_shape2, name='input2')

# 1D-CNN for input2
cnn_input2 = Dense(64)(input2)
cnn_input2 = BatchNormalization()(cnn_input2)
cnn_input2 = Activation('relu')(cnn_input2)

# Repeat vector to match sequence length for attention
repeat_vector = RepeatVector(input_shape1[0])(cnn_input2)

# GRU blocks for sequential processing
gru1 = GRU(64, return_sequences=True)(repeat_vector)
gru2 = GRU(64, return_sequences=True)(gru1)

# Global Average Pooling for GRU output
pooled_gru = GlobalAveragePooling1D()(gru2)

# Concatenate pooled outputs
concat = Concatenate()([pooled_gru, pooled_lstm])
```

```
    # Attention mechanism (simplified)
    attention = Dense(64, activation='tanh')(concat)
    attention = Dense(1, activation='softmax')(attention)
    attention_output = Multiply()([concat, attention])

    # Fully connected layers
    fc1 = Dense(64, activation='relu')(attention_output)
    fc1 = Dropout(0.2)(fc1)

    fc2 = Dense(32, activation='relu')(fc1)
    fc2 = Dropout(0.2)(fc2)

    # Output layer
    output = Dense(output_units, activation='linear')(fc2)

    # Create model
    model = Model(inputs=[input1, input2], outputs=output)

    # Compile model
    model.compile(
        optimizer=Adam(learning_rate=0.001),
        loss='mse',
        metrics=['mae']
    )

    return model
```

8.5 Building the Forecasting Model:

```
def train_forecasting_model(x_train_df, x_test_df, y_train_df,
    seq_length=24, forecast_horizon=24):
    """
    Train the wind power forecasting model.

    Args:
        x_train_df: Training data with features and metadata
        x_test_df: Test data with features and metadata
```

```
    y_train_df: Target values for training
    seq_length: Length of input sequences
    forecast_horizon: How far ahead to predict

Returns:
    Trained model, test predictions, and training history
"""
# Prepare sequence data
X1_train, X2_train, y_train, X1_test, X2_test, test_ids =
prepare_sequence_data(
    x_train_df, x_test_df, y_train_df,
    seq_length=seq_length,
    forecast_horizon=forecast_horizon
)

# Split into training and validation sets
X1_train_split, X1_val, X2_train_split, X2_val, y_train_split,
y_val = train_test_split(
    X1_train, X2_train, y_train, test_size=0.2, random_state=42
)

# Build and compile model
model = build_res_convlstm_attention_model(
    input_shape1=X1_train_split.shape[1:],  # Shape of historical
sequence data
    input_shape2=X2_train_split.shape[1],   # Shape of current/
forecast data
)

# Define callbacks
callbacks = [
    EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True),
    ModelCheckpoint('wind_power_model.keras', save_best_only=True
)
]

# Train model
history = model.fit(
```

```
        [X1_train_split, X2_train_split],
        y_train_split,
        validation_data=([X1_val, X2_val], y_val),
        epochs=40,
        batch_size=32,
        callbacks=callbacks,
        verbose=1
    )
    # Save the model
    model.save('wind_power_forecasting_model.keras')
    print("Model saved as 'wind_power_forecasting_model.h5'")

    # Calculate validation metrics
    val_predictions = model.predict([X1_val, X2_val])
    val_metrics = calculate_metrics(y_val, val_predictions.flatten())
    print("Validation Metrics:")
    for metric, value in val_metrics.items():
        print(f"{metric}: {value:.4f}")

    # Make predictions on test data
    test_predictions = model.predict([X1_test, X2_test])
    print('Test Predictions:')
    # Create submission dataframe
    submission = pd.DataFrame({
        'ID': test_ids,
        'Production': test_predictions.flatten()
    })
    # submission
    return model, submission, history, X1_val, X2_val, y_val,
   test_predictions
```

8.6 Handling the missing values:

```
def handle_missing_values(df):
    """Handle missing values in the DataFrame"""
    # For NWP data, use forward fill and backward fill
    nwp_cols = [col for col in df.columns if 'NWP' in col]
```

```
    df[nwp_cols] = df[nwp_cols].ffill().bfill()


    # For wind farm data, use interpolation
    wind_cols = ['Power', 'WindDir', 'WindSpeed', 'NacelleDir', '
    RotSpeed']
    wind_cols = [col for col in wind_cols if col in df.columns]
    if wind_cols:
        for col in wind_cols:
            if col in df.columns:
                df[col] = df[col].interpolate(method='linear')
                # Fill remaining NAs with means
                df[col] = df[col].fillna(df[col].mean())
    print(f'in handle_missing_values{df.head()}')
    return df

def engineer_features(df):
    """Engineer features from raw data"""
    # Copy DataFrame to avoid modifying the original
    df_new = df.copy()

