THE UNIVERSITY OF DA NANG
VIETNAM – KOREA UNIVERSITY OF INFORMATION
AND COMMUNICATION TECHNOLOGY
FACULTY OF COMPUTER SCIENCE

GRADUATION PROJECT

DEVELOPING A SMART AGRICULTURE RECOMMENDATION SYSTEM BASED ON MULTI-VARIABLE WEATHER FORECASTING MODELS

Students	:	Duong Tan Huy – 21AD025
	:	Hoang Huu Tien Dat – 21AD011
Class	:	21AD
Faculty	:	Information Technology
Major	:	Data Science and Artificial Intelligence
Supervisor	:	MSc. Le Dinh Nguyen

Da Nang – 12/2025

THE UNIVERSITY OF DA NANG
VIETNAM – KOREA UNIVERSITY OF INFORMATION AND COMMUNICATION TECHNOLOGY
FACULTY OF COMPUTER SCIENCE

GRADUATION PROJECT

DEVELOPING A SMART AGRICULTURE RECOMMENDATION SYSTEM BASED ON MULTI-VARIABLE WEATHER FORECASTING MODELS

Students	:	Duong Tan Huy – 21AD025
	:	Hoang Huu Tien Dat – 21AD011
Class	:	21AD
Faculty	:	Information Technology
Major	:	Data Science and Artificial Intelligence
Supervisor	:	MSc. Le Dinh Nguyen

Da Nang – 12/2025

SUPERVISOR'S COMMENTS

……
……
……
……
……
……
……
……
……
……
……
……
……
……

Da Nang, December ..., 2025

Supervisor

(Name and signature)

ACKNOWLEDGEMENTS

To successfully complete this graduation thesis, I have received invaluable support and assistance from the lecturers. With deep respect and sincere gratitude, I would like to express my heartfelt thanks to all the lecturers who have created favorable conditions and provided guidance throughout my study, research, and development of this topic.

First and foremost, I would like to extend my respectful greetings, best wishes for good health, and deepest gratitude to the lecturers of the Vietnam–Korea University of Information and Communication Technology (VKU), The University of Danang. Their dedicated guidance, enthusiastic support, and valuable knowledge imparted during the past semesters have helped me broaden my understanding and gain deeper insights into the research issues. Thanks to this support, I have been able to complete this graduation internship report.

In particular, I would like to express my sincere appreciation to my supervisor, MSc. Le Dinh Nguyen, for his wholehearted guidance, encouragement, and continuous support, which greatly contributed to the successful completion of this graduation thesis.

Due to limited time and experience, this graduation thesis inevitably contains certain shortcomings. I sincerely look forward to receiving constructive feedback and valuable suggestions from the lecturers so that I can further improve my knowledge, experience, and professional skills for my future career.

I sincerely thank you!

TABLE OF CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

Abbreviation	English Term
AI	Artificial Intelligence
API	Application Programming Interface
CSS	Cascading Style Sheets
CSV	Comma-Separated Values
DB	Database
EDA	Exploratory Data Analysis
GPU	Graphics Processing Unit
IoT	Internet of Things
JSON	JavaScript Object Notation
LLM	Large Language Model
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
NLP	Natural Language Processing
RAG	Retrieval-Augmented Generation
RNN	Recurrent Neural Networks
TPU	Tensor Processing Unit
UI	User Interface
WMO	World Meteorological Organization

LIST OF TABLES

LIST OF FIGURES, DIAGRAMS

INTRODUCTION

1. Background

Vietnam is an agricultural nation where rice, corn, potatoes, and cassava play a crucial role; particularly, rice not only ensures food security but is also a strategic export commodity. However, production is increasingly impacted by climate change and erratic weather (storms, droughts, pests, etc.), making it difficult for farmers to make cultivation decisions as they still rely heavily on experience or general forecasts. Furthermore, even ordinary citizens face difficulties in their daily lives when dealing with harsh weather conditions without a truly useful personalized forecasting and recommendation system.

In this context, the development of artificial intelligence, especially Machine Learning and Deep Learning in time series forecasting, along with LLMs (Large Language Models) in natural language processing, opens up opportunities to build a smart agricultural recommendation and weather forecasting system. This system not only provides more accurate multivariate weather forecasts for agricultural production but also has the capability to convert data into specific recommendations: offering detailed cultivation schedules for farmers while providing practical suggestions for daily life. This helps the system serve a diverse range of users, contributing to improved production efficiency, risk minimization, and bringing practical benefits to the community, aiming towards smart agriculture and sustainable living

2. Research Objectives and Tasks

The objective of this thesis is to build a multivariate weather forecasting and smart agricultural recommendation system, applying machine learning and deep learning techniques to support users in making cultivation decisions and monitoring weather conditions.

To achieve the above objective, the thesis focuses on the following main tasks:

- Collecting and preprocessing weather data in the form of time series.
- Building and evaluating multivariate weather forecasting models on a daily and hourly basis.
- Converting forecast results into agricultural recommendations suitable for crops, locations, and growth stages.
- Integrating a smart chatbot to support explaining recommendations and interacting with users.
- Building a complete system that integrates all the above models together.

3. Research Objects and Scope

a. Research Objects

The research objects of the thesis include multivariate weather data represented as time series, weather forecasting models, agricultural recommendation problems based on forecast results, and a chatbot for user interaction support.

b. Research Scope

Within the scope of the graduation thesis, the topic focuses on food crops, with rice being the primary subject. Weather forecasting is performed in the short term, on a daily and hourly basis. Agricultural recommendations are intended to support decision-making and do not replace the role of agricultural experts.

4. Methodology

The thesis is implemented using a systemic approach, combining theoretical research and experimental deployment. Weather data is collected and preprocessed to serve the training of forecasting models. Machine learning and deep learning models are built, evaluated, and selected to solve the multivariate weather forecasting problem. Based on the forecast results, the system generates agricultural recommendations and integrates a smart chatbot to support explanation and user interaction. All components are integrated into a unified application system.

5. Content and Implementation Plan

NO	Time	Implementation Content
1	Week 1 22/9 - 28/9	- Research related documents and studies - Write and finalize the graduation thesis outline.
2	Week 2 29/9 - 5/10	- Analyze requirements and system design. - Design the system database. - Design module pipelines and the overall system pipeline.
3	Week 3 6/10 - 12/10	- Collect weather data (hourly, daily). - Preprocess weather data, store time-series weather data.
4	Week 4 – 5 13/10 - 26/10	- Build weather forecasting models (time series and classification).
5	Week 6 – 7 27/10 - 9/11	- Develop the agricultural recommendation module (based on crop type, geographical location, growth stage, and weather forecast data).
6	Week 8 – 9 10/11 - 23/11	- Build an LLM Chatbot for agricultural consultation, integrating it with the recommendation module..
7	Week 10 24/11 - 30/11	- Develop the backend system and database (metadata, vector DB, recommendation schedule, users, conversations).
8	Week 11 1/12 - 7/12	- Develop the frontend website.
9	Week 12 8/12 - 14/12	- Connect AI modules, backend, and frontend to form a complete system. - Test the system, detect and fix bugs, optimize performance.
10	Week 13 15/12 - 21/12	- Finalize the product, write the report, and prepare presentation slides.
11	Week 14 22/12 - 27/12	- Prepare for the defense and summarize the thesis project.

6. Report Structure

Following the ***Introduction***, the thesis content consists of 5 main chapters:

Chapter 1: ***Overview of the Topic***

Chapter 2: ***System Analysis and Design***

Chapter 3: ***Theoretical Basis and Technologies Used***

Chapter 4: ***Training Multivariate Weather Forecasting Models***

Chapter 5: ***Building the Smart Recommendation System***

Finally, the ***Conclusion*** and ***References*** related to the topic are presented.

CHAPTER 1 – TOPIC OVERVIEW

1.1 Problem Overview

Weather is a factor that directly influences and determines the efficiency of agricultural production activities. Meteorological factors such as temperature, rainfall, air humidity, and wind strongly impact crop growth and development, as well as the yield and quality of agricultural products. In recent years, under the impact of climate change, weather has tended to fluctuate in complex and unpredictable ways, increasing risks in agricultural production, especially for farming models that rely heavily on natural conditions.

Along with the development of data collection and storage technology, weather data is now often represented as multivariate time series, reflecting multiple meteorological factors simultaneously over a continuous period. Effectively exploiting multivariate weather data allows for a more complete description of weather states and trends, thereby better supporting forecasting work. However, the multi-dimensional nature and high volatility of the data also increase the complexity of the analysis and utilization of forecast results.

In reality, current weather forecasting systems mainly provide information in the form of quantitative forecasts or general descriptions, which are not closely tied to the specific context and needs of agricultural production. Users, especially farmers, face many difficulties in converting this forecast information into appropriate cultivation decisions, such as selecting planting times, adjusting irrigation schedules, or preventing risks caused by adverse weather conditions.

Therefore, the problem is not only to improve the accuracy of weather forecasting but also to integrate forecast results with cultivation knowledge and experience to provide practical agricultural recommendations. Building systems capable of combining multivariate weather forecasting with smart agricultural recommendations is considered a necessary approach, contributing to helping users make more effective decisions in the context of modern agricultural production.

1.2 Overview of Current Approaches

1.2.1 Approaches in Weather Forecasting

- Traditional Methods:

+ Based on classical statistical methods and time series analysis. Exploiting trends, seasonality, and fluctuations of historical meteorological data.

+ Meteorological factors such as temperature, rainfall, and humidity are often forecasted individually or through linear combinations with a limited number of variables. It rarely considers complex dependency relationships between multiple variables over time.

+ Pros: Simple models, easy to build and deploy. Does not require large amounts of data. Low computational cost, suitable for small-scale systems.

+ Cons: Difficult to model nonlinear relationships between meteorological factors. Poor forecasting capability when weather fluctuates strongly or abnormally. Unsuitable for multivariate weather forecasting problems in the context of climate change.

- Machine Learning Methods:

+ Exploit historical data to learn patterns from multiple meteorological features.

+ Applied to weather state classification and short-term forecasting problems.
+ Improved accuracy compared to traditional methods.
+ Still limited in capturing long-term dependencies over time.

- Deep Learning Methods:

+ Use time series models to process multivariate weather data.
+ Capable of learning complex relationships and long-term dependencies.
+ Suitable for big data, multi-dimensional data on a daily and hourly basis.
+ Require sufficient computational resources and training data.

1.2.2 Approaches in Agricultural Recommendations

- Build rule sets from expert knowledge and farming experience.

+ Recommendations based on weather thresholds or fixed seasonal schedules.
+ Easy to understand and apply locally.
+ Lack adaptability to fluctuating weather conditions.

- Data-driven Decision Support Systems:

+ Combine weather data with crop information and growth stages.
+ Create more flexible recommendations compared to fixed rule systems.
+ Heavily dependent on the quality and completeness of input data.

- AI-integrated Systems:

+ Apply AI to convert forecast data into specific recommendations.
+ Scalable for various crops and regions.
+ Place high demands on explainability and recommendation reliability.

1.2.3 Approaches in Integration and User Interaction

- Combine weather forecasting and recommendations into a unified system.
- Support users through interactive interfaces and intelligent consultation.
- Trend towards personalized recommendations based on location, time, and usage needs.
- Require the system to be accurate, easy to understand, and usable in practice.

1.3 Limitations of Current Approaches

1.3.1 Limitations in Weather Forecasting

Although weather forecasting methods have seen many improvements, the accuracy of forecast results remains unstable, especially in the context of strong weather fluctuations and the appearance of many extreme phenomena. Many models still struggle to process multivariate weather data and have not fully exploited the dependent relationships and complex interactions between meteorological factors over time. Furthermore, most systems only focus on a specific forecast frame (daily or hourly), limiting flexible application for practical decisions.

1.3.2 Limitations in Agricultural Recommendation Systems

Current agricultural recommendation systems largely rely on fixed rule sets or farming experience, leading to generic recommendations that are difficult to personalize for specific conditions. The integration between weather forecasting and recommendations remains loose, causing recommendations to not fully reflect weather developments in the short and medium term. Additionally, many systems do not manage

cultivation schedules consistently throughout growth stages, reducing the ability to adjust and inherit plans when actual conditions change.

1.3.3 Limitations in User Interaction and Support

Regarding interaction, many systems stop at providing one-way information, lacking mechanisms to support users in asking questions and understanding the basis of the recommendations provided. The ability to explain "why" a recommendation is proposed is limited, reducing the user's level of trust and acceptance. At the same time, current systems do not effectively support updating information from users to adjust recommendations and cultivation schedules flexibly.

1.4 Problem Statement and Research Direction

From the analyzed limitations, it can be seen that the problem is not just about improving weather forecast accuracy, but also about the ability to convert forecast results into specific, understandable recommendations that can be directly applied in practice. This requires a comprehensive approach, where weather forecasting and agricultural recommendations are considered as a unified system.

The thesis focuses on solving the multivariate time series weather forecasting problem with multiple forecast frames, aiming to provide detailed weather information close to usage needs. On that basis, forecast results are exploited to build agricultural recommendations suitable for each crop type, geographical location, and growth stage, rather than just providing pure weather information.

Furthermore, the thesis aims to enhance interaction between the system and users through an intelligent chatbot, allowing for recommendation explanation, receiving feedback, and supporting cultivation plan adjustments when actual conditions change. This approach aims to improve practicality, usability, and system reliability in both agricultural production and daily life.

With the above direction, the thesis not only researches multivariate weather forecasting models but also builds a smart agricultural recommendation system, integrating forecasting, knowledge, and user interaction, serving as the basis for the analysis and system design steps presented in the next chapter.

CHAPTER 2 – SYSTEM ANALYSIS AND DESIGN

2.1 Problem Description

In the context where agricultural production and social life are increasingly impacted by climate change, the need for detailed, accurate weather forecasting capable of being converted into specific actions has become urgent. In reality, current popular weather forecasting sources mainly provide general information, which is difficult to apply directly to specific farming activities or daily life decisions. For farmers, decision-making still relies heavily on personal experience, lacking support from intelligent and personalized information systems.

The problem is to build an intelligent information system capable of exploiting multivariate weather data as time series, thereby:

- Forecasting short-term weather conditions (daily and hourly).
- Classifying weather states that directly affect production and daily life.
- Converting forecast results into specific, understandable, and highly applicable recommendations for each user group.

The system serves two main target groups. First are farmers and agricultural technical staff, who need detailed recommendations by crop type, geographical location, and growth stage to support planning for cultivation, irrigation, fertilization, and pest control. Second are general users, who need to monitor detailed weather forecasts and receive basic suggestions to adjust daily activities and avoid adverse weather conditions.

Thus, the thesis problem is defined as designing and building a smart agricultural recommendation and weather forecasting system, where forecast results are transformed into knowledge and specific actions through recommendation modules and a chatbot, meeting the practical needs of both agricultural production and daily life.

2.2 Actor Identification

2.2.1 Farmers and Agricultural Technical Staff (Agricultural User)

This is the primary actor group of the system. This group uses the system to:

- Monitor detailed weather forecasts on a daily and hourly basis.
- Initialize and manage farming projects (crop type, geographical location, growth stage).
- Receive farming schedules and recommendations related to irrigation, fertilization, and pest control.
- Look up, explain, and adjust farming schedules based on actual conditions via the website interface or chatbot.
- This actor group requires information to be accurate, timely, and easily applicable to practical production.

2.2.2 General Users (General User)

General users are the actor group using the system for the purpose of:

- Monitoring weather forecasts on a daily and hourly basis.
- Receiving basic recommendations to serve daily life activities and avoid adverse weather.
- Asking questions and interacting with the chatbot to look up related information and knowledge.

This group does not require deep specialized knowledge; therefore, the system needs to provide concise, understandable, and visual information.

2.2.3 System Administrator (Admin)

The administrator is the actor responsible for operating and managing the system, including:

- Managing users and access permissions.
- Managing weather data collected and stored in the system.
- Managing the knowledge base serving the chatbot.
- Monitoring the system's operation status and handling technical incidents.

This actor ensures the system operates stably, securely, and continuously.

2.3 Use Cases

2.3.1 List of Use Cases

- User Authentication: Actors (General User, Agricultural User, Admin)
- View Weather Forecast: Actors (General User, Agricultural User, Admin)
- Manage Farming Projects and Agricultural Recommendations: Actors (Agricultural User, Admin)
- Chat with Chatbot: Actors (General User, Agricultural User, Admin)
- User Management: Actors (General User, Agricultural User, Admin)
- Data and System Management: Actors (Admin)

2.3.2 Use Case Specifications

- View Weather Forecast.

Use case	View Weather Forecast
Actors	General User, Agricultural User, Admin
Goal	To allow users to view daily and hourly weather forecast information for an area of interest, supporting weather monitoring and decision-making for daily life or farming.
References	None
Preconditions	For logged-in users, the system may use the saved location; for non-logged-in users, the user must provide the location for the forecast.
Postconditions	Daily and hourly weather forecast information is displayed to the user. Forecast data can be used as input for other use cases.
Description`	Users access the weather forecast viewing function and select or enter the area to monitor. The system receives the request, retrieves corresponding weather data from the database or updated data source, then processes and displays the forecast results on a daily and hourly basis. Information is presented in an understandable format, helping users quickly grasp weather conditions during the forecast period.

Table 2.1: Use Case View Weather Forecast

- Manage Farming Projects and Agricultural Recommendations.

Use case	Manage Farming Projects and Agricultural Recommendations
Actors	Agricultural User, Admin
Goal	To allow users to initialize and manage farming projects based on basic information such as location, crop type, and timing; thereby, the system generates and manages appropriate agricultural recommendation schedules to support effective farming and risk minimization.
References	Daily and hourly weather forecast data provided by the forecast system Knowledge base and agricultural recommendation rules integrated via external API. Technical documents and cultivation guidelines for food crops.
Preconditions	User is logged into the system as an Agricultural User or Admin.
Postconditions	The farming project is successfully created and stored in the system. An agricultural recommendation schedule is generated based on project information and weather data.
Description	Users initialize a new farming project by entering basic information including farming location, crop type, and cultivation start time. After the project is successfully created, the system uses project information combined with weather forecast data to generate an agricultural recommendation schedule corresponding to each growth stage of the crop. Users can view proposed recommendations, adjust content or implementation time to suit actual conditions, and then confirm the application. Confirmed recommendations are stored by the system and linked to the farming project, serving progress tracking and decision support in subsequent stages.

Table 2.2: Use Case Manage Farming Projects and Agricultural Recommendations

- Chat with Chatbott.

Use case	Chat with Chatbot
Actors	General User, Agricultural User, Admin
Goal	To allow users to interact with the intelligent chatbot to look up information and answer questions related to weather, agriculture, farming projects, and agricultural recommendations, in order to support quick and convenient decision-making..
References	Chatbot user manual within the system. Agricultural knowledge base and farming data integrated into the system.

	Documents describing the AI model and information query mechanism.
Preconditions	The chatbot system and AI services are functioning normally.
Postconditions	The chatbot's answer is displayed to the user. The user can continue the exchange or end the chat session.
Description	Users enter questions or request information through the chatbot interface. The system receives content, analyzes context, and identifies the query type (general information, weather forecast, farming project, or agricultural recommendation). Based on the query content, the system can combine internal data and send requests to the AI service to generate appropriate answers. The processed result is displayed as a conversational response, helping users easily access and understand information. Users can continue to ask questions for multi-turn exchanges with the chatbot.

Table 2.3: Chat with Chatbot

2.4 Use Use Case Diagrams

2.4.1 System Overview Use Case Diagram

- Use Case Diagram for the Smart Agricultural Recommendation System based on Multivariate Weather Forecasting Models.

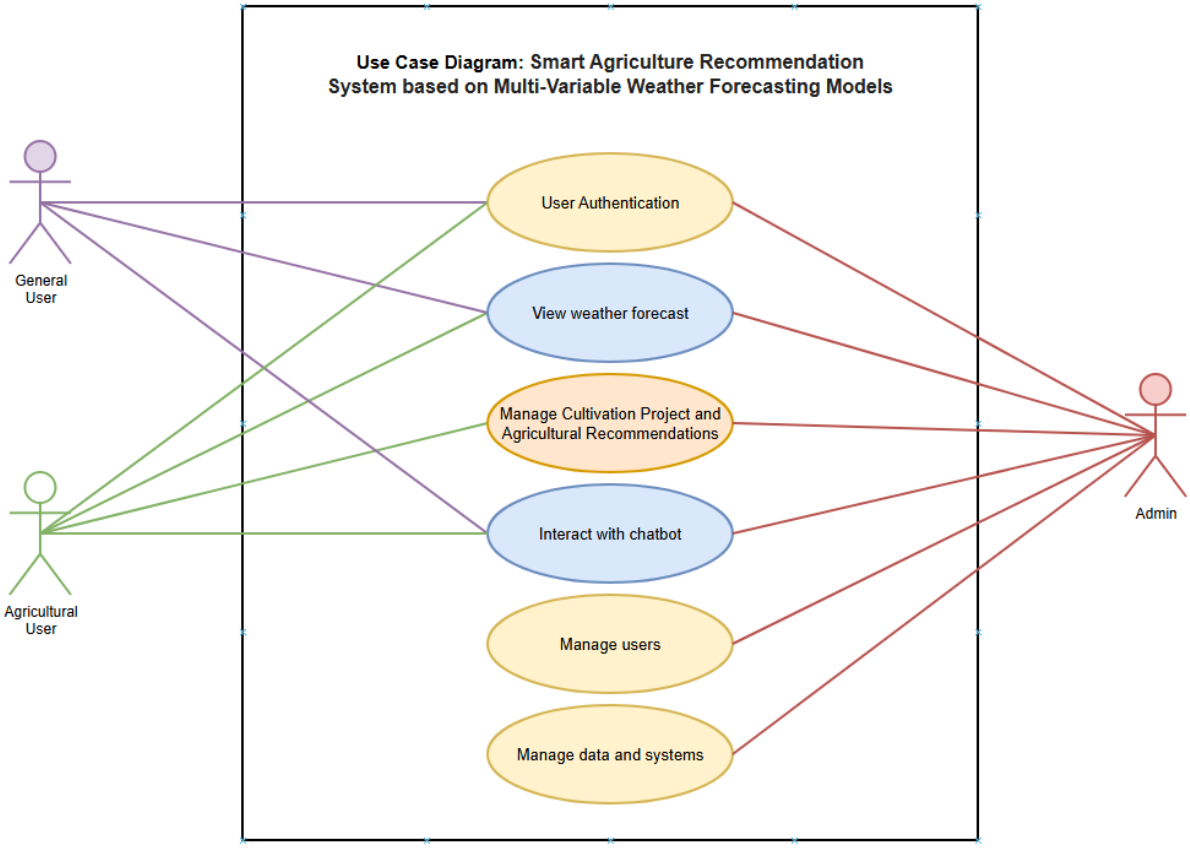


Figure 2.1: Use Case Diagram for the Smart Agricultural Recommendation System based on Multivariate Weather Forecasting Models

2.4.2 Decomposed Use Case Diagrams

- Decomposed Use Case Diagram for Viewing Weather Forecast.