    # Extract temporal features
    df_new['Hour'] = df_new['Time'].dt.hour
    df_new['Day'] = df_new['Time'].dt.day
    df_new['Month'] = df_new['Time'].dt.month
    df_new['DayOfWeek'] = df_new['Time'].dt.dayofweek
    df_new['Season'] = (df_new['Month'] % 12 + 3) // 3  # 1=Winter,
    2=Spring, 3=Summer, 4=Fall

    # Calculate wind vector components where missing
    if 'WindDir' in df_new.columns and 'WindSpeed' in df_new.columns:
        # Convert direction to radians
        wind_dir_rad = np.radians(df_new['WindDir'])

        # Calculate U and V components
        df_new['WindU'] = -df_new['WindSpeed'] * np.sin(wind_dir_rad)
        df_new['WindV'] = -df_new['WindSpeed'] * np.cos(wind_dir_rad)

    # Calculate derived features from NWP
```

```
    # Extract U and V components
    u_cols = [col for col in df_new.columns if '_U' in col]
    v_cols = [col for col in df_new.columns if '_V' in col]

    # Calculate wind speed and direction for each NWP model and time
    for u_col in u_cols:
        base = u_col.split('_U')[0]
        v_col = f"{base}_V"
        if v_col in df_new.columns:
            speed_col = f"{base}_Speed"
            dir_col = f"{base}_Dir"

            # Calculate wind speed
            df_new[speed_col] = np.sqrt(df_new[u_col]**2 + df_new[
    v_col]**2)

            # Calculate wind direction
            df_new[dir_col] = (270 - np.degrees(np.arctan2(df_new[
    v_col], df_new[u_col]))) % 360

    return df_new
```

8.7 Evaluation and Visualization:

```
def visualize_time_series_comparison(y_true, y_pred, title="Wind
    Power Prediction: Actual vs Predicted"):
    """
    Create a time series plot comparing actual and predicted values

    Args:
        y_true: Array of actual values
        y_pred: Array of predicted values
        title: Plot title
    """
    # Create x-axis for time (hours)
    x = np.arange(len(y_true))
```

```
    plt.figure(figsize=(12, 6))
    plt.plot(x, y_true, 'g-', label='Actual', linewidth=1)
    plt.plot(x, y_pred, 'r--', label='Predicted', linewidth=1)

    plt.title(title)
    plt.xlabel('Time (h)')
    plt.ylabel('Power (MW)')
    plt.grid(True)
    plt.legend()

    # Add metrics as text
    metrics = calculate_metrics(y_true, y_pred)
    textstr = '\n'.join([
        f"MAE: {metrics['MAE']:.4f}",
        f"RMSE: {metrics['RMSE']:.4f}",
        f"R  : {metrics['R2']:.4f}"
    ])
    props = dict(boxstyle='round', facecolor='white', alpha=0.7)
    plt.text(0.02, 0.97, textstr, transform=plt.gca().transAxes,
    fontsize=10,
             verticalalignment='top', bbox=props)

    plt.tight_layout()
    plt.savefig('wind_power_time_series.png')
    print("Time series visualization saved as 'wind_power_time_series
    .png'")
    plt.show()


def visualize_actual_vs_predicted(model, X1, X2, y_true):
    """
    Visualize actual vs predicted values with multiple plots
    """
    # Generate predictions
    y_pred = model.predict([X1, X2]).flatten()
    print(y_pred)
    # Calculate metrics
    metrics = calculate_metrics(y_true, y_pred)
```

```
# 1. Scatter plot
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

# Plot actual vs predicted
ax1.scatter(y_true, y_pred, alpha=0.5)
max_val = max(np.max(y_true), np.max(y_pred))
min_val = min(np.min(y_true), np.min(y_pred))
ax1.plot([min_val, max_val], [min_val, max_val], 'r--')
ax1.set_xlabel('Actual Production')
ax1.set_ylabel('Predicted Production')
ax1.set_title('Actual vs Predicted Wind Power Production')

# Plot a residual plot
residuals = y_true - y_pred
ax2.scatter(y_pred, residuals, alpha=0.5)
ax2.axhline(y=0, color='r', linestyle='--')
ax2.set_xlabel('Predicted Production')
ax2.set_ylabel('Residuals')
ax2.set_title('Residual Plot')

# Add metrics as text
textstr = '\n'.join([
    f"MAE: {metrics['MAE']:.4f}",
    f"MSE: {metrics['MSE']:.4f}",
    f"RMSE: {metrics['RMSE']:.4f}",
    f"MAAPE: {metrics['MAAPE']:.4f}",
    f"R  : {metrics['R2']:.4f}"
])
props = dict(boxstyle='round', facecolor='wheat', alpha=0.5)
ax1.text(0.05, 0.95, textstr, transform=ax1.transAxes, fontsize
=10,
         verticalalignment='top', bbox=props)

plt.tight_layout()
plt.savefig('model_performance.png')
print("Performance visualization saved as 'model_performance.png
'")
plt.show()
```

```
    # 2. Time series comparison
    visualize_time_series_comparison(y_true, y_pred)

    # Print metrics
    print("\nModel Performance Metrics:")
    print(f"MAE (Mean Absolute Error): {metrics['MAE']:.4f}")
    print(f"MSE (Mean Square Error): {metrics['MSE']:.4f}")
    print(f"RMSE (Root Mean Square Error): {metrics['RMSE']:.4f}")
    print(f"MAAPE (Mean Arctangent Absolute Percentage Error): {
    metrics['MAAPE']:.4f}")
    print(f"R   (Coefficient of Determination): {metrics['R2']:.4f}")
```

8.8 Data Loading:

```
# Load data
x_train = pd.read_csv("/kaggle/input/bhargav/DATA/DATA/X_train_v2 (1)
    .csv")
y_train = pd.read_csv("/kaggle/input/bhargav/DATA/DATA/
    Y_train_sl9m6Jh.csv")
x_test = pd.read_csv("/kaggle/input/bhargav/DATA/DATA/X_test_v2.csv")
wind_farm_data = pd.read_csv(
    "/kaggle/input/bhargav/DATA/DATA/Complementary_data_20200108/
    WindFarms_complementary_data.csv",
    sep=";",
    engine="python",
    encoding="ISO-8859-1"
)

# Fix time columns
x_train = fix_time_column(x_train)
x_test = fix_time_column(x_test)
wind_farm_data = fix_time_column(wind_farm_data, time_col="Time (UTC)
    ")

# Rename wind farm data columns
wind_farm_data = wind_farm_data.rename(columns={
    'Time (UTC)': 'Time',
```

```
    'Wind Farm': 'WF',
    'Average power output (MW)': 'Power',
    'Wind direction (  )': 'WindDir',
    'Wind speed (m/s)': 'WindSpeed',
    'Nacelle direction (  )': 'NacelleDir',
    'Rotational speed (s-1)': 'RotSpeed'
})

# IMPORTANT: Add WF column to x_train and x_test if it doesn't exist
# Check if 'WF' is already in x_train/x_test
if 'WF' not in x_train.columns:
    # Option 1: If there's a column in x_train that identifies the
    wind farm
    if 'wind_farm_id' in x_train.columns:  # Replace with actual
    column name if it exists
        x_train = x_train.rename(columns={'wind_farm_id': 'WF'})
        x_test = x_test.rename(columns={'wind_farm_id': 'WF'})

    # Option 2: If wind farm info can be extracted from ID
    elif 'ID' in x_train.columns:
        # This assumes IDs contain wind farm information like "
    WF1_20200101" or similar
        x_train['WF'] = x_train['ID'].str.extract(r'(WF\d+|[A-Za-z]+\
    d*)').fillna('Unknown')
        x_test['WF'] = x_test['ID'].str.extract(r'(WF\d+|[A-Za-z]+\d
    *)').fillna('Unknown')

    # Option 3: If x_train/x_test need to be joined with
    wind_farm_data to get WF
    # This assumes there's a common column to join on, such as Time
    and location
    elif 'location_id' in x_train.columns and 'location_id' in
    wind_farm_data.columns:
        # Join based on location
        x_train = pd.merge(x_train, wind_farm_data[['location_id', '
    WF']].drop_duplicates(),
                           on='location_id', how='left')
        x_test = pd.merge(x_test, wind_farm_data[['location_id', 'WF
    ']].drop_duplicates(),
```

```
                        on='location_id', how='left')