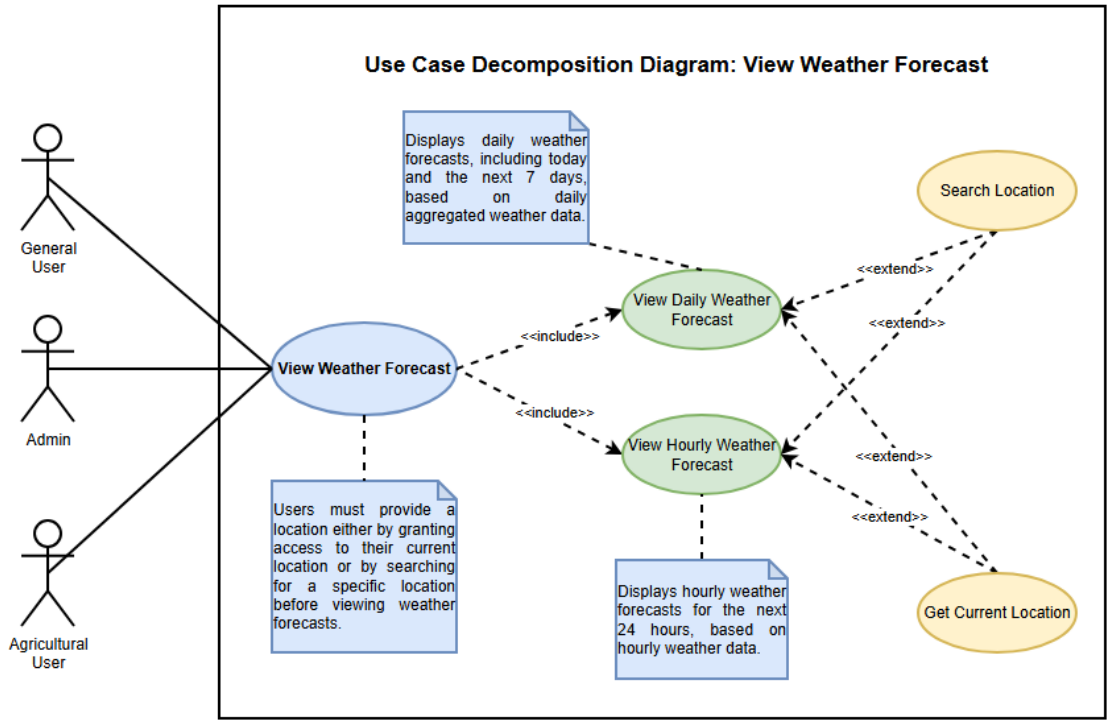


Figure 2.2: Decomposed Use Case Diagram for Viewing Weather Forecast

- Decomposed Use Case Diagram for Managing Farming Projects and Agricultural Recommendation.

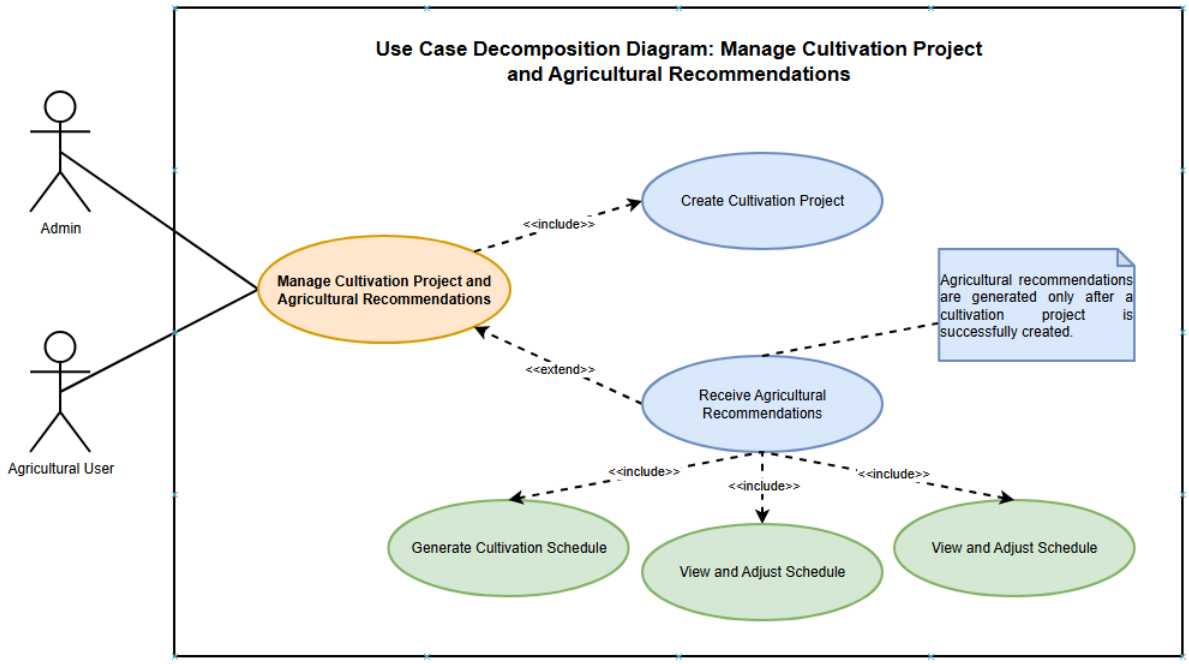


Figure 2.3: Decomposed Use Case Diagram for Managing Farming Projects and Agricultural Recommendations

- Decomposed Use Case Diagram for Chatting with Chatbot.

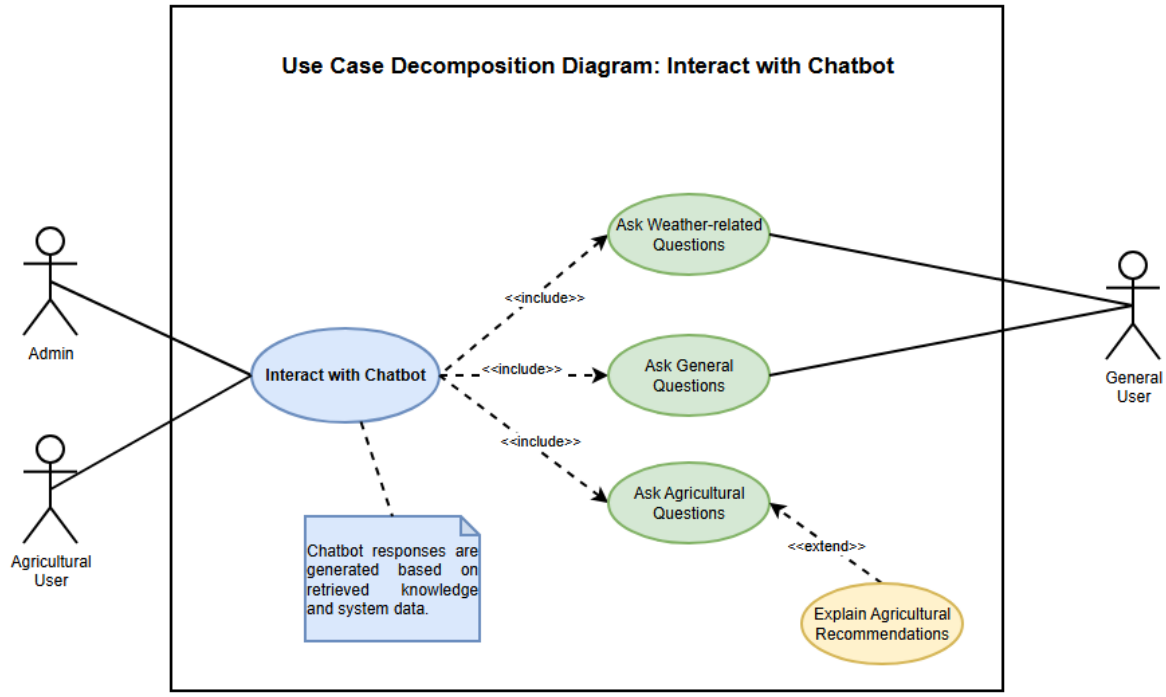


Figure 2.4: Decomposed Use Case Diagram for Chatting with Chatbot

2.5 Structural Modeling

2.5.1 Identifying Classes

Class Name	Attributes	Description
Role	roleID role	This class is used to determine the user's role in the system, serving the purpose of authorization and access control for functions.
User	userID username password email status createdAt	This is the base class representing all users of the system.
GeneralUser	(Inherits from User)	This class inherits from User; this class mainly uses the system to view weather forecasts and chat with the chatbot, and does not participate in agricultural

		farming management functions.
AgriculturalUser	(Inherits from User)	This class inherits from User; this class adds the farming location to provide context for weather forecasting and building appropriate agricultural recommendation schedules.
Admin	(Inherits from User)	This class inherits from User; this class is responsible for managing users, data, and system configurations to ensure the system operates stably and securely.
ChatSession	sessionID createdAt updatedAt	This class represents an exchange session between a user and the chatbot.
ChatMessage	messageID sender content timestamp	This class represents individual messages within a chat session.
FarmingProject	projectID projectName location cropType startDate status	This class represents a farming project initiated by an agricultural user.
RecommendationSchedule	scheduleID startDate endDate createdDate	This class represents an agricultural recommendation schedule for a specific period (usually 7 days).
DailyRecommendation	dailyRecommendationID date content actionType	This class represents specific agricultural recommendations for each day.

Table 2.4: Identifying Classes

2.5.2 Relationships between Classes

Relation	Relationship Type	Role
Role – User	Association (1-*)	assigns
User – GeneralUser	Inheritance	inherits from
User – AgriculturalUser	Inheritance	inherits from
User – Admin	Inheritance	inherits from
AgriculturalUser – FarmingProject	Association (1-*)	manages
FarmingProject – RecommendationSchedule	Association (1-1)	generates
RecommendationSchedule – DailyRecommendation	Composition (1-*)	consists of
User – ChatSession	Association (1-*)	initiates
ChatSession – ChatMessage	Composition (1-*)	contains

Table 2.5: Relationships between Classes

2.5.3 Biểu đồ lớp phân tích

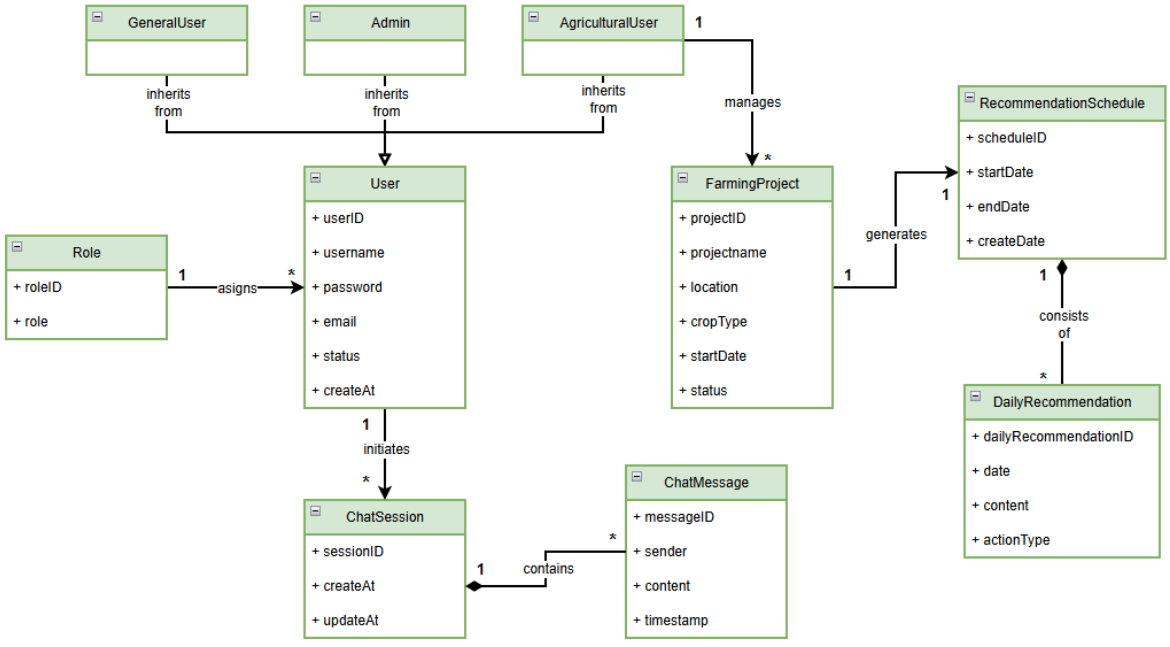


Figure 2.5: Analysis Class Diagram

2.6 Behavioral Modeling

2.6.1 Activity Diagrams

- Activity Diagram for Viewing Weather Forecast.

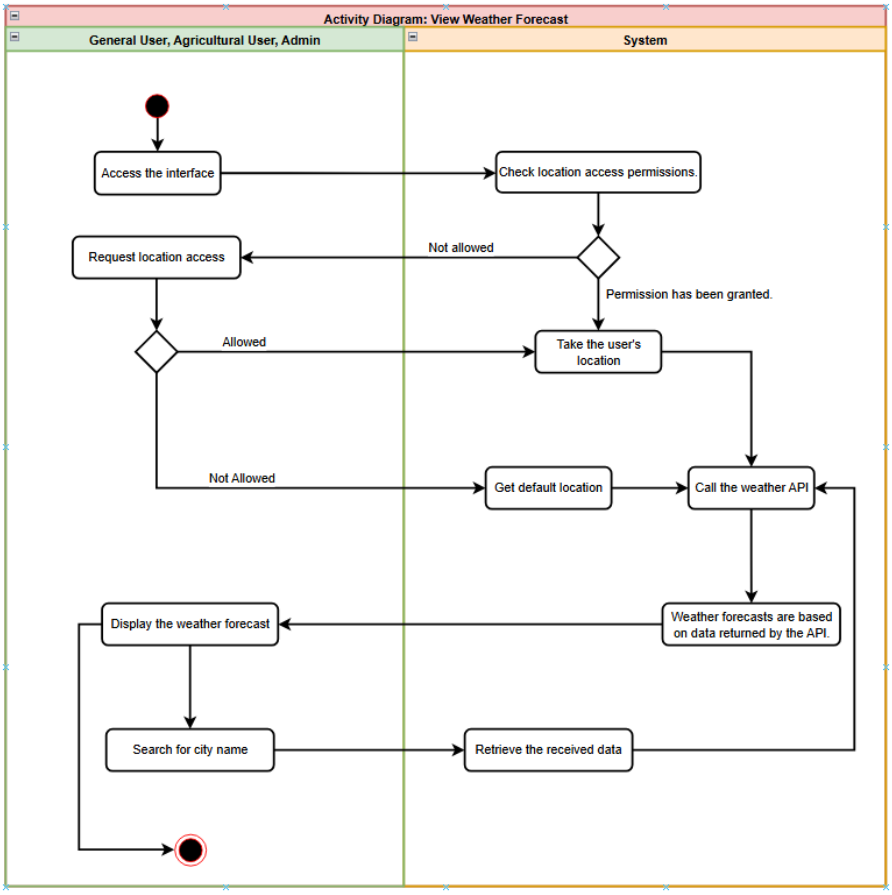


Figure 2.6: Activity Diagram for Viewing Weather Forecast

- Activity Diagram for Managing Farming Projects and Agricultural Recommendations

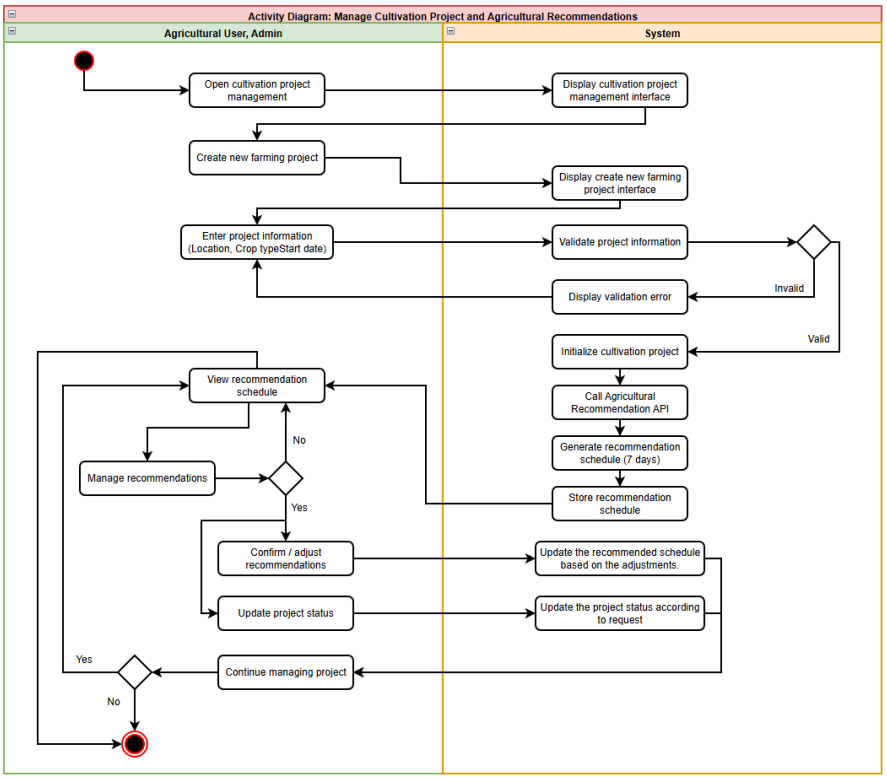


Figure 2.7: Activity Diagram for Managing Farming Projects and Agricultural Recommendations

- Activity Diagram for Chatting with Chatbot.

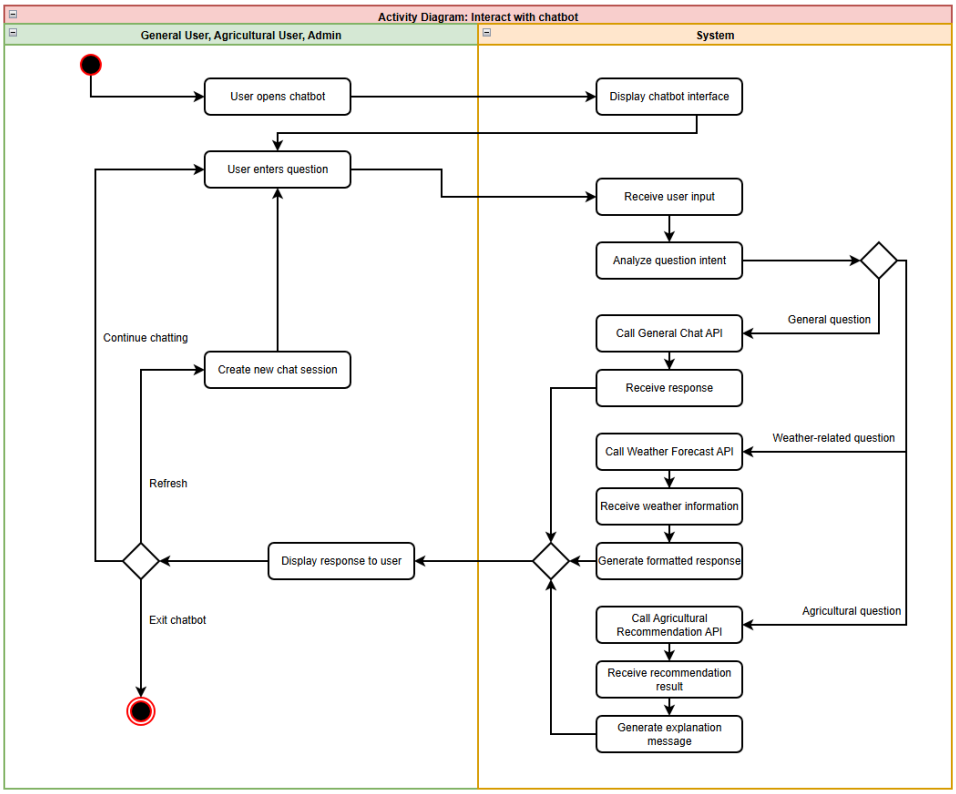


Figure 2.8: Activity Diagram for Chatting with Chatbot

2.6.2 State Diagrams

- User State Diagram.