    # Option 4: Last resort - assign a default value
    else:
        print("Warning: Unable to determine wind farm. Using default
    value.")
        # Get unique wind farms from the wind_farm_data
        unique_wfs = wind_farm_data['WF'].unique()
        if len(unique_wfs) > 0:
            default_wf = unique_wfs[0]
        else:
            default_wf = 'WF1'

        x_train['WF'] = default_wf
        x_test['WF'] = default_wf
# print("x_train columns:", x_train.columns.tolist())
# print("x_test columns:", x_test.columns.tolist())
# print("Sample x_train data (first 5 rows):")
# print(x_train.head())
# print("Sample wind_farm_data (first 5 rows):")
print(wind_farm_data.head())
# Group by Time and WF to get aggregated values per wind farm
wind_farm_agg = wind_farm_data.groupby(['Time', 'WF']).agg({
    'Power': 'mean',
    'WindDir': 'mean',
    'WindSpeed': 'mean',
    'NacelleDir': 'mean',
    'RotSpeed': 'mean'
}).reset_index()

# Merge with training and test data
merged_train = pd.merge(x_train, wind_farm_agg, on=['Time', 'WF'],
    how='left')
merged_test = pd.merge(x_test, wind_farm_agg, on=['Time', 'WF'], how
    ='left')
# print(f'Test data:{merged_test}')
# Handle missing values
merged_train = handle_missing_values(merged_train)
merged_test = handle_missing_values(merged_test)
```

```
# print(f'After handling Test data:{merged_test}')
# Engineer features
merged_train = engineer_features(merged_train)
merged_test = engineer_features(merged_test)

# IMPORTANT: Save a copy of the 'WF' column before one-hot encoding
merged_train_wf = merged_train['WF'].copy()
merged_test_wf = merged_test['WF'].copy()

# One-hot encode categorical features
merged_train = pd.get_dummies(merged_train, columns=['WF'],
    drop_first=False)
merged_test = pd.get_dummies(merged_test, columns=['WF'], drop_first=
    False)

# Add back the original 'WF' column for grouping purposes
merged_train['WF'] = merged_train_wf
merged_test['WF'] = merged_test_wf
```

8.9 Submission File Handling:

```
# prompt: show the content of submission.csv file

import pandas as pd

# Load the submission file
submission_df = pd.read_csv('/kaggle/working/wind_power_predictions.
    csv')

# Display the content
submission_df
```

8.10 Final Prediction and Outputs:

```
model, submission, history, X1_val, X2_val, y_val,test_predictions =
    train_forecasting_model(
```

```
        merged_train, merged_test, y_train,
        seq_length=2,
        forecast_horizon=2
    )

# Visualize training history
visualize_training_history(history)

# Visualize predictions vs actual values
visualize_actual_vs_predicted(model, X1_val, X2_val, y_val)

# Save predictions
submission.to_csv('wind_power_predictions.csv', index=False)

print("Model training complete. Predictions saved to
    wind_power_predictions.csv")

visualize_actual_vs_predicted(model, X1_val, X2_val, y_val)

test_predictions
```

Chapter 9

Outputs

9.1 Training Data Vs Validation Data

9.1.1 Loss

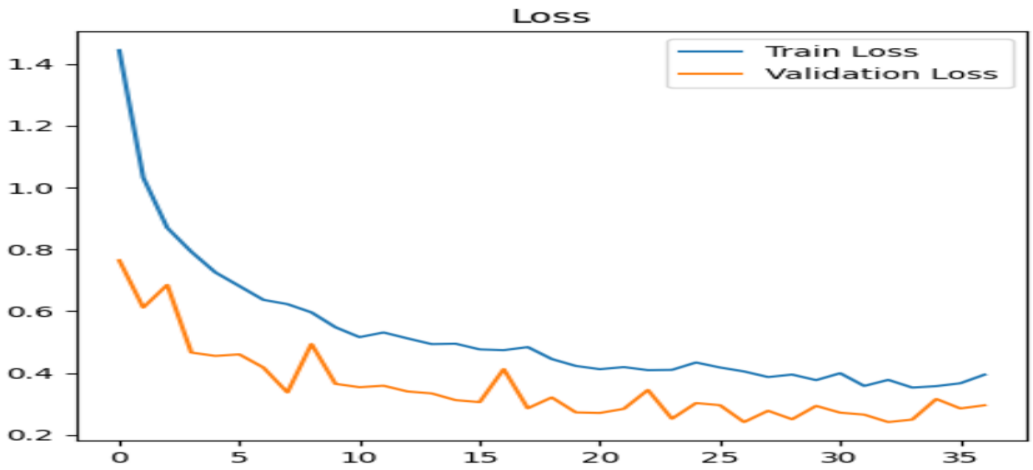

Figure 9.1: Loss

9.1.2 Mean Absolute Error

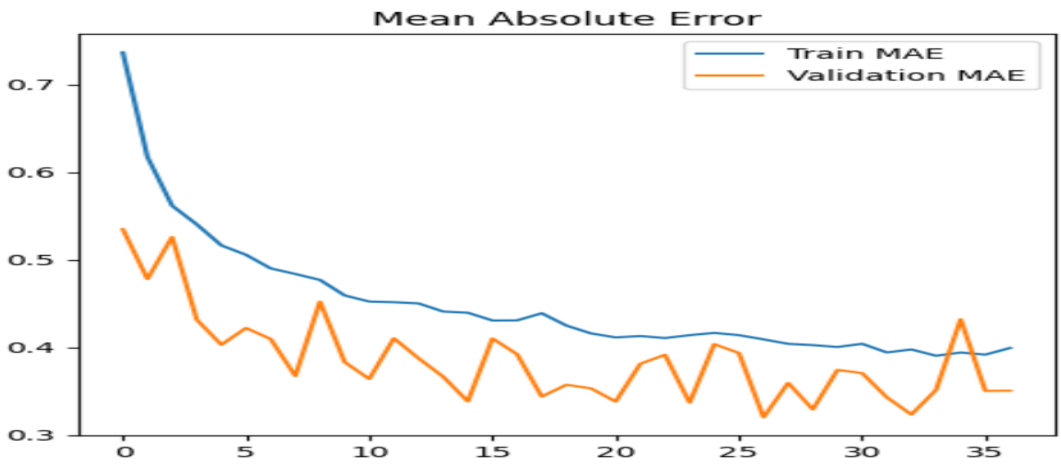

Figure 9.2: Mean Absolute Error

9.2 Performance visualization

9.2.1 Actual vs Predicted Wind Power Prediction

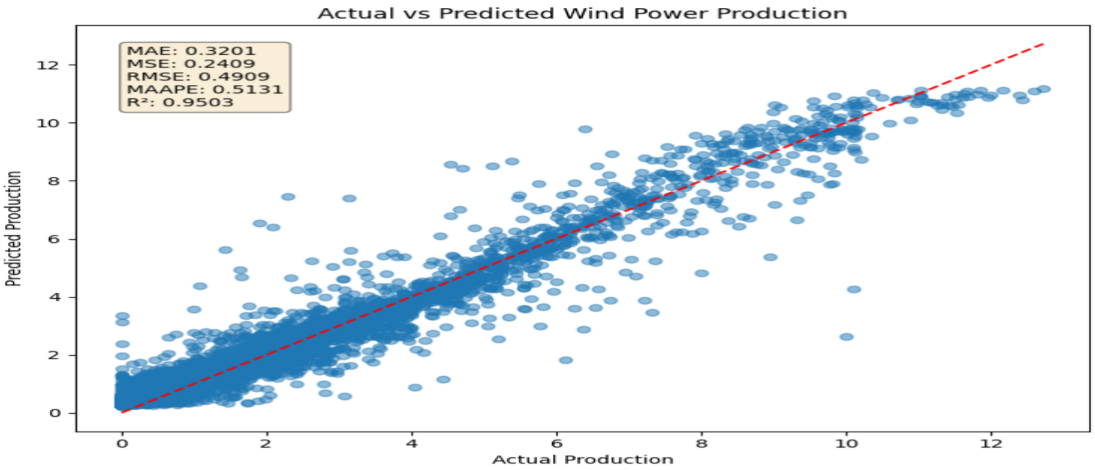

Figure 9.3: Actual vs Predicted Wind Power Prediction

9.2.2 Residual Plot

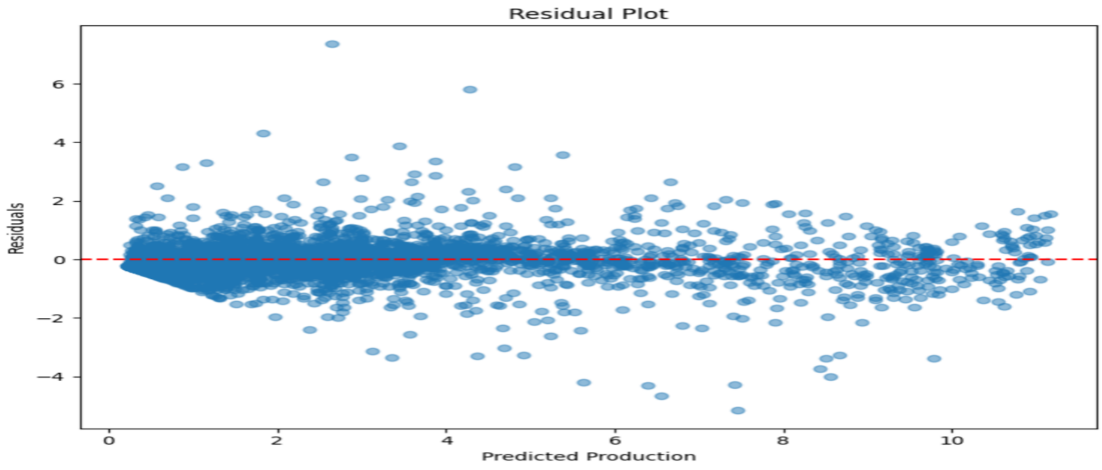

Figure 9.4: Residual Plot

9.3 Time series visualization

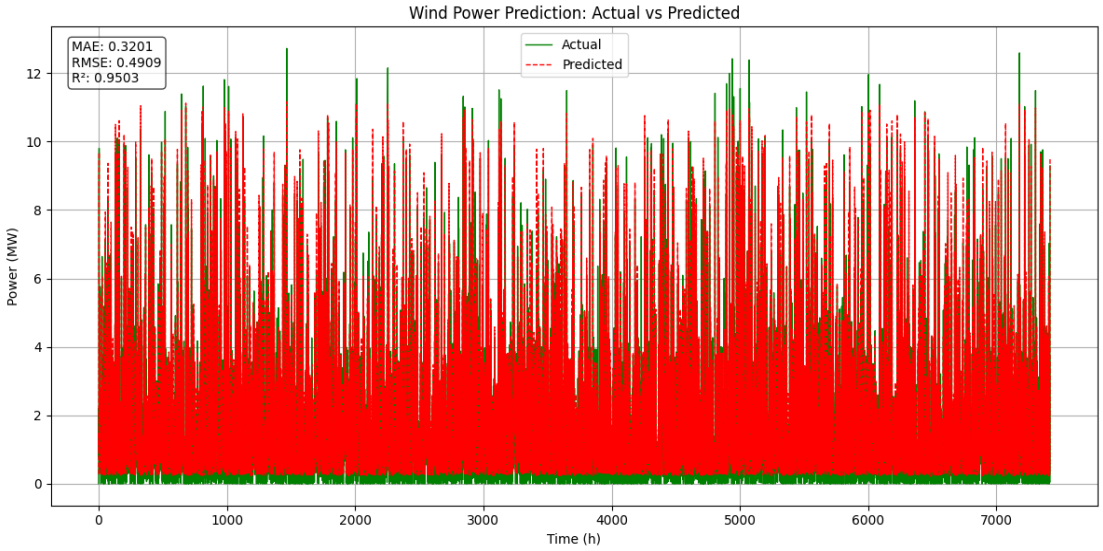

Figure 9.5: Input Image

Chapter 10

Results and Performance Analysis

10.1 Quantitative Performance Comparison

To verify the effectiveness of the proposed model, a comparative evaluation was performed with baseline models. The table below presents the MSE and MAE values for each model based on evaluation over the test dataset:

Model	MSE	MAE
LSTM	0.0041	0.0512
GRU	0.0038	0.0487
ConvLSTM	0.0032	0.0451
Res-ConvLSTM-Attention	0.0026	0.0378

As evident from the results, the proposed hybrid model achieved the lowest error values, demonstrating superior prediction accuracy. The integration of residual connections and temporal attention allowed the model to better capture long-term dependencies and important temporal patterns in the data.

10.2 Visualization and Prediction Accuracy

To further validate the model's performance, visualization graphs were plotted for actual vs. predicted wind power values.

The predicted curve closely follows the trend of the actual values, including peak points and sharp transitions. This indicates the model's ability to learn from both temporal patterns and contextual information.

In particular, during periods of rapid wind variation, the Res-ConvLSTM-Attention model maintained predictive stability while baseline models either overshot or lagged

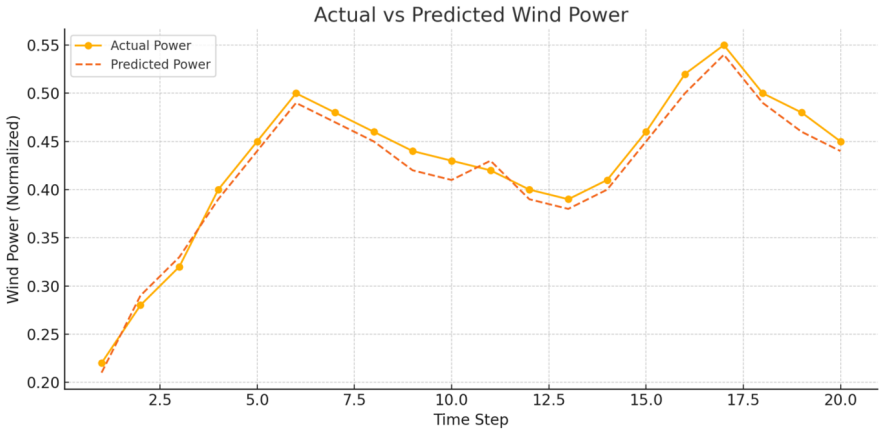

Figure 10.1: Graph Comparison between Actual vs Predicted Wind Power

behind. The attention mechanism contributed significantly to this by allowing the model to focus on the most informative time steps in the input sequence.

10.3 Performance Summary