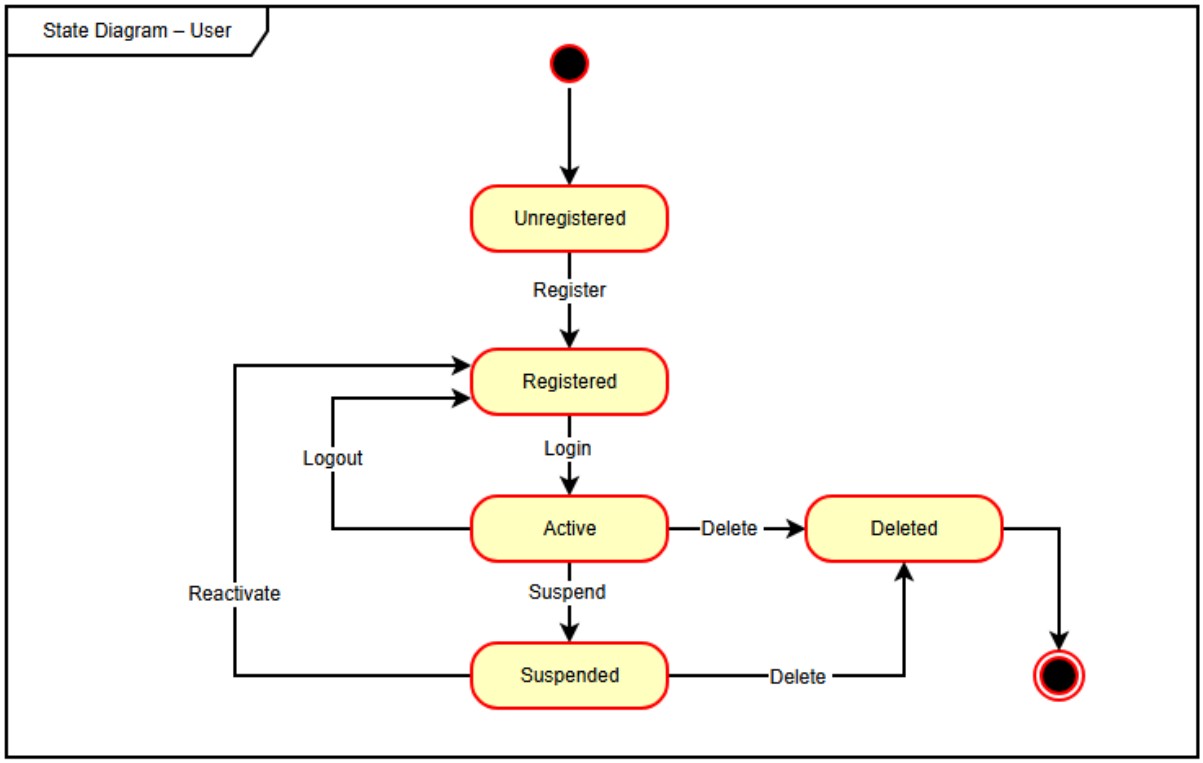


Figure 2.9: User State Diagram

- ChatSession State Diagram.

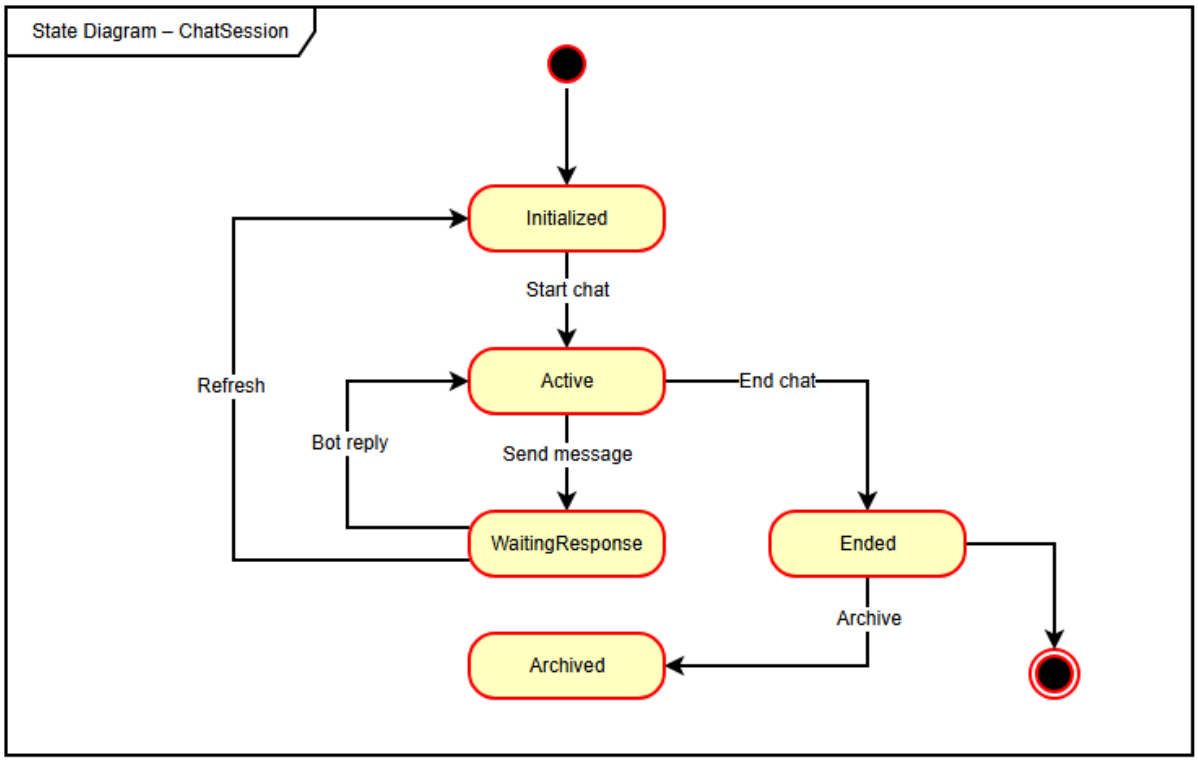


Figure 2.10: ChatSession State Diagram

- ChatMessage State Diagram

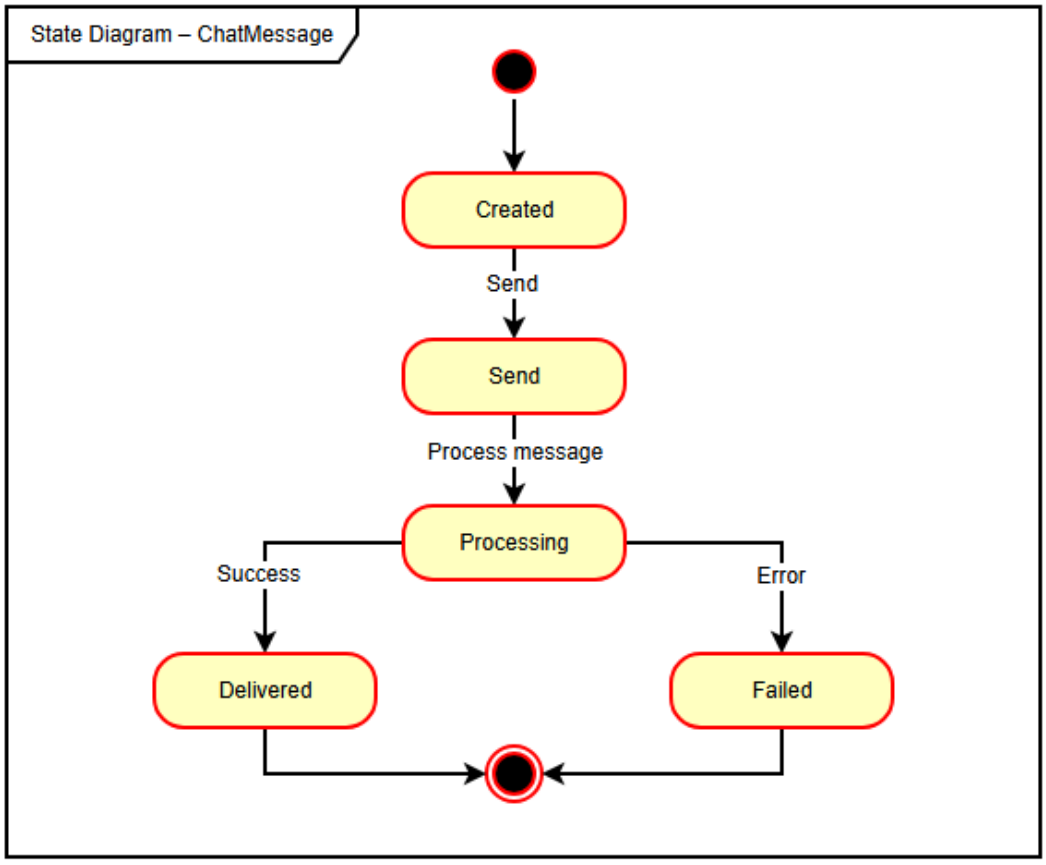


Figure 2.11: ChatMessage State Diagram

- FarmingProject State Diagram

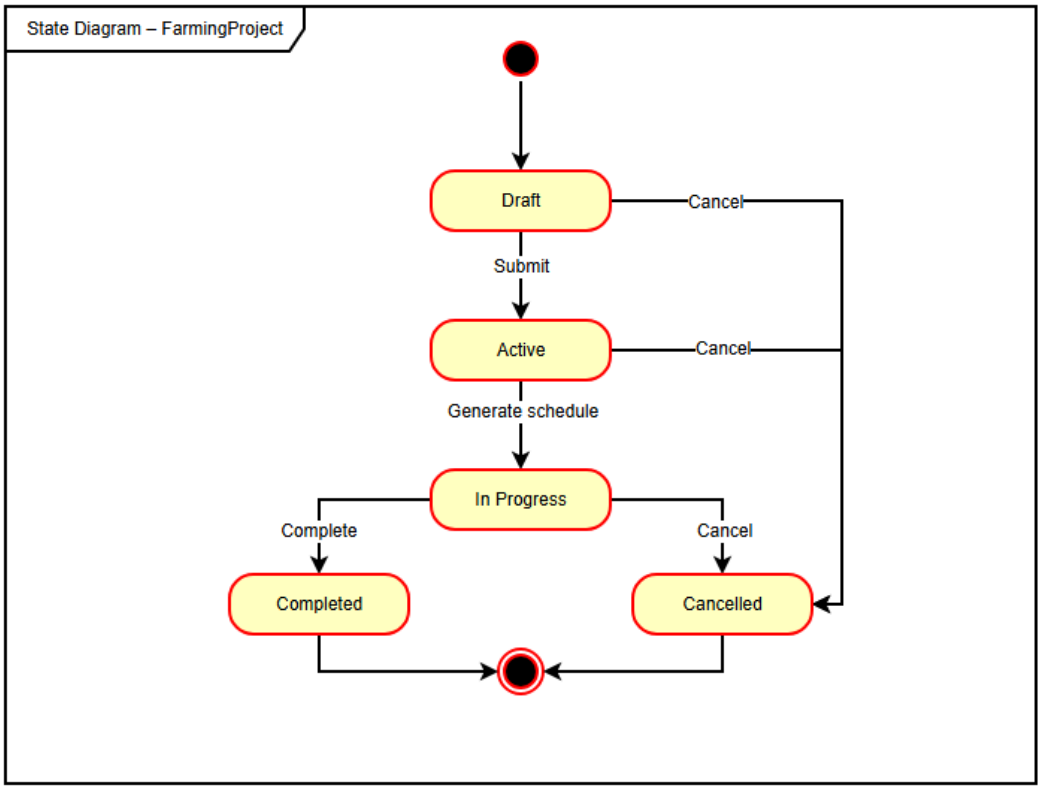


Figure 2.12: FarmingProject State Diagram

- RecommendationSchedule State Diagram.

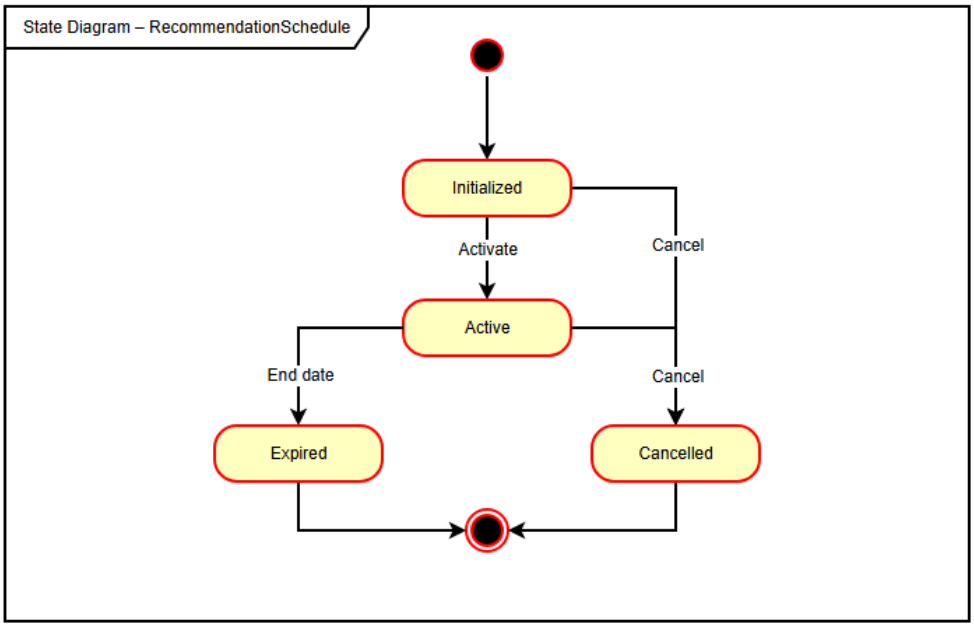


Figure 2.13: RecommendationSchedule State Diagram

2.6.3 Sequence Diagrams

- Sequence Diagram for Viewing Weather Forecast.

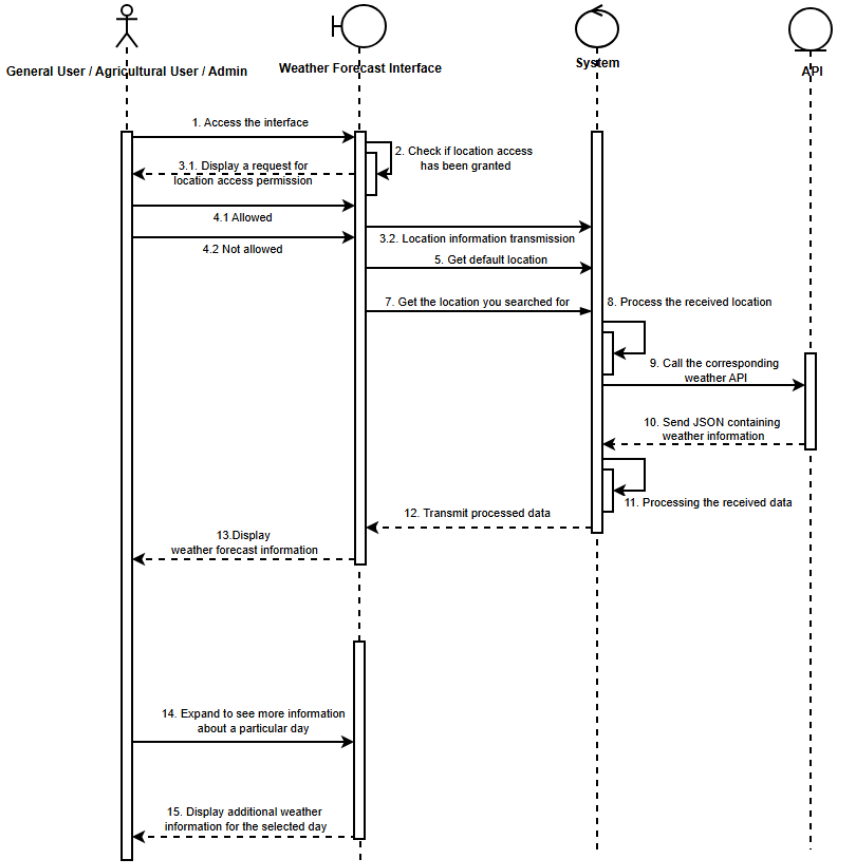


Figure 2.14: Sequence Diagram for Viewing Weather Forecast

- Sequence Diagram for Managing Farming Projects and Agricultural Recommendations

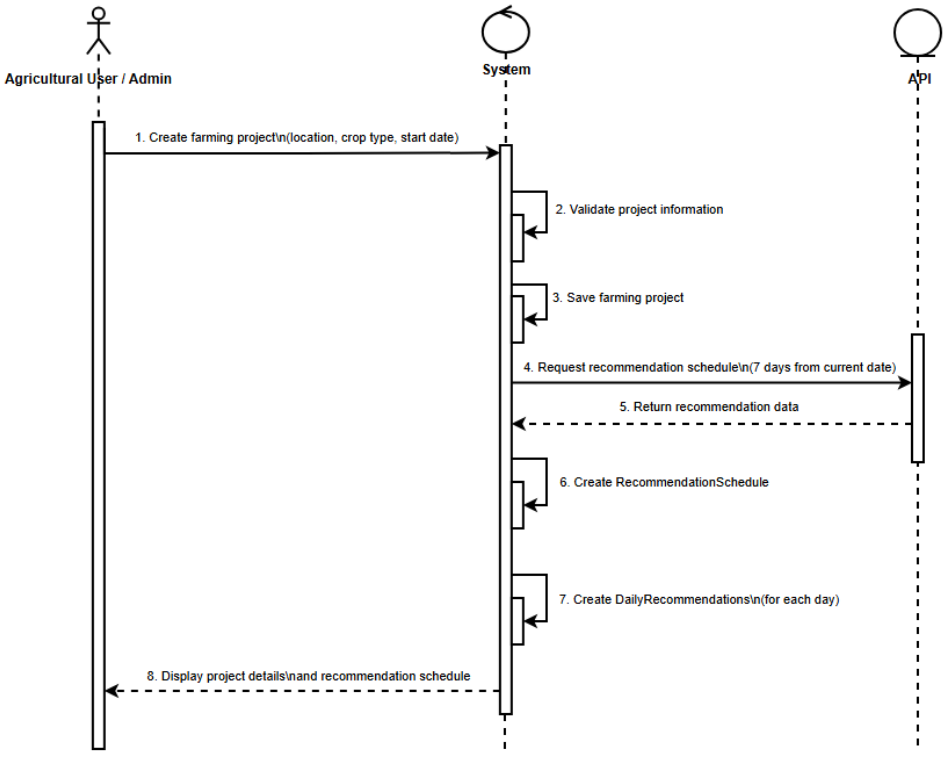


Figure 2.15: Sequence Diagram for Managing Farming Projects and Agricultural Recommendations

- Sequence Diagram for Chatting with Chatbot.

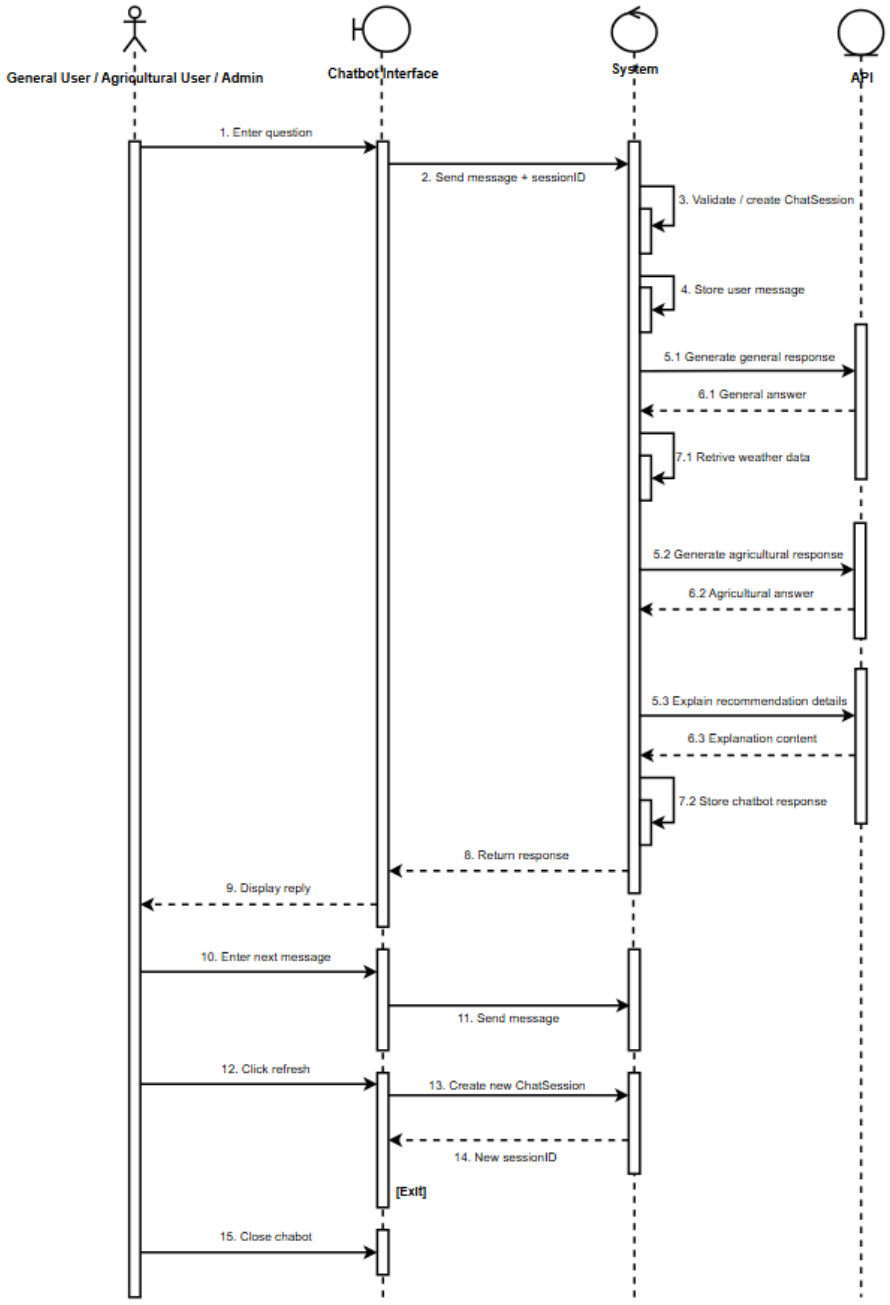


Figure 2.16: Sequence Diagram for Chatting with Chatbot

2.6.4 Communication Diagrams

- Communication Diagram for Viewing Weather Forecast.