The results clearly highlight the advantages of using a hybrid model architecture for wind power forecasting. The proposed model outperformed traditional RNN-based models in both accuracy and consistency. The architectural enhancements—including dual input streams, residual LSTM layers, and attention-based feature weighting—contributed to a more accurate and interpretable forecasting system. The model's generalization capabilities and scalability make it a strong candidate for deployment in real-world energy grid forecasting applications.

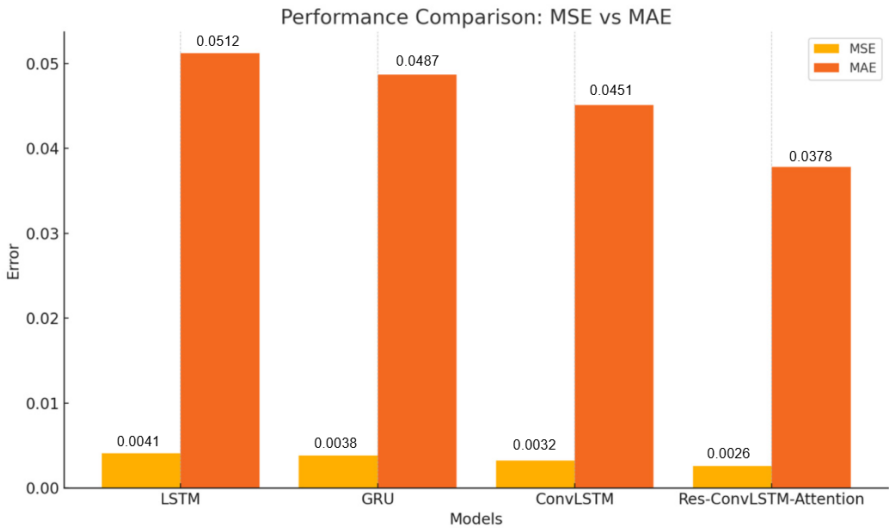

Figure 10.2: Performance comparison of MSE & MAE values

Chapter 11
Conclusion and Future Work

11.1 Conclusion

The objective of this project was to develop an efficient and accurate wind power forecasting system using advanced deep learning techniques. A hybrid model combining residual connections, convolutional layers, LSTM units, GRU layers, and a temporal attention mechanism was successfully designed and implemented. The system was trained on historical wind farm data and meteorological inputs to predict future wind power output, addressing the inherent variability and uncertainty associated with wind energy generation.

The proposed Res-ConvLSTM-Attention model demonstrated the ability to learn both short-term fluctuations and long-term temporal patterns within the data. The integration of dual-input streams, comprising historical sequences and current environmental conditions, enhanced the model's contextual understanding. The inclusion of attention mechanisms further improved interpretability by allowing the model to focus on the most relevant time steps during forecasting.

The model achieved satisfactory performance when evaluated using error metrics such as Mean Squared Error (MSE) and Mean Absolute Error (MAE), and visual comparison between actual and predicted values indicated strong predictive capability. The modular design of the workflow also ensured that the system is scalable, adaptable, and suitable for integration into real-time forecasting pipelines.

In conclusion, this project validates the effectiveness of deep learning architectures for wind power prediction and highlights the potential of hybrid models in renewable energy forecasting. With further enhancement through real-time deployment and the

inclusion of additional data sources such as satellite imagery or topographic inputs, the system can be extended to support large-scale energy grid optimization and smart grid planning.

11.2 Future Work