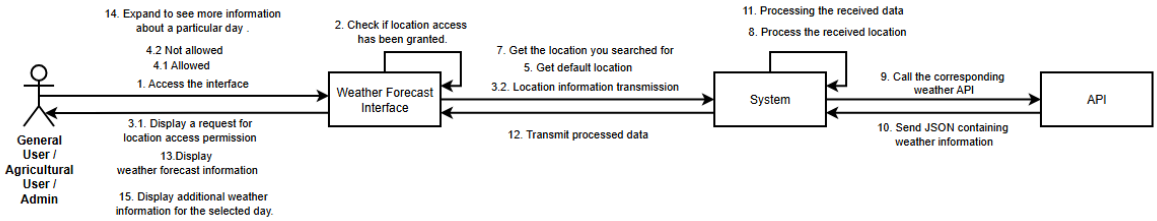


Figure 2.17: Communication Diagram for Viewing Weather Forecast

- Communication Diagram for Chatting with Chatbot.

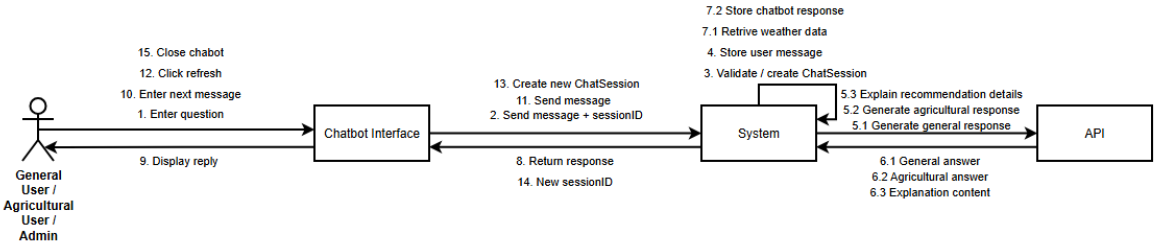


Figure 2.18: Communication Diagram for Chatting with Chatbot

- Communication Diagram for Managing Farming Projects and Agricultural Recommendations.

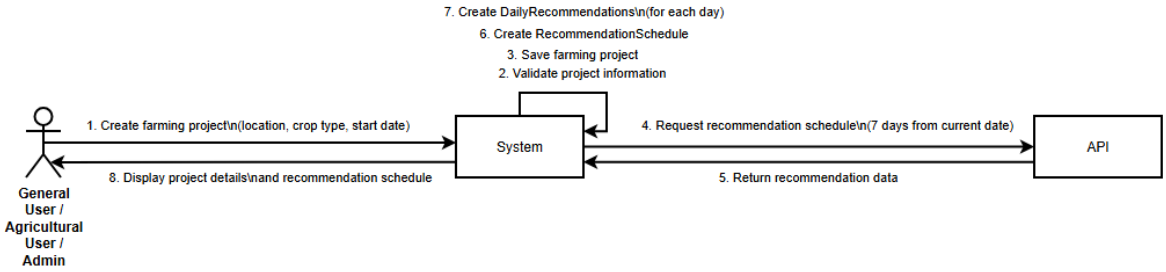


Figure 2.19: Communication Diagram for Managing Farming Projects and Agricultural Recommendations

2.7 Design Class Diagram

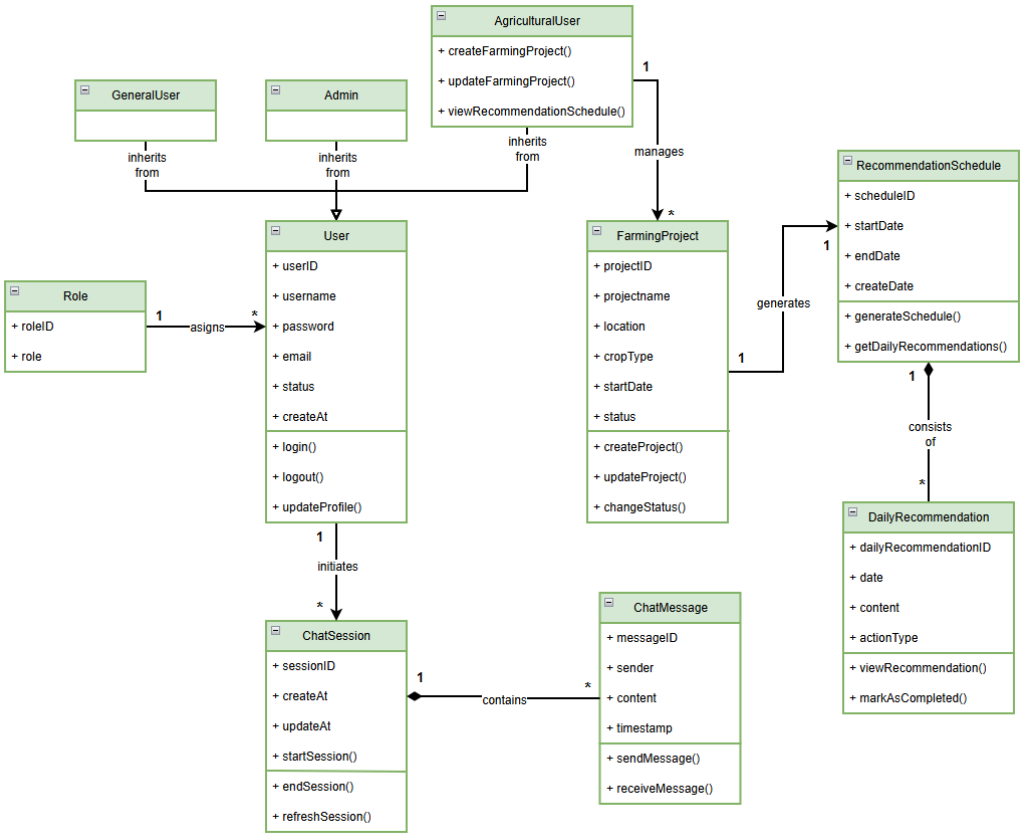


Figure 2.20: Design Class Diagram

CHAPTER 3 – THEORETICAL BASIS AND TECHNOLOGIES USED

3.1 Theoretical Basis

3.1.1 Multivariate Time Series in Weather Problems

Weather forecasting is a time series forecasting problem where future values are estimated based on historical data. In this thesis, the problem belongs to the multivariate time series type because, at each point in time, multiple weather factors exist simultaneously, such as temperature, humidity, rainfall, pressure, and wind speed.

Weather variables are closely related and change over time, so forecasting needs to consider both the sequential nature of data and the correlations between variables. Data is collected at two levels: hourly and daily, corresponding to short-term forecasting and trend forecasting requirements.

The multivariate weather forecasting problem is characterized by noisy data, nonlinear relationships, and long-term time dependencies. Therefore, machine learning and deep learning methods are selected to effectively exploit these features, serving as the basis for the smart agricultural recommendation system.

3.1.2 LSTM model for multivariate time series forecasting

LSTM (Long Short-Term Memory) is a Recurrent Neural Network (RNN) architecture designed to overcome limitations in remembering long-term information. While RNNs use a simple hidden state to pass information through time steps, LSTM adds a memory cell along with three control gate mechanisms: the forget gate, input gate, and output gate. These gates allow the model to select information to remember or discard, thereby maintaining context in long data sequences.

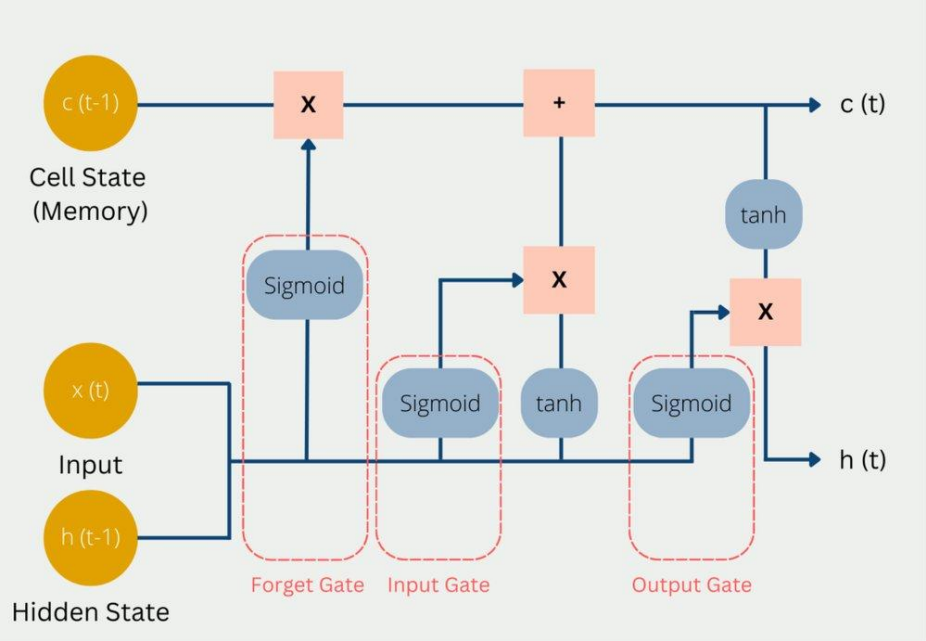


Figure 3.1: The LSTM model is a feedback neural network architecture

The highlight of LSTM is its ability to learn and retain long-term dependencies in sequential data. This is particularly useful in applications requiring extended context understanding such as Natural Language Processing (NLP), machine translation, speech recognition, or time series forecasting.

By solving the vanishing gradient problem—a major barrier in training deep recurrent networks—LSTM not only ensures a stable learning process but also opens up possibilities for wide application in many fields requiring time-based data processing.

a. Basic Structure in Long Short-Term Memory

The basic structure of LSTM revolves around three main gates: the Forget Gate, Input Gate, and Output Gate, along with a special memory to help the model process long data sequences efficiently. Specifically:

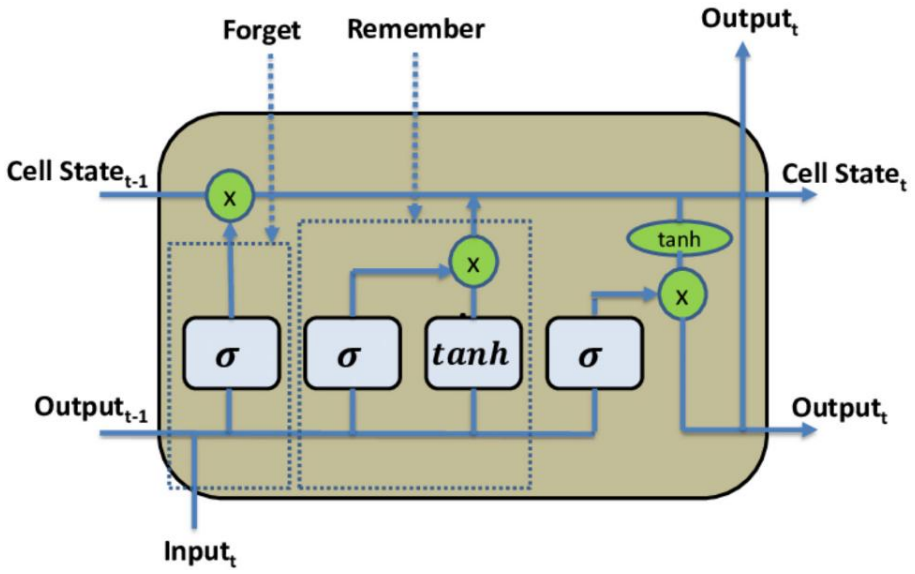


Figure 3.2: Basic structure in LSTM revolving around 3 main gates

The Forget Gate decides which information from the previous state needs to be retained or discarded. It uses a sigmoid activation function to output a value between 0 and 1, where 0 means completely discarding old information and 1 means keeping it intact. Thanks to this mechanism, LSTM can filter out unnecessary information, optimizing the ability to remember important data.

The Input Gate determines which new input information needs to be added to the memory. This mechanism involves two main steps: a sigmoid function (deciding which parts of the information will be updated) and a tanh function (creating a candidate vector containing new data), then combining with the output of the sigmoid function to adjust the memory. Thanks to this, LSTM can integrate new information selectively, helping the model adapt to data over time.

The Output Gate controls which information will be passed to the next step. The operating mechanism includes: a sigmoid function (filtering important information from memory) and a tanh function (adjusting the memory state before outputting the result). The Output Gate helps LSTM focus on important features and limit noise, increasing model accuracy in sequence processing tasks.

b. Pros and Cons of LSTM

LSTM is a powerful model with long-term memory capabilities and complex sequence processing. However, implementation requires balancing advantages and accompanying technical challenges.

Pros

LSTM is considered one of the advanced neural network architectures, bringing significant advantages such as:

- Good handling of long sequences: The standout strength of LSTM is the ability to remember information from previous time steps, even when relevant information is far apart. This is particularly useful in problems like Natural Language Processing (NLP), machine translation, sentiment analysis, or time series forecasting.

- Solves the vanishing gradient problem: A common issue in training RNNs is the vanishing gradient phenomenon during backpropagation. However, thanks to the special structure including control gates (like the forget gate), LSTM helps maintain a stable gradient flow, thereby improving training efficiency in deep networks.

- Flexibly handling variable-length sequences: Unlike traditional models requiring fixed inputs, LSTM can flexibly adapt to data sequences of varying lengths. This opens up possibilities for wide application in practical problems.

- Efficient internal memory: The characteristic "memory" component of LSTM helps retain important information for a long time while eliminating unnecessary elements. This helps increase the model's "context understanding" capability in many complex tasks.

- Intelligent gradient flow control: The gate mechanism in LSTM allows for effective control of gradient propagation, minimizing the risk of information loss during prolonged training processes.

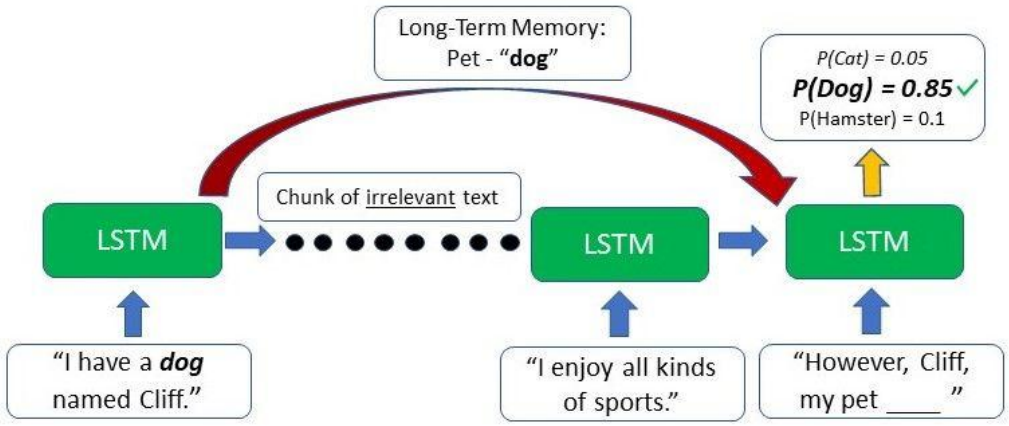


Figure 3.3: LSTM flexible intelligent gradient flow control

Cons

Despite bringing many superior advantages, LSTM models still have some limitations:

- High computational costs: Compared to simpler architectures like Feedforward networks or basic RNNs, LSTM has a more complex structure, requiring larger computational resources, resulting in high costs.

- Prone to overfitting if training data is insufficient: Like many other deep learning models, LSTM can learn too closely to the training data, leading to poor efficiency when working on real-world data. Therefore, it is necessary to apply anti-overfitting techniques such as dropout, regularization, or data augmentation to overcome this.

- Difficult hyperparameter optimization: Selecting hyperparameters such as the number of LSTM units, learning rate, or input sequence length significantly affects model performance. However, this fine-tuning process often takes a lot of time and requires personnel with in-depth experience.

- Low interpretability: Like many deep learning models, LSTM operates like a "black box," and it is difficult to determine the reason why a certain prediction is made.

This causes difficulties in application fields requiring transparency or detailed explanation like finance or healthcare.

- Long training time: Due to the multi-layer structure and complex mechanisms, LSTM often consumes a significant amount of time to train, especially when applied to large datasets. To optimize efficiency, using GPUs or TPUs is almost mandatory in these cases.

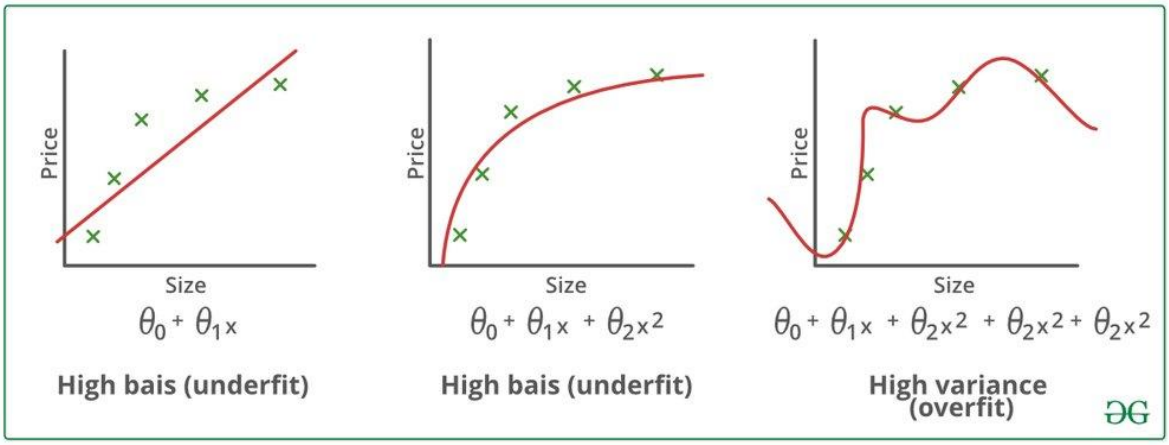


Figure 3.4: If the training data is insufficient, the model is prone to overfitting

3.1.3 Weather State Classification Problem

Besides the time series forecasting problem, the thesis also poses the problem of classifying weather states (sunny, rainy, high humidity, strong wind, etc.) based on meteorological features. For this problem, traditional machine learning models prove effective due to their ability to process tabular data and nonlinear relationships.

In the system, daily data is processed using the VotingClassifier model, combining multiple sub-models to enhance stability and accuracy. Specifically, the VotingClassifier is built from four models: XGBoost (XGBClassifier), LightGBM (LGBMClassifier), Extra Trees, and Random Forest. This ensemble approach leverages the strengths of each model while reducing overfitting and dependence on a single model.

For hourly data, the HistGradientBoostingClassifier model is used due to its fast training capability, good handling of large data, and suitability for rapidly changing features over time. This model is particularly effective when the sample size is large and features have complex distributions.

The combination of multiple machine learning models for different data types helps the forecasting system be more flexible and accurate, creating a reliable foundation for the agricultural recommendation module and intelligent chatbot.

3.1.4 Machine Learning and Ensemble Models in Weather Classification

a. Hourly

For hourly weather data, characterized by high frequency, large sample size, and rapid fluctuation over time, the thesis selects HistGradientBoostingClassifier for the weather state classification problem.

HistGradientBoostingClassifier is a machine learning model belonging to the Gradient Boosting family, designed to improve training speed and scalability when working with large datasets. Instead of processing continuous values directly, the model

uses histogram-based learning techniques, helping reduce computational costs during decision tree construction

Model Structure

Structurally, HistGradientBoostingClassifier relies on sequential boosting, where multiple weak learners (usually shallow decision trees) are trained sequentially to continuously correct the errors of previous models.

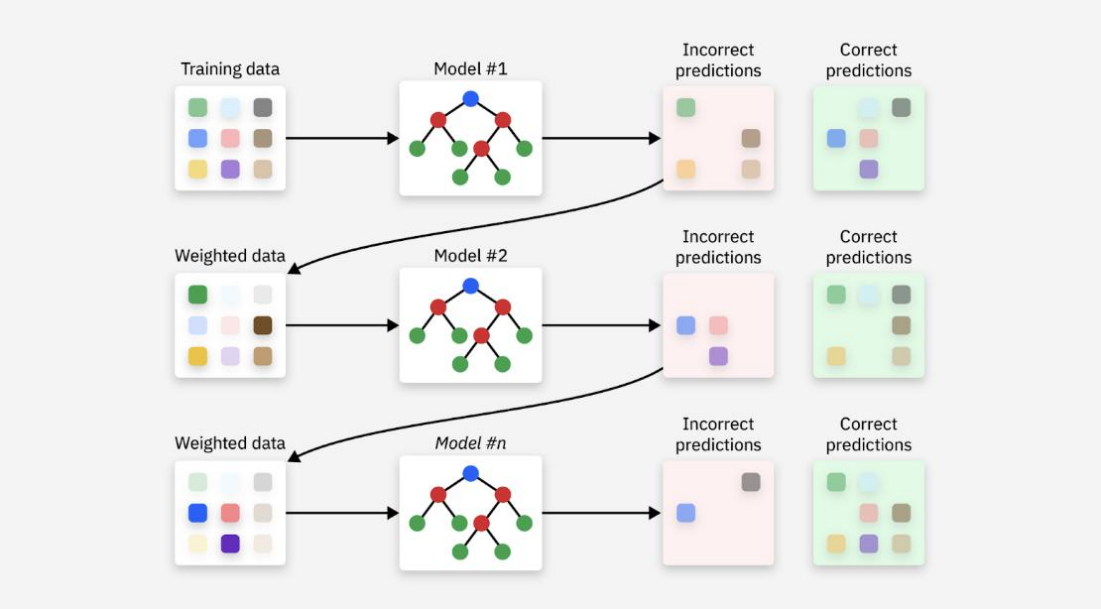


Figure 3.5: Simple structure of the Gradient Boosting model

An important improvement lies in input data processing: continuous features are discretized into histograms (bins) with a fixed number of bins. Finding split points in decision trees is performed on bins rather than original values, thereby:

- Reducing operations.
- Memory costs.
- Increasing training speed while maintaining the Gradient Boosting mechanism.