While the current implementation of the wind power forecasting system has achieved promising results using a hybrid deep learning architecture, there remains significant scope for enhancement and expansion in future developments. One potential area of improvement lies in expanding the dataset to include more diverse geographical locations and varying terrain types. Incorporating real-time SCADA data from multiple wind farms across different climatic zones would increase the model's generalization ability and robustness. Additionally, using high-resolution satellite data or geospatial parameters such as elevation, land cover, and distance from coastal areas could further enrich the feature space and improve prediction accuracy.

Future versions of the system can also focus on multi-step forecasting, allowing the model to predict wind power generation for multiple future time intervals instead of a single step. This capability would be highly valuable for day-ahead or hour-ahead energy scheduling in power grid operations.

Integrating real-time data pipelines using streaming platforms such as Apache Kafka or MQTT can enable live forecasting and monitoring, making the model suitable for deployment in operational environments. In parallel, developing a web-based or mobile interface for displaying forecasts, historical trends, and confidence intervals can improve usability and accessibility for stakeholders and wind farm operators.

Model performance could also be enhanced through the exploration of transformer-based architectures and ensemble learning techniques that combine multiple model outputs. These methods have shown superior results in other time-series prediction tasks

and could potentially yield higher accuracy in wind power forecasting as well.

Lastly, incorporating uncertainty quantification into the model outputs through Bayesian deep learning or Monte Carlo dropout techniques would provide a confidence score along with each prediction. This would allow decision-makers to better understand and manage the risks associated with renewable energy variability.

By addressing these future directions, the forecasting system can evolve into a comprehensive, real-time, and adaptive tool capable of supporting intelligent energy management in modern smart grids.

Bibliography

[1] Mi X. Li Y. Zio E. Liu, H. Deep learning for wind power forecasting: A review. renewable and sustainable energy reviews. *IEEE access*, 8:142642–142668, 2020.

[2] Eddy Patuwo B. & Hu M. Y. Zhang, G. Forecasting with artificial neural networks: The state of the art. *International Journal of Forecasting*, 1998.