With large-scale, rapidly changing, and complex nonlinear hourly weather data, this model leverages continuous error correction from boosting while ensuring performance via histogram-based learning.

b. Daily

For daily weather data, which has fewer samples but is more aggregated and stable, the thesis uses VotingClassifier to improve classification accuracy and stability. The model combines different classifiers to reduce overfitting and applies soft voting to exploit probability prediction information from sub-models.

Model Structure

VotingClassifier is an ensemble learning method where multiple independent machine learning models are trained in parallel and their results are combined to produce the final prediction. Unlike boosting, sub-models do not depend on each other during training, increasing diversity and stability.

Four tree-based models with different mechanisms are used to compensate for each other's weaknesses:

- XGBoost: Strong learning with nonlinear relationships using gradient boosting and regularization.

- LightGBM: Similar structure to XGBoost but optimized for speed and memory using leaf-wise growth.

- Random Forest: Bagging method training independent trees, stable and less sensitive to noise.

- Extra Trees: Variant of Random Forest with higher randomness in node splitting, reducing variance.

Combining two boosting models (XGBoost, LightGBM) and two bagging models (Random Forest, Extra Trees) allows the VotingClassifier to leverage the strengths of both approaches, thereby enhancing the accuracy and reliability of daily weather classification results.

Soft Voting Mechanism (Soft Voting)

With soft voting, each sub-model provides a prediction probability for each class; the final result is determined by averaging (weighted or unweighted) these probabilities and selecting the class with the highest probability. This approach yields smoother and more stable prediction results compared to hard voting, especially when data is noisy or class boundaries are unclear.

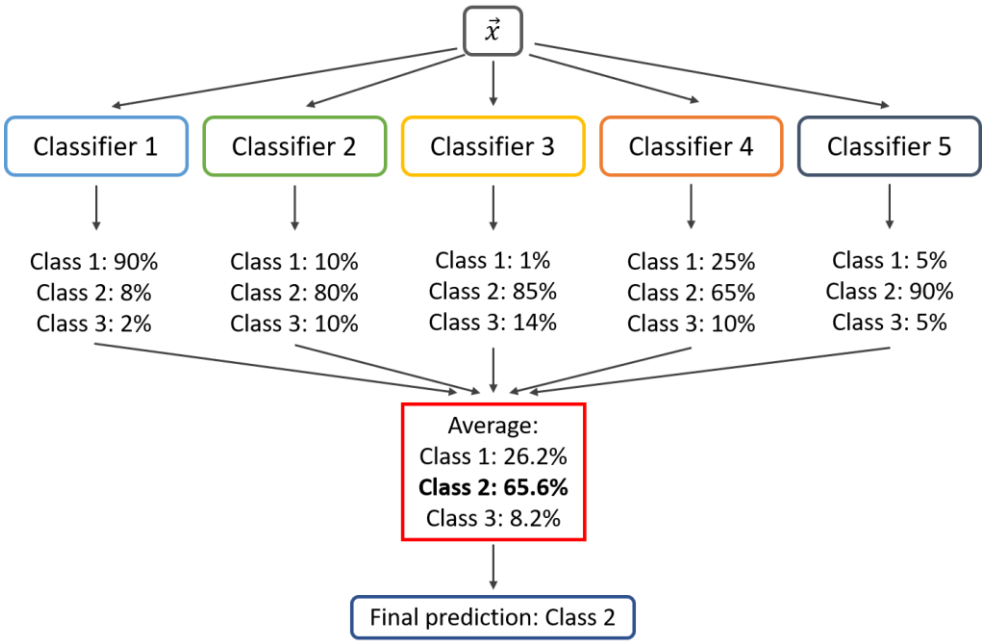


Figure 3.6: VotingClassifier with Soft voting

Thanks to combining multiple powerful and diverse models, VotingClassifier is suitable for daily weather data—where each data sample is more aggregated and stable compared to hourly data.

3.2 Technologies Used

3.2.1 Data Processing and Analysis Technologies

Python is used as the primary language for data processing and analysis in the thesis due to its rich ecosystem of libraries, which is particularly suitable for machine learning and deep learning problems.

NumPy and Pandas are used to process tabular data, including cleaning data, handling missing values, normalizing, and transforming meteorological features. This serves as the foundation for the entire input data preprocessing workflow.

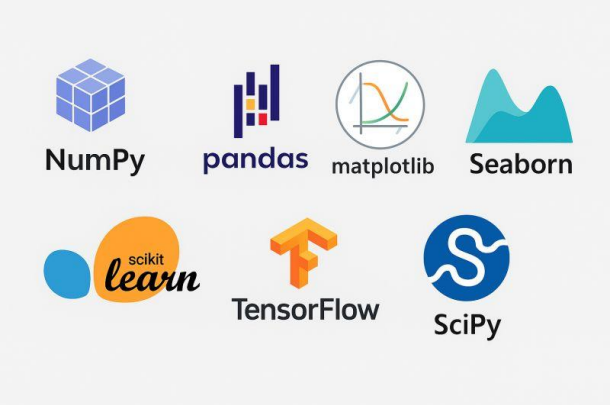


Figure 3.7: Data Processing, Visualization and Analysis Library

Matplotlib and Seaborn support data visualization, helping to analyze feature distribution, trends over time, and relationships between variables, thereby serving the feature selection process.

In the advanced preprocessing stage, Feature-engine and Scikit-learn (sklearn) are used to transform features, normalize data, split training-testing sets, and support the evaluation of the influence of input variables. XGBoost is further exploited to determine feature importance.

3.2.2 Backend Development Technologies

The system's backend is built using Python combined with the FastAPI framework to provide data processing services, model training-inference, and communication with the frontend. FastAPI was selected due to its high performance, clear syntax, and strong support for modern API applications.

Figure 3.8: FastAPI

FastAPI operates based on the RESTful architecture, allowing the construction of APIs that handle weather prediction, query results, and provide data for the recommendation system. This framework supports asynchronous operations (async/await), helping the system efficiently handle concurrent requests, which is especially suitable for machine learning model inference tasks.

Additionally, FastAPI integrates Swagger UI and OpenAPI, supporting automated API testing and documentation, making the development and system expansion process more convenient. The backend plays a central role in connecting input data, prediction models, and the frontend components of the system.

3.2.3 Frontend Development Technologies

The system's frontend is built using React, a popular JavaScript library for user interface development. React allows for component-based interface construction, enhancing reusability, ease of maintenance, and system scalability.

Figure 3.9: React JS

TypeScript (TSX) is used in conjunction with React to add static type checking, helping reduce errors during development and improving source code safety. The use of TypeScript is particularly useful in applications with many components and complex data flows.

To design the interface, the system uses Tailwind CSS, a utility-class-based CSS framework. This approach helps build interfaces quickly, consistently, and allows for direct customization within JSX code without writing many separate CSS files.

Figure 3.10: Tailwind CSS

Additionally, the lucide-react library is used to provide lightweight, modern interface icons that are easy to integrate with React. Library management and the frontend project build process are performed via npm, ensuring a stable development environment convenient for deployment.

3.2.4 Database Management System

The system uses Supabase as the primary database management system. Supabase is a key-value database that operates on an embedded model, allowing data to be stored directly in files without deploying a separate database server.

Supabase was selected due to its simple structure, high read performance, and suitability for small to medium-scale systems. This database meets the needs for storing weather data, prediction results, and information serving the recommendation system well.

Furthermore, Supabase supports transaction mechanisms and ensures data consistency, helping the backend access and update data securely. The use of Supabase contributes to reducing deployment complexity while aligning with the goal of building a lightweight and easily scalable system within the scope of the thesis.

3.2.5 Chatbot and Recommendation System Technologies

The advantage of using LLMs in the chatbot and recommendation system is the ability to synthesize and interpret information, rather than just answering in list form or copying data. LLMs allow the system to explain weather forecast results, offer appropriate farming recommendations, and support multi-turn conversations with users. Llama-3.3-70B provides detailed answer quality and good control via prompts.

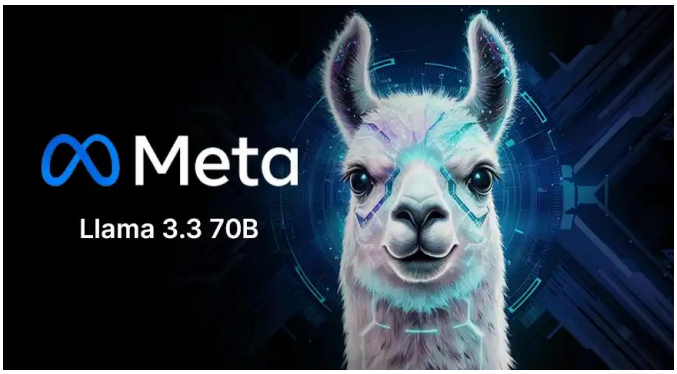


Figure 3.11: Llama-3.3-70B-Versatile

However, the use of LLMs also presents some limitations, such as consuming computational resources and the risk of generating inaccurate information if input or context control mechanisms are not strict. Therefore, in the system, the LLM is used as the final language generation layer, based on data already processed and forecasted by machine learning models, to ensure the consistency and reliability of the results.

CHAPTER 4 – TRAINING MULTIVARIATE WEATHER FORECASTING MODELS

4.1 General Introduction to the Dataset

The dataset used in the thesis is weather forecast data collected from the Open-Meteo platform. The data contains detailed information about weather conditions over time, organized by location and by day or hour, suitable for serving analysis, forecasting, and weather classification problems.

The daily dataset includes approximately 21 features describing meteorological and environmental factors. The group collected weather data focused on Vietnam covering 63 provinces and cities (before administrative mergers), spanning from January 1, 2020, to December 31, 2024 (5 years).

Below are the main components of the dataset and their definitions:

NO	Column Name	Definition	Unit
1	date	Time	-
2	temperature_2m_mean	Mean temperature	°C
3	temperature_2m_max	Maximum temperature	°C
4	temperature_2m_min	Minimum temperature	°C
5	apparent_temperature_mean	Mean apparent temperature	°C
6	apparent_temperature_max	Maximum apparent temperature	°C
7	apparent_temperature_min	Minimum apparent temperature	°C
8	dew_point_2m_mean	Dew point temperature	°C
9	precipitation_sum	Total precipitation (rain, showers, snow)	mm
10	rain_sum	Total rain	mm
11	snowfall_sum	Total snowfall	cm
12	cloud_cover_mean	Mean cloud cover	%
13	relative_humidity_2m_mean	Mean relative humidity	%
14	wind_gusts_10m_mean	Mean wind gusts	km/h
15	wind_speed_10m_mean	Mean wind speed	km/h
16	winddirection_10m_dominant	Dominant wind direction	°
17	surface_pressure_mean	Mean surface pressuret	hPa
18	pressure_msl_mean	Mean sea level pressure	hPa
19	daylight_duration	Daylight duration	seconds
20	sunshine_duration	Sunshine duration	seconds
21	weather_code	Weather condition code	WMO code

Table 4.1: Definition of Daily Dataset

The daily data CSV files are read sequentially from Google Drive, where the system extracts geographical coordinate information (latitude, longitude) from the metadata of each file and assigns a provider_id (self-defined) to identify the data source.

The data is then cleaned by removing metadata lines, normalizing headers, and adding spatial attributes before being overwritten to the CSV file, ensuring consistency for subsequent analysis and model building steps.

The system uses two separate daily datasets for different forecasting objectives. The weather type forecast dataset is attached with a province_id for identification by administrative unit, having a size of (115,101 records, 22 attributes). Meanwhile, the next 7-day forecast dataset does not use province_id but replaces it with latitude and longitude coordinates, having a size of (115,101 records, 23 attributes), to support spatial location-based forecasting.

The hourly dataset includes approximately 16 features describing meteorological and environmental factors. The group collected weather data focused on Vietnam covering 63 provinces and cities (before administrative mergers), spanning from January 1, 2020, to December 31, 2024 (5 years).

Below are the main components of the dataset and their definitions:

NO	Column Name	Definition	Unit
1	date	Time	-
2	temperature_2m	Temperature	°C
3	apparent_temperature	Apparent temperature	°C
4	dew_point_2m	Dew point temperature	°C
5	precipitation	Precipitation (rain, showers, snow)	mm
6	rain	Rain	mm
7	snowfall	Snowfall	cm
8	snowdepth	Snow depth	m
9	cloud_cover	Cloud cover	%
10	relative_humidity_2m	Relative humidity	%
11	wind_gusts_10m	Wind gusts	km/h
12	wind_speed_10m	Wind speed	km/h
13	wind_direction_10m	Wind direction	°
14	surface_pressure	Surface pressure	hPa
15	pressure_msl	Pressure reduced to mean sea level	hPa
16	weather_code	Weather condition code	WMO code

Table 4.2: Definition of Hourly Dataset

The CSV data files for each province are read sequentially from the storage directory. For each file, the system removes the initial metadata lines, uses the first data row as the column header, while normalizing column names and eliminating redundant header rows. After the preprocessing step, all data tables are concatenated vertically (row-wise concatenation) using the Pandas library to form a unified dataset. Additionally, the system extracts geographical coordinate information (latitude, longitude) from each file's metadata.

The system utilizes two separate hourly datasets for different forecasting objectives. The weather type forecast dataset remains unchanged with a size of (2,762,424 records, 16 attributes). Meanwhile, the next 24-hour forecast dataset is

supplemented with latitude and longitude coordinates, resulting in a size of (2,762,424 records, 18 attributes), to support spatial location-based forecasting.

4.2 Implementation of Daily Weather Condition Prediction Model

4.2.1 Dataset Overview

In this section, we will load the dataset and perform a preliminary examination of its structure and content.

```
Mounted at /content/drive
```

	date	temperature_2m_mean (°C)	temperature_2m_max (°C)	temperature_2m_min (°C)	apparent_temperature_mean (°C)	apparent_temperature_max (°C)	apparent_temperature_min (°C)	dew_point_2m_mean (°C)	precip
0	2020-01-01	19.9	23.0	18.3	21.6	23.9	20.1	17.6	
1	2020-01-02	21.1	25.4	18.5	23.0	27.0	20.6	18.3	
2	2020-01-03	21.5	25.4	19.8	23.6	27.4	21.7	19.2	
3	2020-01-04	21.1	23.5	19.4	22.8	25.3	21.2	19.2	
4	2020-01-05	21.5	25.6	19.0	23.0	26.5	20.5	19.2	

5 rows × 22 columns

Figure 4.1: First few rows of the dataset

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 115101 entries, 0 to 115100
Data columns (total 22 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   date                            115101 non-null  object
 1   temperature_2m_mean (°C)        115101 non-null  float64
 2   temperature_2m_max (°C)         115101 non-null  float64
 3   temperature_2m_min (°C)         115101 non-null  float64
 4   apparent_temperature_mean (°C)  115101 non-null  float64
 5   apparent_temperature_max (°C)   115101 non-null  float64
 6   apparent_temperature_min (°C)   115101 non-null  float64
 7   dew_point_2m_mean (°C)          115101 non-null  float64
 8   precipitation_sum (mm)          115101 non-null  float64
 9   rain_sum (mm)                   115101 non-null  float64
 10  snowfall_sum (cm)               115101 non-null  float64
 11  cloud_cover_mean (%)            115101 non-null  int64
 12  relative_humidity_2m_mean (%)   115101 non-null  int64
 13  wind_gusts_10m_mean (km/h)      115101 non-null  float64
 14  wind_speed_10m_mean (km/h)      115101 non-null  float64
 15  winddirection_10m_dominant (°)  115101 non-null  int64
 16  surface_pressure_mean (hPa)     115101 non-null  float64
 17  pressure_msl_mean (hPa)         115101 non-null  float64
 18  daylight_duration (s)           115101 non-null  float64
 19  sunshine_duration (s)           115101 non-null  float64
 20  weather_code (wmo code)         115101 non-null  int64
 21  province_id                     115101 non-null  int64
dtypes: float64(16), int64(5), object(1)
memory usage: 19.3+ MB
```

Figure 4.2: Summary information about the data

	temperature_2m_mean (°C)	temperature_2m_max (°C)	temperature_2m_min (°C)	apparent_temperature_mean (°C)	apparent_temperature_max (°C)	apparent_temperature_min (°C)	dew_point_2m_mean (°C)
count	115101.000000	115101.000000	115101.000000	115101.000000	115101.000000	115101.000000	115101.000000
mean	25.074765	29.123903	21.985847	28.669666	33.355380	25.217428	21.152149
std	4.137683	4.545753	4.260064	6.170504	6.675523	6.395342	4.564801
min	5.100000	6.000000	3.000000	0.000000	1.100000	-2.500000	-6.300000
25%	23.000000	26.700000	19.900000	25.400000	29.500000	21.800000	19.100000
50%	26.200000	30.000000	23.300000	30.500000	34.900000	27.300000	22.700000
75%	27.700000	32.100000	24.900000	33.000000	38.200000	29.800000	24.400000
max	35.600000	42.900000	32.200000	41.000000	48.800000	37.100000	28.600000

8 rows × 21 columns

Figure 4.3: Summary statistics

After checking for empty data, no features were found to contain null values.

date	0
temperature_2m_mean (°C)	0
temperature_2m_max (°C)	0
temperature_2m_min (°C)	0
apparent_temperature_mean (°C)	0
apparent_temperature_max (°C)	0
apparent_temperature_min (°C)	0
dew_point_2m_mean (°C)	0
precipitation_sum (mm)	0
rain_sum (mm)	0
snowfall_sum (cm)	0
cloud_cover_mean (%)	0
relative_humidity_2m_mean (%)	0
wind_gusts_10m_mean (km/h)	0
wind_speed_10m_mean (km/h)	0
winddirection_10m_dominant (°)	0
surface_pressure_mean (hPa)	0
pressure_msl_mean (hPa)	0
daylight_duration (s)	0
sunshine_duration (s)	0
weather_code (wmo code)	0
province_id	0

dtype: int64

Figure 4.4: Checking for null values

Standardizing feature names for cleanliness

```
df.columns = (
    df.columns
      .str.replace(r'\s*\(.*?\)', '', regex=True)
      .str.replace('Â', '', regex=False)
      .str.strip()
)
```

Figure 4.5: Standardizing feature names

4.2.2 Exploratory Data Analysis (EDA)

Exploratory Data Analysis helps us understand the distribution of features, relationships between variables, and identify patterns or anomalies. In this section, we will visualize and analyze the dataset.

Distribution plots, boxplots, and probability plots are generated. For each column, skewness and kurtosis are calculated and printed.

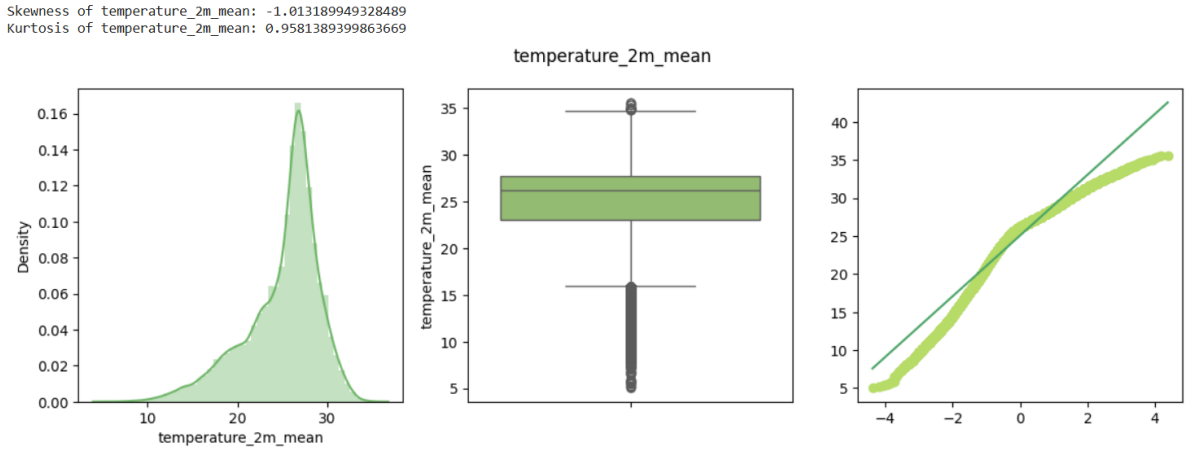


Figure 4.6: Distribution plots, boxplots, and probability plots

Bar chart showing the frequency of predictable weather types in the Weather_code feature. It has 10 distinct labels representing each weather condition.

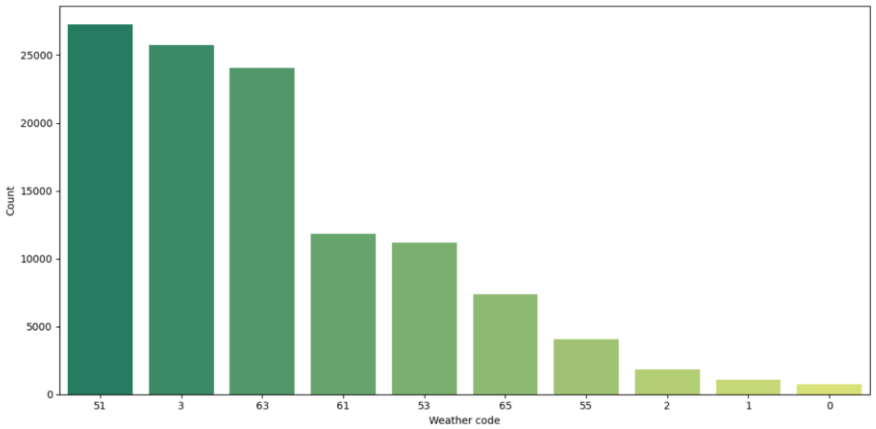


Figure 4.7: Bar chart of Weather Type Frequencies

weather_code	count
51	27247
3	25715
63	24067
61	11807
53	11169
65	7352
55	4080
2	1835
1	1066
0	763

dtype: int64

Figure 4.8: Frequency of occurrence of weather types

Heatmap for the correlation matrix.

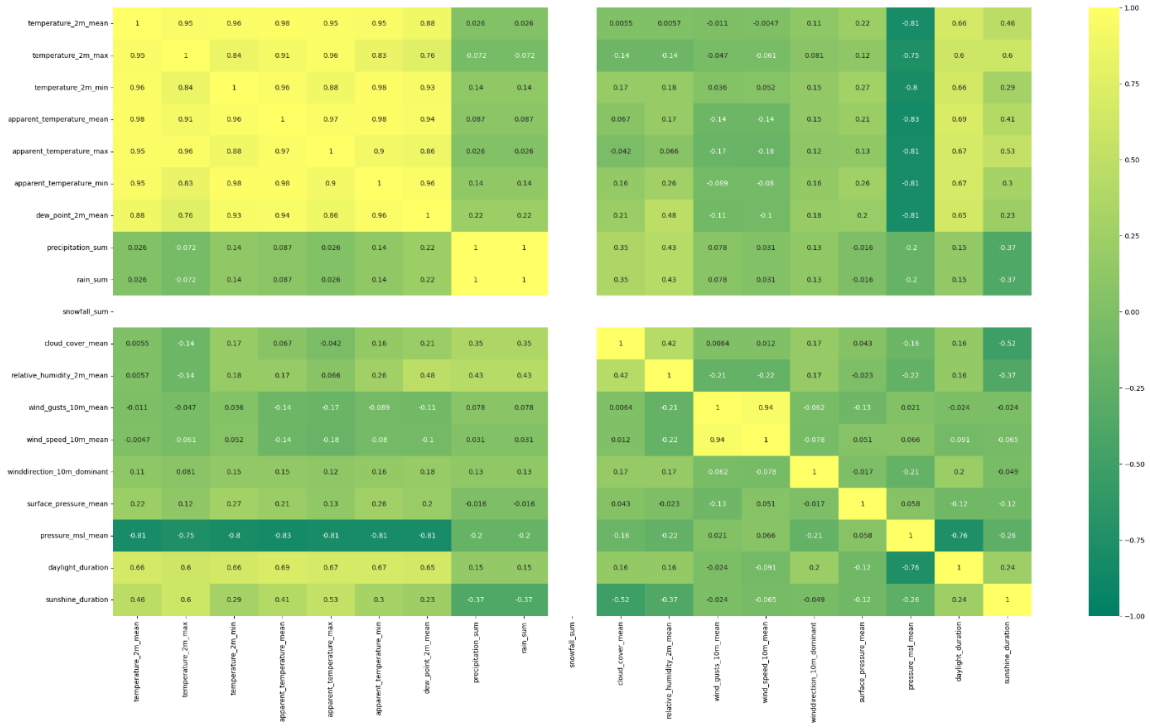


Figure 4.9: Heatmap chart

Dividing into 4 small groups to draw scatter plots between variables and the target variable Weather_code, and normalizing this variable to categorical form.