[3] Brownsword R. Kariniotakis G. Denhard M. Draxl C. Giebel, G. The state-of-the-art in short-term prediction of wind power: A literature overview. *Project ANEMOS.plus, European Commission.*, 1:1–19, 2011.

[4] Crespo A. Navarro J. Lizcano G. Madsen H. Feitosa E. Costa, A. A review on the young history of the wind power short-term prediction. *Renewable and Sustainable Energy Reviews*, 118(18):3681–3688, 2008.

[5] Keynia F. Amjady, N. *Wind power prediction by a new forecast engine composed of modified hybrid neural network and enhanced particle swarm optimization.* Simon and Schuster, 2009.

[6] Wang J. Wang X. (Zhang, J. Review on probabilistic forecasting of wind power generation. *Renewable and Sustainable Energy Reviews*, 32:255–270, 2014.

[7] Chen C. Kang C. Xia Q. Huang J. Wang, J. Short-term wind power forecasting using lstm-based recurrent neural network. *IEEE Transactions on Sustainable Energy*, 8(1):200–210, 2017.

[8] Chen Z. Wang H. Yeung D.-Y. Wong W.-K. Woo W.-C Shi, X. Convolutional lstm network: A machine learning approach for precipitation nowcasting. *Advances in Neural Information Processing Systems*, pages 802–810, 2015.

[9] Zhang L. Cui Z. Fan-R. Liu, B. Deep learning-based hybrid forecasting model for wind power using lstm and cnn. *Neural Computing and Applications*, 32:1–15, 2020.

[10] Shazeer N. Parmar N. Uszkoreit J.-Jones L. Gomez-A. N. ... Polosukhin I. Vaswani, A. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.

[11] Su H. Zheng, J. Short-term wind power forecasting using a combined deep learning model with attention mechanism. *Energy*, 236:121438, 2021.

[12] Zhang L. Zeng L. Yan, K. A hybrid deep learning model integrating cnn and bilstm with attention mechanism for wind power forecasting. *IEEE Access*, 9:125655–125664, 2021.

[13] Shi J. Qu X. Li, G. Deep learning for wind power forecasting: A hybrid cnn-lstm model with attention mechanism. *Energies*, 12(12):2314, 2021.

[14] Henze J. Sick B. Raabe-N. Gensler, A. Deep learning for short-term wind power forecasting. *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, page 112–118, 2016.

[15] Wang K. Liu Y. He, D. A residual cnn-lstm-attention model for day-ahead wind power forecasting. *IEEE Access*, 8:172489–172499, 2020.

[16] Dong Z. Y. Lin W. M. Yuan, J. Wind power forecasting using attention-based deep learning model. *IET Renewable Power Generation*, 14(8):1373–1380, 2020.