```
group1 = ['temperature_2m_mean', 'temperature_2m_max', 'temperature_2m_min', 'apparent_temperature_mean', 'apparent_temperature_max', 'apparent_temperature_min']
group2 = ['dew_point_2m_mean', 'precipitation_sum', 'rain_sum', 'snowfall_sum', 'cloud_cover_mean', 'relative_humidity_2m_mean']
group3 = ['wind_gusts_10m_mean', 'wind_speed_10m_mean', 'winddirection_10m_dominant', 'surface_pressure_mean', 'pressure_msl_mean', 'daylight_duration',
          'sunshine_duration']
```

```
codes = [0, 1, 2, 3, 51, 53, 55, 61, 63, 65]
df['weather_code'] = df['weather_code'].astype('category')
df['weather_code'] = df['weather_code'].cat.set_categories(codes)
```

Figure 4.10: Group of scatter plots

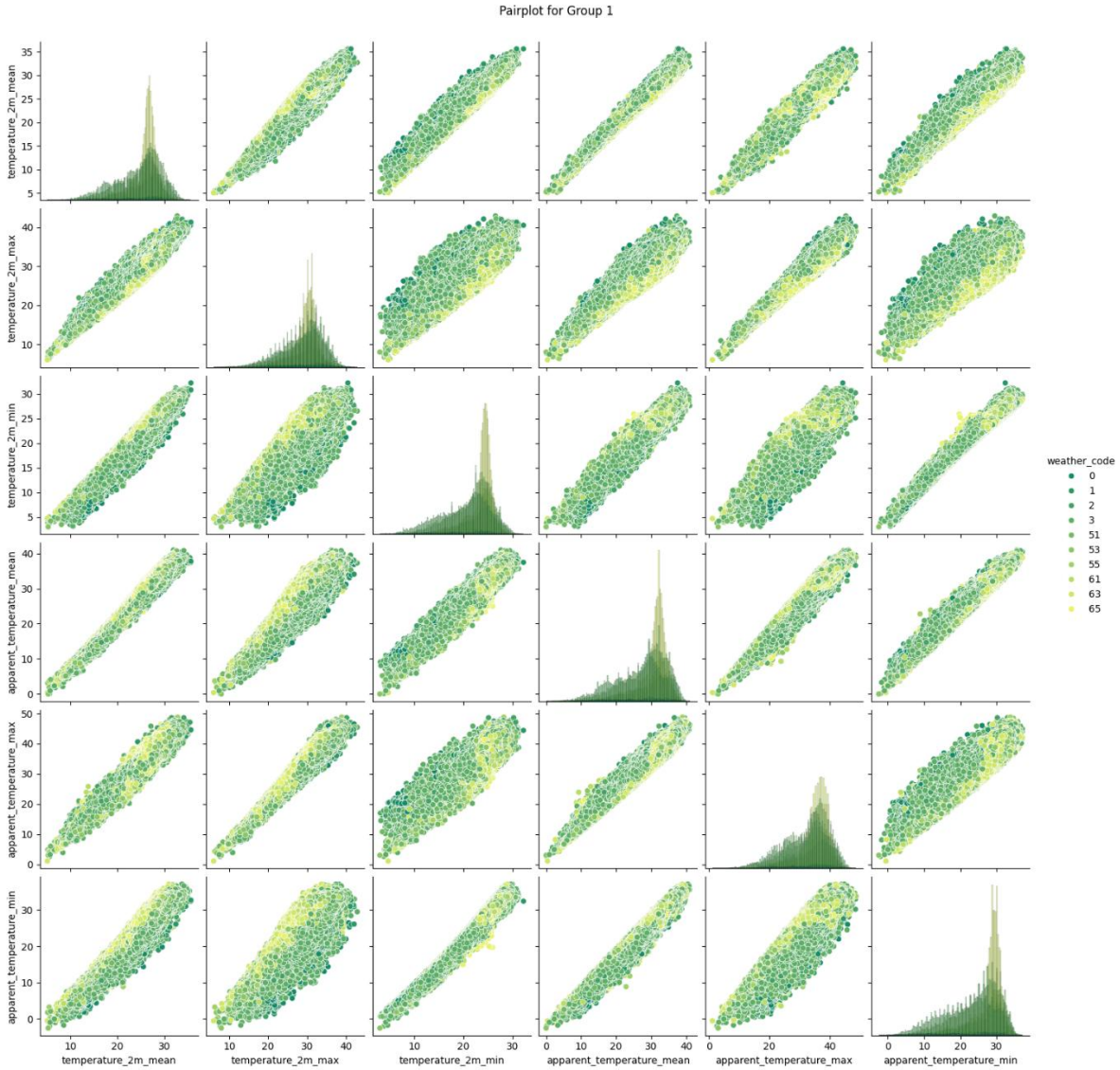


Figure 4.11: Scatter plot for Group 1

4.2.3 Feature Engineering

a. Feature Extraction

Extract time components from the date feature to create new columns including sin_doy, cos_doy and remove the date and province_id features.

```
df['date'] = pd.to_datetime(df['date'], errors='coerce')

print(df['date'].dtype)          # datetime64[ns]
print(df['date'].isna().sum())   # 0
```

```
datetime64[ns]
0
```

```
df['dayofyear'] = df['date'].dt.dayofyear
df['sin_doy'] = np.sin(2 * np.pi * df['dayofyear'] / 365)
df['cos_doy'] = np.cos(2 * np.pi * df['dayofyear'] / 365)

df = df.sort_values(['province_id', 'date']).reset_index(drop=True)
```

```
df = df.drop(columns=['date','dayofyear'])
```

Figure 4.12: Processing the date column

b. Feature Splitting

Select the target variable as Weather_code and the feature variables as the other features.

```
features = ['temperature_2m_mean', 'temperature_2m_max', 'temperature_2m_min',
            'apparent_temperature_mean', 'apparent_temperature_max', 'apparent_temperature_min',
            'dew_point_2m_mean', 'precipitation_sum', 'rain_sum', 'snowfall_sum', 'cloud_cover_mean', 'relative_humidity_2m_mean',
            'wind_gusts_10m_mean', 'wind_speed_10m_mean', 'winddirection_10m_dominant',
            'surface_pressure_mean', 'pressure_msl_mean', 'daylight_duration', 'sunshine_duration', 'sin_doy', 'cos_doy'
]

target_column = ['weather_code']
```

Figure 4.13: Creating feature variables and target variable

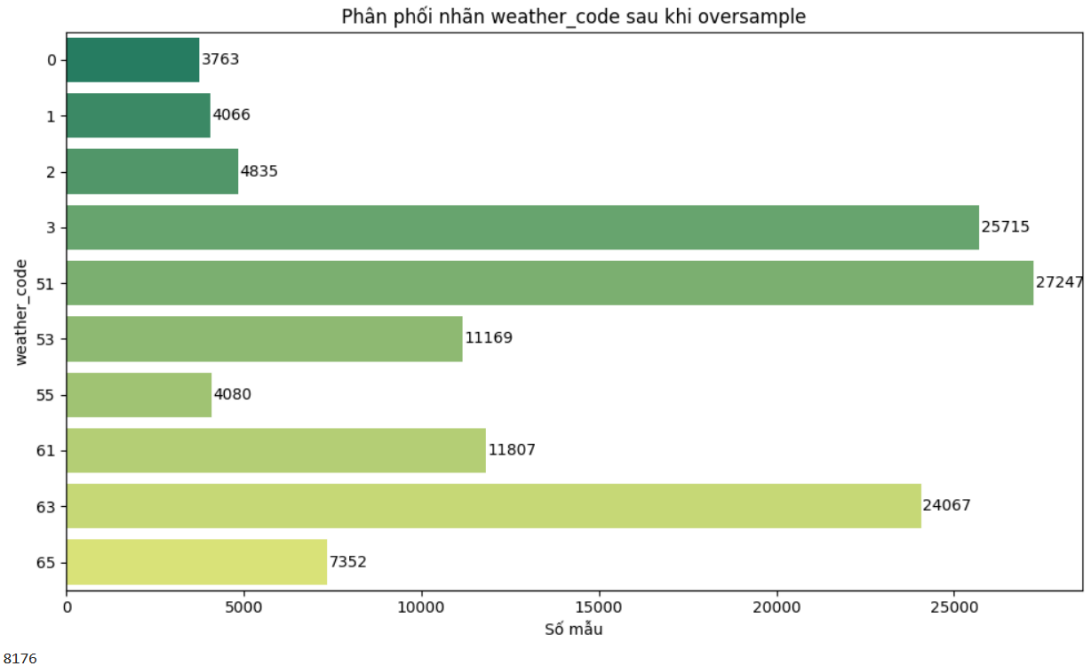
c. Balancing the target classes

Balance the data by augmenting samples for the 3 rare labels (having the fewest samples), which are "0", "1", "2", with a target of at least 3000 samples.

```
def oversample_to_min(X, y, label_value, min_size):
    mask = y['weather_code'] == label_value
    X_label = X[mask]
    y_label = y[mask]

    current_size = len(X_label)
    if current_size >= min_size:
        return X, y

    repeats = min_size // current_size + 1

    X_over = pd.concat([X_label] * repeats, ignore_index=True).iloc[:min_size]
    y_over = pd.concat([y_label] * repeats, ignore_index=True).iloc[:min_size]

    return (
        pd.concat([X, X_over], ignore_index=True),
        pd.concat([y, y_over], ignore_index=True)
    )
```

```
X, y = oversample_to_min(X, y, 0, 3000)
X, y = oversample_to_min(X, y, 1, 3000)
X, y = oversample_to_min(X, y, 2, 3000)
```

Figure 4.14: Data balancing



8176

Figure 4.15: Countplot of labels after balancing

d. Label Encoding

```
encoder = LabelEncoder()
y_encoded = encoder.fit_transform(y['weather_code'])
```

```
encoder.classes_
```

```
array([ 0,  1,  2,  3, 51, 53, 55, 61, 63, 65])
```

```
np.unique(y_encoded)
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Figure 4.16: Label encoding

e. Train Test Split

Split the data into training and testing sets.

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y_encoded,
    test_size=0.2,
    random_state=42,
    shuffle=True,
    stratify=y_encoded
)
```

```
print("Shape of the training set:",X_train.shape)
print("Shape of the testing set:",X_test.shape)
```

```
Shape of the training set: (99280, 21)
Shape of the testing set: (24821, 21)
```

Figure 4.17: Splitting data into training and testing sets

f. Feature Selection

Check which features are most important to select the optimal features for training. Here, after running the Pipeline, we remove 1 completely duplicate feature and 1 feature with meaningless values, which are snowfall_sum and rain_sum.

```
pipeline = Pipeline([
    ('constant', DropConstantFeatures()),
    ('duplicate', DropDuplicateFeatures())
    # ('correlated', DropCorrelatedFeatures())
])
```

```
X_train = pipeline.fit_transform(X_train)
X_test = pipeline.transform(X_test)
```

```
print("Shape of the training set:",X_train.shape)
print("Shape of the testing set:",X_test.shape)
```

```
Shape of the training set: (99280, 19)
Shape of the testing set: (24821, 19)
```

Figure 4.18: Applying pipeline with 2 data files

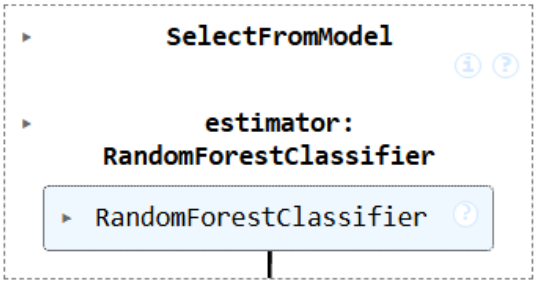
Feature selection for the classification problem by calculating the feature score of each feature for each label, then selecting the top 10 features with the highest average influence.

```
from sklearn.feature_selection import SelectKBest, f_classif

selector = SelectKBest(score_func=f_classif, k='all')
selector.fit(X_train, y_train)

scores = selector.scores_

score_df = pd.DataFrame({
    'feature': X_train.columns,
    'score': scores
}).sort_values('score', ascending=False)

top_10_features = score_df.head(10)['feature'].tolist()

print("Top 10 đặc trưng được chọn:", top_10_features)
```

Figure 4.19: Feature selection using feature score

Feature selection for the classification problem by using an automated feature selection method based on a machine learning model - specifically RandomForestClassifier - to select the maximum of 10 most important features.

```
sfm = SelectFromModel(estimator=RandomForestClassifier(),max_features=10)
sfm.fit(X_train,y_train)
```




```
selected_features = sfm.get_feature_names_out()
selected_features
```

```
array(['precipitation_sum', 'cloud_cover_mean'], dtype=object)
```

Figure 4.20: Automatic feature selection based on RandomForestClassifier

Feature selection for the multi-label classification problem by training the XGBClassifier model and visualizing feature importances.

```
def plot_feature_importances(feat_imp_type, figsize=(10, 8)):
    feat_imps = xgb.get_booster().get_score(importance_type=feat_imp_type)
    keys = list(feat_imps.keys())
    values = list(feat_imps.values())

    feat_imps_df = pd.DataFrame(data=values, index=keys, columns=["Importance"])\
        .sort_values(by="Importance", ascending=False).reset_index()
    feat_imps_df.rename({'index': 'Feature'}, axis=1, inplace=True)

    plt.figure(figsize=figsize)
    fig = sns.barplot(
        x='Importance',
        y='Feature',
        data=feat_imps_df,
        orient='h',
        palette='summer'
    )
    plt.title(f"{feat_imp_type.title()} Feature Importance", fontsize=16)
    plt.tight_layout()
    plt.show()
    plt.close('all')
    del fig
    gc.collect()
```

Figure 4.21: Function to draw countplot of features based on importance type

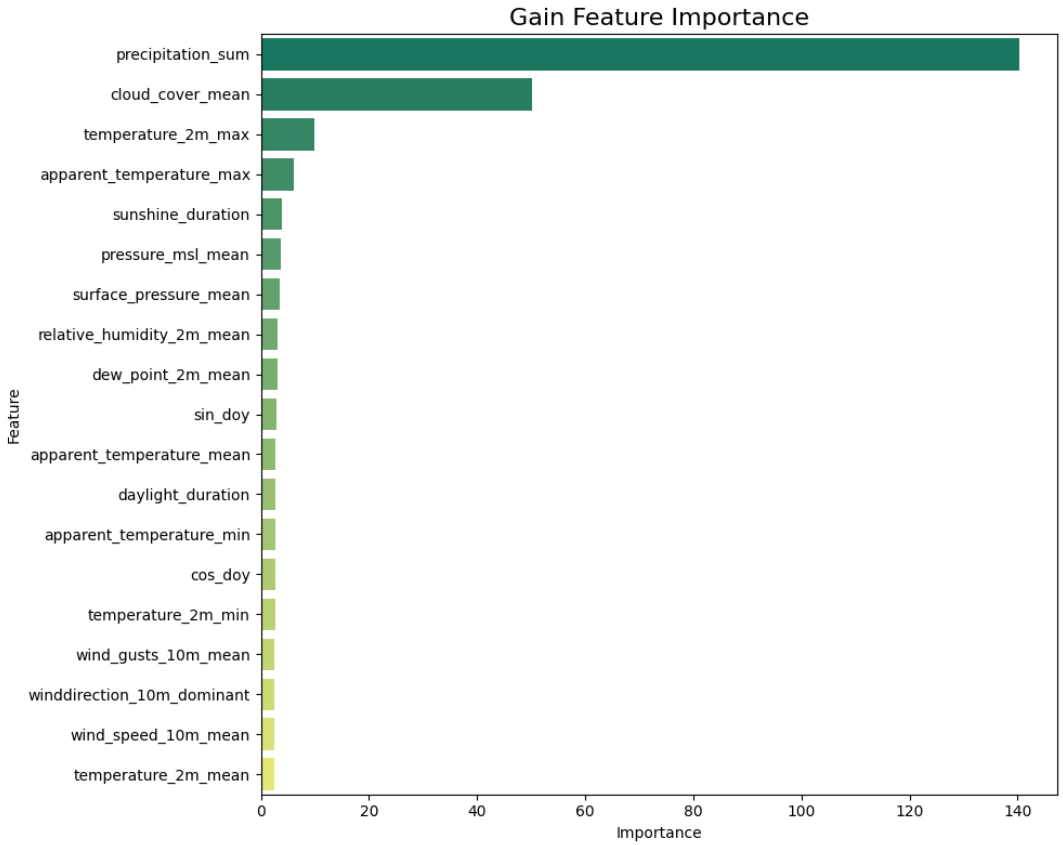


Figure 4.22: Function plot_feature_importances passing 'gain' as parameter

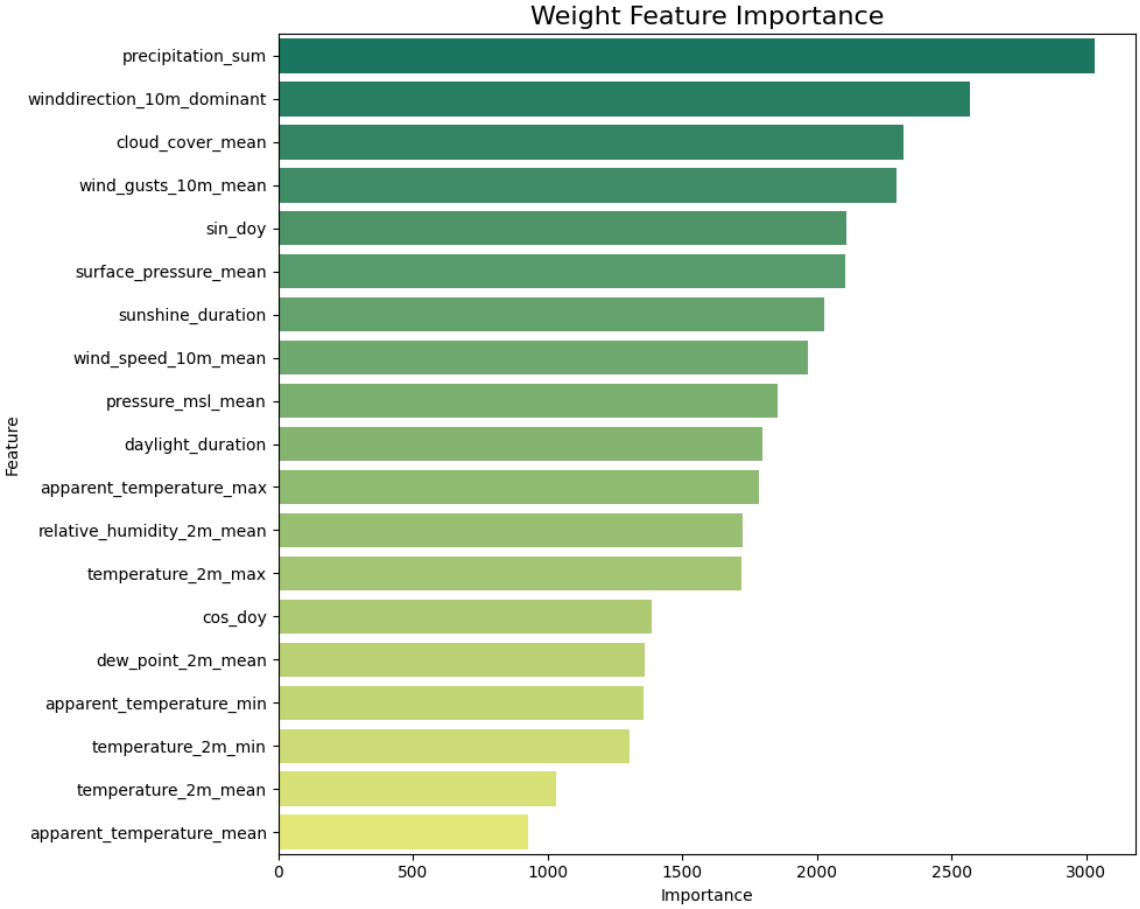


Figure 4.23: Function plot_feature_importances passing 'weight' as parameter

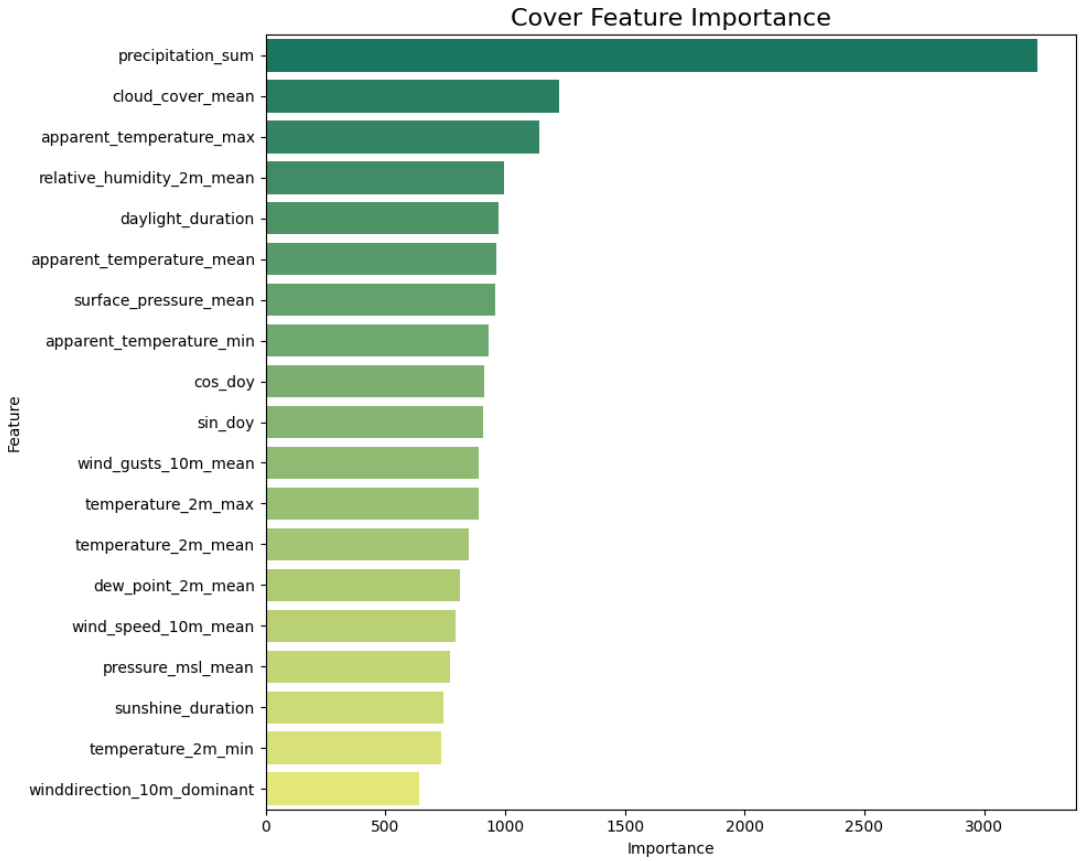


Figure 4.24: Function plot_feature_importances passing 'cover' as parameter

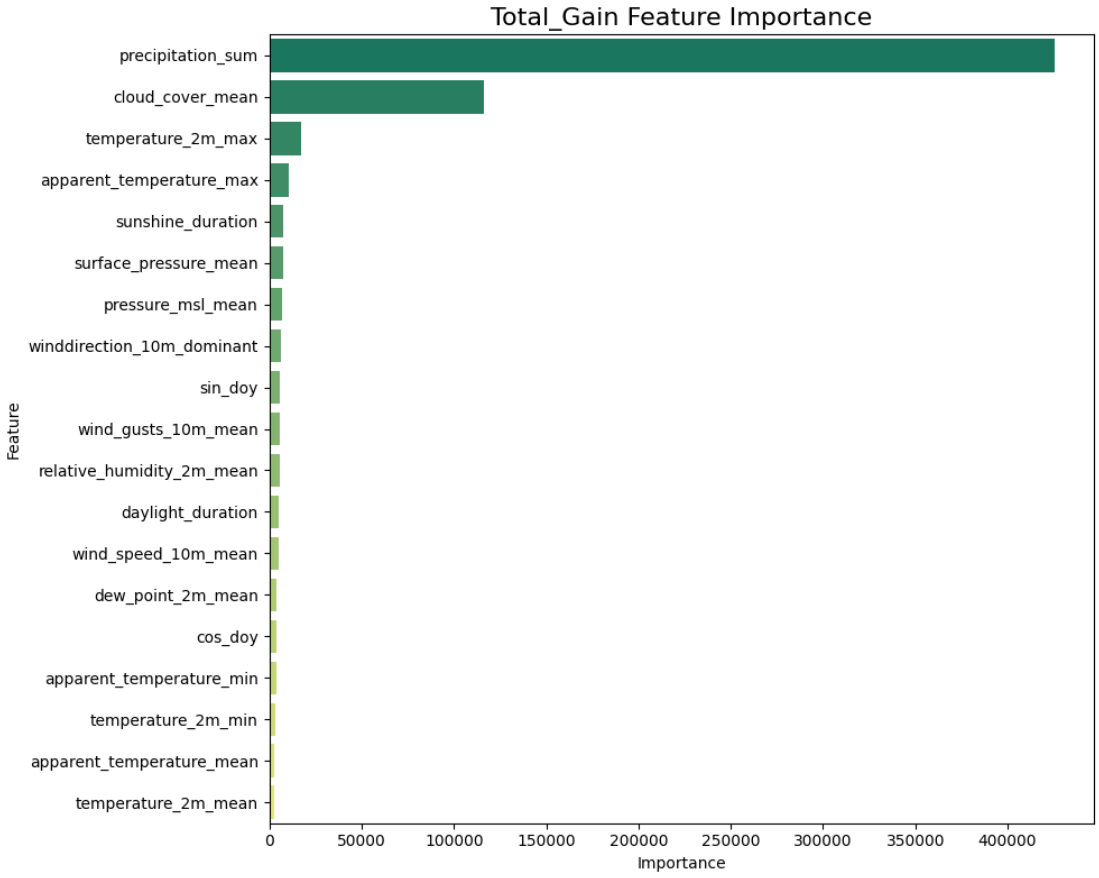


Figure 4.25: Function plot_feature_importances passing 'total_gain' as parameter

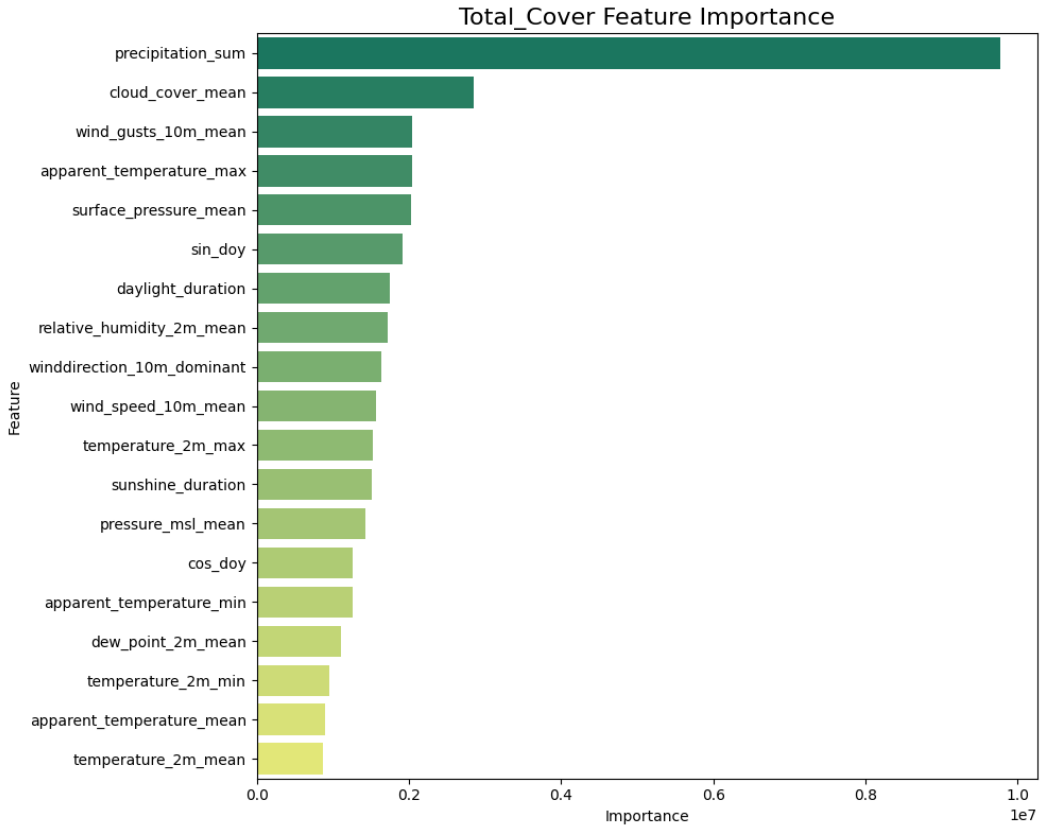


Figure 4.26: Function plot_feature_importances passing 'total_cover' as parameter

```
final_selected_features = ['temperature_2m_mean', 'temperature_2m_max', 'temperature_2m_min', 'apparent_temperature_mean', 'apparent_temperature_max', 'apparent_temperature_
                          'dew_point_2m_mean', 'precipitation_sum', 'cloud_cover_mean', 'relative_humidity_2m_mean',
                          'wind_gusts_10m_mean', 'wind_speed_10m_mean', 'winddirection_10m_dominant',
                          'surface_pressure_mean', 'pressure_msl_mean', 'daylight_duration', 'sunshine_duration', 'sin_doy', 'cos_doy']

final_X_train = X_train[final_selected_features]
final_X_test = X_test[final_selected_features]
```

Figure 4.27: Selecting final features for training

g. Feature Scaling

Normalize data to prepare for model training.

```
scaler = StandardScaler()
features = final_X_train.columns
final_X_train = scaler.fit_transform(final_X_train)
final_X_train = pd.DataFrame(final_X_train,columns=features)
final_X_test = scaler.transform(final_X_test)
final_X_test = pd.DataFrame(final_X_test,columns=features)

final_X_train.head()
```

	temperature_2m_mean	temperature_2m_max	temperature_2m_min	apparent_temperature_mean	apparent_temperature_max	apparent_temperature_min	dew_point_2m_mean
0	-0.858083	-1.097418	-0.540065	-0.940577	-1.300130	-0.581397	-0.850303
1	0.813463	0.935347	0.818830	0.523210	0.651345	0.509586	0.535980
2	-2.854651	-3.195058	-2.249644	-2.778096	-3.046186	-2.291587	-2.460837
3	-1.368833	-1.875923	-1.066090	-1.158587	-1.520221	-0.949972	-0.666824
4	-2.251037	-2.221926	-1.942796	-2.264213	-2.341894	-2.011469	-1.849241

Figure 4.28: Normalizing data in preparation for model training

4.2.4 Model Training & Evaluation

Create arrays to store values such as: model names input for training and metrics like Accuracy, Precision, Recall, F1. Write a train_and_evaluate_model function to train models and print the classification report of the corresponding model.

```
models = []
accuracy_scores = []
precision_scores = []
recall_scores = []
f1_scores = []
hamming_losses = []
```

```
def train_and_evaluate_model(model):
    model.fit(final_X_train, y_train)
    y_pred = model.predict(final_X_test)

    print("Classification Report:")
    print(classification_report(y_test, y_pred, zero_division=0))
    print('-'*50)
    ConfusionMatrixDisplay.from_predictions(y_test,y_pred)

    acc = accuracy_score(y_test, y_pred)
    hamming = hamming_loss(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='macro', zero_division=0)
    recall = recall_score(y_test, y_pred, average='macro', zero_division=0)
    f1 = f1_score(y_test, y_pred, average='macro', zero_division=0)

    accuracy_scores.append(acc)
    hamming_losses.append(hamming)
    precision_scores.append(precision)
    recall_scores.append(recall)
    f1_scores.append(f1)
    models.append(model)
    gc.collect()
```

Figure 4.29: Arrays storing values and Model training function

Train and evaluate the VotingClassifier model with 4 sub-models, including: XGBClassifier, LGBMClassifier, ExtraTreesClassifier, RandomForestClassifier.

```
voting_clf = VotingClassifier(
    estimators=[
        ('XGB', XGBClassifier()),
        ('LGBM', LGBMClassifier()),
        ('ET', ExtraTreesClassifier()),
        ('RF', RandomForestClassifier())
    ],
    voting='soft',
    weights=[3, 3, 2, 2],
    n_jobs=-1,
    verbose=1
)

train_and_evaluate_model(voting_clf)
```

Figure 4.30: Training parameters for VotingClassifier model

Results after training the VotingClassifier model.

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       753
           1       0.95      0.99      0.97       813
           2       0.87      0.95      0.91       967
           3       0.99      0.97      0.98      5143
           4       0.89      0.92      0.90      5450
           5       0.55      0.61      0.57      2234
           6       0.35      0.02      0.04       816
           7       0.51      0.51      0.51      2361
           8       0.73      0.84      0.78      4814
           9       0.74      0.55      0.63      1470

    accuracy                           0.80     24821
   macro avg       0.76      0.73      0.73     24821
weighted avg       0.79      0.80      0.79     24821

--------------------------------------------------
```

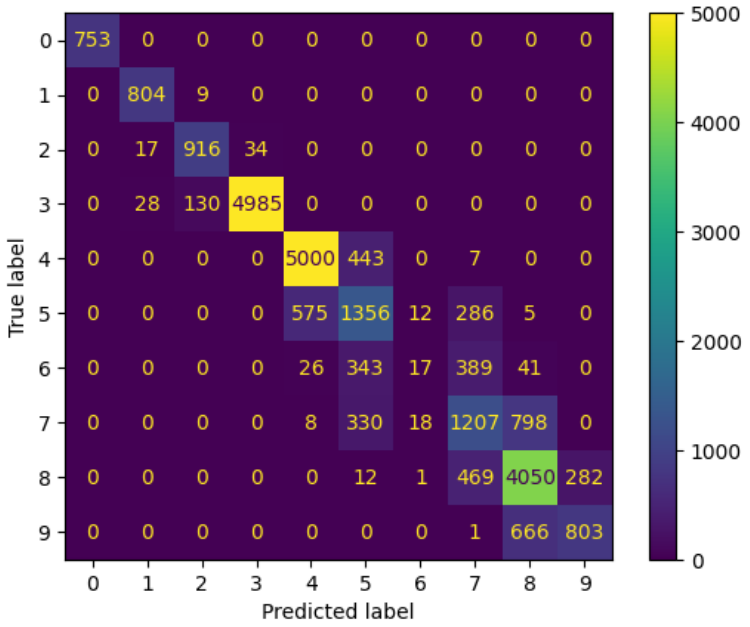


Figure 4.31: Results of VotingClassifier model

4.2.5 Comparing Performance of Base Models

```
model_perfs = pd.DataFrame({'Model': models,
                            'Accuracy': accuracy_scores,
                            'Precision': precision_scores,
                            'Recall': recall_scores,
                            'F1': f1_scores}).sort_values('Accuracy',ascending=False)
model_perfs
```

	Model	Accuracy	Precision	Recall	F1
10	StackingClassifier(cv=5,\n e...	0.806011	0.765008	0.741473	0.735579
9	VotingClassifier(estimators=[('XGB',\n ...	0.801378	0.758012	0.734949	0.729387
2	(DecisionTreeClassifier(max_features=4, random...	0.797268	0.748075	0.726932	0.724459
7	XGBClassifier(base_score=None, booster=None, c...	0.793038	0.742605	0.727093	0.723115
6	(ExtraTreeClassifier(max_features=4, random_st...	0.792917	0.748140	0.724778	0.726858
5	HistGradientBoostingClassifier()	0.789009	0.735970	0.718170	0.714414
8	<catboost.core.CatBoostClassifier object at 0x...	0.783329	0.715187	0.687684	0.684626
4	(DecisionTreeClassifier(random_state=195292617...	0.780790	0.730484	0.725956	0.723748
3	([DecisionTreeRegressor(criterion='friedman_ms...	0.759518	0.671926	0.644146	0.641423
1	DecisionTreeClassifier(random_state=42)	0.734741	0.686181	0.704886	0.694719
0	LogisticRegression(max_iter=1000)	0.704605	0.591086	0.587550	0.584013

Figure 4.32: Comparing performance of base models

Based on the results table, we can compare the performance of 11 models as follows:

Accuracy: Tells us the proportion of correct predictions by the model.

Precision: Tells us the proportion of predictions that are correct among the predictions the model made.

Recall: Tells us the proportion of actual cases that were predicted correctly.

F1 Score: Is a composite measure for both precision and recall.

Based on all four performance metrics, models 2, 9, 10 (RandomForestClassifier, VotingClassifier, StackingClassifier) have the best performance in predicting the target variable. Model 0 (LogisticRegression) has the lowest performance and needs further improvement.

4.2.6 Models Hyperparameter Tuning and Cross-Validation

Define the hyperparameter space to perform optimization for the Random Forest Classifier model in the label classification problem using RandomizedSearchCV. Specifically:

- Hyperparameter set (param_distributions) includes key factors such as:

+ n_estimators
+ criterion
+ max_features
+ bootstrap
+ oob_score
+ class_weight
+ max_depth

- Use RandomizedSearchCV to randomly try 5 hyperparameter combinations and select the best one through cross-validation (3 data splits).

```
param_distributions = {
    'n_estimators': [200, 300, 400],
    'criterion': ['gini'],
    'max_features': ['sqrt'],
    'bootstrap': [True],
    'oob_score': [False],
    'class_weight': ['balanced'],
    'max_depth': [10, 15, 20]
}

base_model = RandomForestClassifier(random_state=42,n_jobs=1)

grid_rf = RandomizedSearchCV(
    estimator=base_model,
    param_distributions=param_distributions,
    n_iter=5,
    cv=3,
    verbose=2,
    n_jobs=1,
    random_state=42
)

grid_rf.fit(final_X_train, y_train)
```

Figure 4.33: Defining hyperparameter space for RandomForestClassifier

```
grid_rf.best_score_
```

```
np.float64(0.7188759057597688)
```

```
grid_rf.best_params_
```

```
{'oob_score': False,
 'n_estimators': 400,
 'max_features': 'sqrt',
 'max_depth': 20,
 'criterion': 'gini',
 'class_weight': 'balanced',
 'bootstrap': True}
```

Figure 4.34: Result of best_score and best_params after training

Additionally, 3 other models were also hyperparameter tuned and cross-validated. However, in terms of accuracy and future usage, VotingClassifier has high accuracy, stability, and is the easiest to use.

4.2.7 Training and evaluating deep learning models

In the weather forecasting project, the use of deep learning models plays an important role in improving prediction accuracy. Deep Learning is a branch of machine learning, notable for its ability to learn and process complex features from large data. By applying deep neural networks, the model can exploit non-linear relationships between weather factors, thereby making more accurate predictions compared to traditional methods.

This project initializes and configures a Deep Neural Network model using the Keras library in Python. This model is designed to solve the classification problem, in this case, weather forecasting based on input parameters.

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	1,280
dense_1 (Dense)	(None, 128)	8,320
dropout (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 128)	16,512
dropout_1 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 10)	1,290

Total params: 27,402 (107.04 KB)
Trainable params: 27,402 (107.04 KB)
Non-trainable params: 0 (0.00 B)

Figure 4.35: Model summary

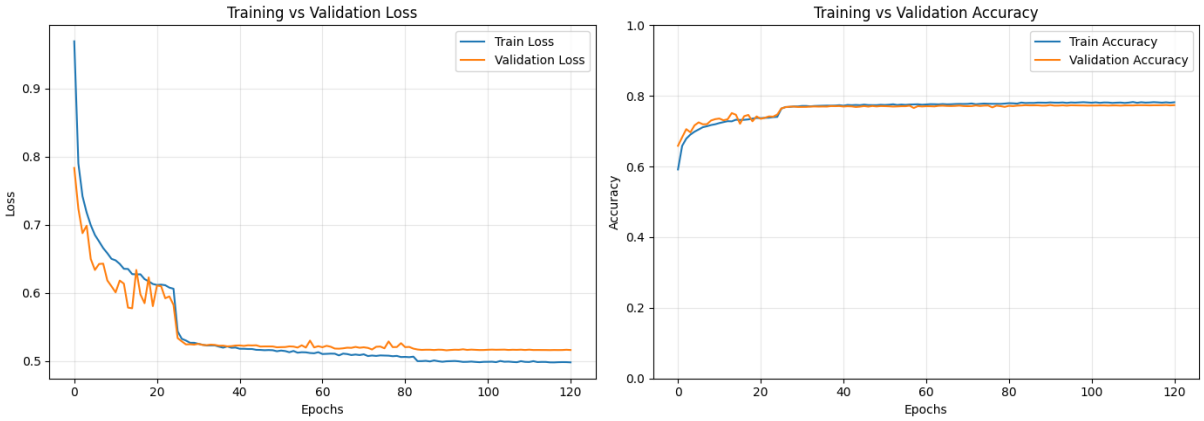


Figure 4.36: Loss chart and Accuracy chart

	STT	Model	Accuracy	Precision	Recall	F1
10	1	StackingClassifier(cv=5,\n e...	0.806011	0.765008	0.741473	0.735579
9	2	VotingClassifier(estimators=[('XGB',\n ...	0.801378	0.758012	0.734949	0.729387
2	3	(DecisionTreeClassifier(max_features=4, random...	0.797268	0.748075	0.726932	0.724459
7	4	XGBClassifier(base_score=None, booster=None, c...	0.793038	0.742605	0.727093	0.723115
6	5	(ExtraTreeClassifier(max_features=4, random_st...	0.792917	0.748140	0.724778	0.726858
5	6	HistGradientBoostingClassifier()	0.789009	0.735970	0.718170	0.714414
8	7	<catboost.core.CatBoostClassifier object at 0x...	0.783329	0.715187	0.687684	0.684626
4	8	(DecisionTreeClassifier(random_state=195292617...	0.780790	0.730484	0.725956	0.723748
11	9	Deep Learning	0.774143	0.661159	0.662696	0.657468
3	10	([DecisionTreeRegressor(criterion='friedman_ms...	0.759518	0.671926	0.644146	0.641423
1	11	DecisionTreeClassifier(random_state=42)	0.734741	0.686181	0.704886	0.694719
0	12	LogisticRegression(max_iter=1000)	0.704605	0.591086	0.587550	0.584013

Figure 4.37: Comparison after training the Deep Learning model

```
dump(scaler, '/content/drive/MyDrive/DATN/source/model_daily/scaler.joblib', compress=3)

dump(encoder, '/content/drive/MyDrive/DATN/source/model_daily/label_encoder.joblib', compress=3)
```

```
['/content/drive/MyDrive/DATN/source/model_daily/label_encoder.joblib']
```

```
dump(best_model, '/content/drive/MyDrive/DATN/source/model_daily/votingC.joblib', compress=0)
```

```
['/content/drive/MyDrive/DATN/source/model_daily/votingC.joblib']
```

Figure 4.38: Saving the best performing model for deployment

4.3 Implementation of Hourly Weather Condition Prediction Model

For hourly data, the preprocessing steps are almost identical to daily data. However, because the features are slightly different, there will be minor differences in a few places.

First, these are the original features when reading the data.

time
temperature_2m (°C)
apparent_temperature (°C)
dew_point_2m (°C)
precipitation (mm)
rain (mm)
snowfall (cm)
snow_depth (m)
cloud_cover (%)
relative_humidity_2m (%)
wind_gusts_10m (km/h)
wind_speed_10m (km/h)
wind_direction_10m (°)
surface_pressure (hPa)
pressure_msl (hPa)
weather_code (wmo code)

Figure 4.39: Initial features of hourly data

The first difference lies in processing the time column. Here, the data contains both date and hour, so we will process it to create 4 new features, including: sin_hour, cos_hour, sin_doy, cos_doy. Then, we remove the redundant columns: time, hour, dayofyear.

```
df['time'] = pd.to_datetime(df['time'], errors='coerce')

print(df['time'].dtype)          # datetime64[ns]
print(df['time'].isna().sum())   # 0
```

```
datetime64[ns]
0
```

```
df['hour'] = df['time'].dt.hour
df['dayofyear'] = df['time'].dt.dayofyear

df['sin_hour'] = np.sin(2 * np.pi * df['hour'] / 24)
df['cos_hour'] = np.cos(2 * np.pi * df['hour'] / 24)

df['sin_doy'] = np.sin(2 * np.pi * df['dayofyear'] / 365)
df['cos_doy'] = np.cos(2 * np.pi * df['dayofyear'] / 365)
```

```
df = df.drop(columns=['time', 'hour', 'dayofyear'])
```

Figure 4.40: Processing the time column

In the data balancing section, 4 rare labels were detected, including 65, 55, 63, and 61. Here, we will balance these rare labels with a set minimum number of samples, specific to each label. For label 65, the minimum is 50,000 samples; for label 55, it is 80,000 samples; for label 63, it is 100,000 samples; and for label 61, it is 110,000 samples. Below is the distribution chart of labels after balancing.

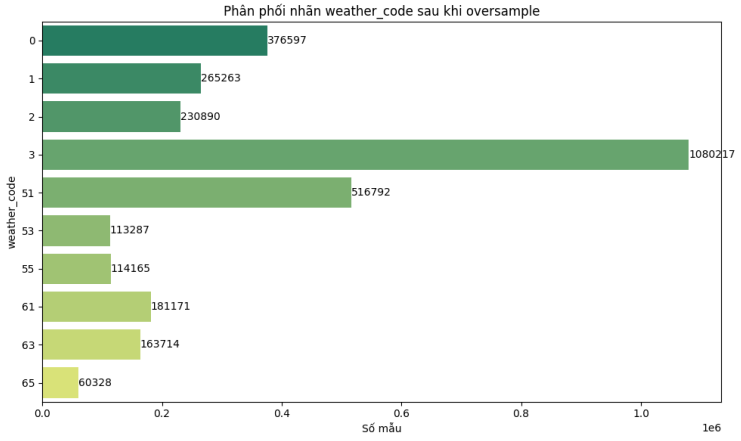


Figure 4.41: Label distribution chart after balancing

After the steps to select the final features before training, these are the selected features.

```
final_selected_features = ['temperature_2m', 'apparent_temperature', 'dew_point_2m', 'precipitation', 'cloud_cover', 'relative_humidity_2m',
                           'wind_gusts_10m', 'wind_speed_10m', 'wind_direction_10m', 'surface_pressure', 'pressure_msl',
                           'sin_hour', 'cos_hour', 'sin_doy', 'cos_doy']

final_X_train = X_train[final_selected_features]
final_X_test = X_test[final_selected_features]
```

Figure 4.42: Features before training

The training method is similar to that for daily data. Here are the training results of the HistGradientBoostingClassifier model.

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     75319
           1       1.00      1.00      1.00     53053
           2       1.00      1.00      1.00     46178
           3       1.00      1.00      1.00    216044
           4       1.00      1.00      1.00    103358
           5       1.00      1.00      1.00     22657
           6       1.00      1.00      1.00     22833
           7       1.00      1.00      1.00     36234
           8       1.00      1.00      1.00     32743
           9       1.00      1.00      1.00     12066

    accuracy                           1.00    620485
   macro avg       1.00      1.00      1.00    620485
weighted avg       1.00      1.00      1.00    620485

--------------------------------------------------
```

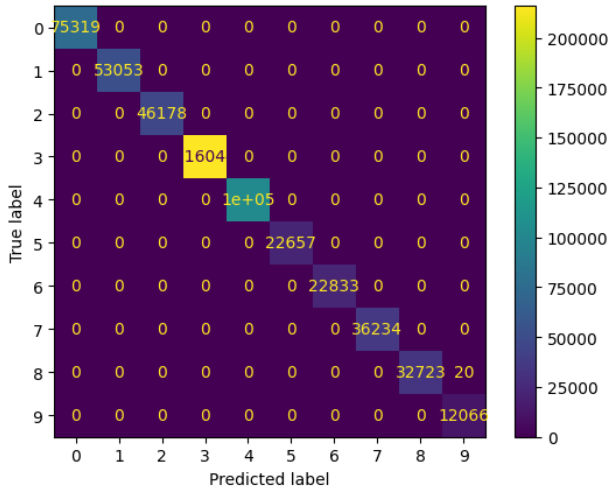


Figure 4.43: Training results of the HistGradientBoostingClassifier model

For hourly data, the training results of the models show very high accuracy and other metrics. Therefore, priority will be given to stability and applicability to select the most suitable model, which is HistGradientBoostingClassifier.

	Model	Accuracy	Precision	Recall	F1
1	DecisionTreeClassifier(random_state=42)	1.000000	1.000000	1.000000	1.000000
4	XGBClassifier(base_score=None, booster=None, c...	0.999987	0.999934	0.999976	0.999955
2	(DecisionTreeClassifier(max_features=3, random...	0.999974	0.999935	0.999923	0.999929
3	HistGradientBoostingClassifier()	0.999968	0.999835	0.999939	0.999887
0	LogisticRegression(max_iter=1000)	0.994665	0.993402	0.995022	0.994201

Figure 4.44: Results of training models with hourly data

After selecting the HistGradientBoostingClassifier model, we proceed to Hyperparameter Tuning and Cross-Validation for this model with the parameters.

```
param_distributions = {
    'learning_rate': [0.05, 0.1],
    'max_depth': [6, 8],
    'max_iter': [150, 250],
    'min_samples_leaf': [50, 100],
    'l2_regularization': [0.0, 0.1]
}

base_hgb = HistGradientBoostingClassifier(
    random_state=42,
    class_weight='balanced',
    early_stopping=False
)

search_hgb = RandomizedSearchCV(
    estimator=base_hgb,
    param_distributions=param_distributions,
    n_iter=5,
    scoring='f1_macro',
    cv=3,
    verbose=2,
    n_jobs=1,
    random_state=42
)

search_hgb.fit(final_X_train, y_train)
```

Figure 4.45: Finetuning the HistGradientBoostingClassifier model

After fine-tuning, the model has improved slightly, so it was decided to use this model to predict weather conditions (hourly).

```
search_hgb.best_score_
```

```
np.float64(0.9991455084621957)
```

```
best_hgb = search_hgb.best_estimator_
print("Best params:", search_hgb.best_params_)
```

```
y_pred = best_hgb.predict(final_X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     75319
           1       1.00      1.00      1.00     53053
           2       1.00      1.00      1.00     46178
           3       1.00      1.00      1.00    216044
           4       1.00      1.00      1.00    103358
           5       1.00      1.00      1.00     22657
           6       1.00      1.00      1.00     22833
           7       1.00      1.00      1.00     36234
           8       1.00      1.00      1.00     32743
           9       1.00      1.00      1.00     12066

    accuracy                           1.00    620485
   macro avg       1.00      1.00      1.00    620485
weighted avg       1.00      1.00      1.00    620485
```

Figure 4.46: Results after fine-tuning the model

4.4 Implementation of Multivariate Forecasting Model Construction (Daily & Hourly)

4.4.1 With Daily Data

In this section, since it is also daily data, there are not too many differences in the preprocessing stage. In the date column processing section, besides creating new columns sin_doy and cos_doy and removing redundant columns date and dayofyear, here 3 additional columns are created: day, month, year.

```
df['date'] = pd.to_datetime(df['date'], errors='coerce')

print(df['date'].dtype)          # datetime64[ns]
print(df['date'].isna().sum())   # 0
```

```
datetime64[ns]
0
```

```
df['year'] = df['date'].dt.year
df['month'] = df['date'].dt.month
df['day'] = df['date'].dt.day
```

```
df['dayofyear'] = df['date'].dt.dayofyear
df['sin_doy'] = np.sin(2 * np.pi * df['dayofyear'] / 365)
df['cos_doy'] = np.cos(2 * np.pi * df['dayofyear'] / 365)

df = df.sort_values(['date']).reset_index(drop=True)
```

```
df = df.drop(columns=['date','dayofyear'])
```

Figure 4.47: Processing the date column

Additionally, data must be resorted by features according to location and time. Then proceed to select input and output features for the model.

```
features = ['temperature_2m_mean', 'temperature_2m_max', 'temperature_2m_min',
            'apparent_temperature_mean', 'apparent_temperature_max',
            'apparent_temperature_min', 'dew_point_2m_mean', 'precipitation_sum',
            'rain_sum', 'snowfall_sum', 'cloud_cover_mean',
            'relative_humidity_2m_mean', 'wind_gusts_10m_mean',
            'wind_speed_10m_mean', 'winddirection_10m_dominant',
            'surface_pressure_mean', 'pressure_msl_mean', 'daylight_duration',
            'sunshine_duration', 'latitude', 'longitude', 'sin_doy', 'cos_doy']

df = df.sort_values(['latitude', 'longitude', 'year', 'month', 'day'])

# ===== Feature cho LSTM (dynamic) =====
SEQ_FEATURES = ['temperature_2m_mean', 'temperature_2m_max', 'temperature_2m_min',
                'apparent_temperature_mean', 'apparent_temperature_max', 'apparent_temperature_min',
                'dew_point_2m_mean', 'precipitation_sum', 'cloud_cover_mean',
                'relative_humidity_2m_mean', 'wind_gusts_10m_mean',
                'wind_speed_10m_mean', 'winddirection_10m_dominant',
                'surface_pressure_mean', 'pressure_msl_mean', 'daylight_duration',
                'sunshine_duration', 'sin_doy', 'cos_doy'
]

# ===== Feature dùng để group city =====
CITY_FEATURES = ['latitude', 'longitude']
```

```
Y_FEATURES = ['temperature_2m_mean', 'temperature_2m_max', 'temperature_2m_min',
              'apparent_temperature_mean', 'apparent_temperature_max', 'apparent_temperature_min',
              'dew_point_2m_mean', 'precipitation_sum', 'cloud_cover_mean',
              'relative_humidity_2m_mean', 'wind_gusts_10m_mean',
              'wind_speed_10m_mean', 'winddirection_10m_dominant',
              'surface_pressure_mean', 'pressure_msl_mean', 'daylight_duration',
              'sunshine_duration'
]
```

Figure 4.48: Selecting features for the model

Split data by city into train and validation sets according to time series, while avoiding data leakage due to the input window.

Figure 4.49: Splitting data into train and validation sets

Normalize input features (X) and target variable (y) based on the train set, then save these scalers to serve prediction later.

```
scaler_x = StandardScaler()
scaler_y = StandardScaler()

scaler_x.fit(df_train_raw[SEQ_FEATURES])
scaler_y.fit(df_train_raw[Y_FEATURES])
```


Figure 4.50: Normalizing input features

Apply the scaler learned from the train set to normalize both train and validation data, ensuring the same scale.

```
df_train = df_train_raw.copy()
df_val   = df_val_raw.copy()

df_train[SEQ_FEATURES] = scaler_x.transform(df_train_raw[SEQ_FEATURES])
df_train[Y_FEATURES]   = scaler_y.transform(df_train_raw[Y_FEATURES])

df_val[SEQ_FEATURES] = scaler_x.transform(df_val_raw[SEQ_FEATURES])
df_val[Y_FEATURES]   = scaler_y.transform(df_val_raw[Y_FEATURES])
```

Figure 4.51: Normalizing features for both train and val sets

Create data sequences by city for the time series forecasting problem (sliding window), with X being the input window and y being the future forecast window.

```python
def create_sequences_by_city(
    df,
    city_cols=('latitude', 'longitude'),
    input_window=30,
    output_window=7,
    seq_features=None,
    y_features=None
):
    X, y = [], []

    for _, g in df.groupby(list(city_cols)):
        g = g.sort_values(['year', 'month', 'day'])

        if len(g) < input_window + output_window:
            continue

        seq_data = g[seq_features].values
        y_data = g[y_features].values

        for i in range(len(g) - input_window - output_window + 1):
            X.append(seq_data[i:i + input_window])
            y.append(y_data[i + input_window:i + input_window + output_window])

    return np.array(X), np.array(y)
```

Figure 4.52: Creating data sequences by city

Create train/validation datasets in sequence form for the forecasting model, with 30 days input and 7 days output, while checking correct data dimensions.

```python
X_train, y_train = create_sequences_by_city(
    df_train,
    city_cols=CITY_FEATURES,
    input_window=30,
    output_window=7,
    seq_features=SEQ_FEATURES,
    y_features=Y_FEATURES
)

X_val, y_val = create_sequences_by_city(
    df_val,
    city_cols=CITY_FEATURES,
    input_window=30,
    output_window=7,
    seq_features=SEQ_FEATURES,
    y_features=Y_FEATURES
)

print("X shape:", X_train.shape)  # (samples, 30, len(SEQ_FEATURES))
print("y shape:", y_train.shape)  # (samples, 7, len(Y_FEATURES))
print("X shape:", X_val.shape)  # (samples, 30, len(SEQ_FEATURES))
print("y shape:", y_val.shape)  # (samples, 7, len(Y_FEATURES))
```

```
X shape: (87885, 30, 19)
y shape: (87885, 7, 17)
X shape: (20790, 30, 19)
y shape: (20790, 7, 17)
```

Figure 4.53: Creating sequence data with 30-day input and 7-day output

Build and configure the LSTM model to forecast the time series for the next 7 days from 30 days of input.

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 30, 64)	21,504
dropout (Dropout)	(None, 30, 64)	0
lstm_1 (LSTM)	(None, 32)	12,416
dropout_1 (Dropout)	(None, 32)	0
dense (Dense)	(None, 119)	3,927
reshape (Reshape)	(None, 7, 17)	0

Total params: 37,847 (147.84 KB)
Trainable params: 37,847 (147.84 KB)
Non-trainable params: 0 (0.00 B)

Figure 4.54: Configuring the LSTM model

Training Process

- Val MAE decreased sharply from ~0.57 → ~0.42, then converged stably from epoch ~24.

- Train MAE (~0.36) < Val MAE (~0.42) → no overfitting, the gap is acceptable.

- ReduceLROnPlateau worked effectively, each time LR was reduced, val MAE improved clearly.

Model Accuracy

- Val MAE ≈ 0.42 (normalized) → mean error ~0.42σ, a quite good level for 7-day weather forecasting.

- Huber loss helps the model be stable with outliers (consistent with the characteristics of meteorological data).

```
print("Train y range:", y_train.min(), y_train.max())
print("Val y range  :", y_val.min(), y_val.max())
```

```
Train y range: -6.038649418414488 48.092022997767344
Val y range  : -5.1307132527767125 22.389014156900586
```

```
print("Train std:", y_train.std(axis=(0, 1)))
print("Val std  :", y_val.std(axis=(0, 1)))
```

```
Train std: [0.9935572  0.99356085 0.99379242 0.99416239 0.99446265 0.99412628
 0.99473157 1.00658036 0.9968656  0.99759725 0.99812378 0.9987109
 1.0016679  0.99943028 0.99723218 0.98990906 1.00057597]
Val std  : [0.90532216 0.95049148 0.88302255 0.90296291 0.93575131 0.89117336
 0.89870068 1.01713445 1.0376214  1.02086089 0.95892882 0.99398066
 0.99550651 0.9937428  0.96792779 0.96725112 0.9932446 ]
```

Figure 4.55: Distribution of y values and standard deviation

Distribution of y values

- Val y range lies completely within the train range → no out-of-distribution.

- Train has a larger amplitude → the model learned from more difficult situations than validation.

Standard Deviation (std)

- Train std ≈ 1 → correctly normalized.

- Val std is slightly lower than train in most variables → validation has less noise, explaining the stable val MAE.

- No feature has serious distribution skew.

The processing and training pipeline is built standardly for time series problems: data is separated by city, normalized appropriately, and no information leakage occurs. The training process shows that the model converges well, val MAE decreases stably from ~0.57 down to ~0.42 and saturates early, proving good generalization capability and no signs of overfitting. The target value distribution between train and validation is consistent, val lies completely within the train domain, standard deviation after normalization is approximately 1, and there is no serious distribution skew between features. Overall, the LSTM model achieves quite good accuracy, stability, and reliability, suitable for application to the 7-day medium-term multivariate weather forecasting problem.

4.4.2 With data hourly

Overall, the implementation process with hourly data has few differences compared to daily data, as they follow a unified time series processing pipeline. The difference mainly lies in time resolution and how hourly features are extracted.

In the preprocessing step, the datetime column is split into hour, day, month, year features and the 24-hour cycle is encoded using sin_hour, cos_hour variables. After extracting features, redundant time columns are removed. Data continues to be sorted by city and chronological order, then input feature set (X) and target variable (y) are selected.

Data is split into train/validation by city and time series, while removing overlapping parts due to the input window to avoid data leakage. Input features and target variables are normalized based on the train set and applied uniformly to the validation set.

Finally, data is converted to sequence form (sliding window) by hour and fed into the LSTM model for multivariate forecasting. Training results show the model converges stably, validation error decreases steadily, and there are no signs of overfitting, showing that the processing pipeline is suitable and the model has good generalization capability for the hourly weather forecasting problem.

CHAPTER 5 – BUILDING THE INTELLIGENT RECOMMENDATION SYSTEM

5.1 Building the Farming Schedule Recommendation System

5.1.1 Overview and Problem Statement

In the context of modern agricultural farming, weather forecast data plays a key role but often exists in the form of raw indicators (temperature, rainfall, humidity, wind speed, etc.). Farmers often face difficulties in converting these technical figures into specific farming decisions. The "7-Day Schedule Creation" function of the AgriWeather application was built to solve this problem. The system not only displays weather information but also acts as a Decision Support System, automatically proposing specific tasks for each day based on crop characteristics and local weather developments.

5.1.2 Data Processing Process and Prompt Engineering Techniques

The system operates on the Groq AI Integration infrastructure, utilizing the llama-3.3-70b-versatile Large Language Model as its central processing core. The integration of Groq accelerates inference speeds, enabling the system to generate intelligent farming recommendations (Smart Task Generation) near-instantaneously. The specific processing workflow is as follows:

First, Multi-dimensional Data Aggregation and Pre-processing: When a user initiates a plan, the system not only collects basic information but also aggregates detailed environmental factors to ensure maximum accuracy:

- Farming Information: Includes Crop Type, specific Geographical Location, and Seasonal Goals targeted by the farmer.

- Comprehensive Weather Data: The Weather API provides 7-day forecast data, including critical indices directly affecting crops such as temperature, precipitation, humidity, and Wind conditions.

Second, Prompt Engineering and Weather-Aware Planning: This is the pivotal step in transforming raw data into Optimal Farming Schedules. The system applies Prompt Engineering techniques with a "Weather-Aware" mechanism, requiring the model to perform the following tasks:

- Contextual Analysis: Cross-referencing crop growth characteristics with weather patterns (e.g., What are the pest and disease risks associated with high humidity combined with low temperature?).

- Smart Task Generation: Automatically proposing specific daily tasks that align with actual conditions (e.g., Avoiding pesticide spraying when strong winds or rain are forecast).

Output: The data is returned in a structured format to be rendered as a visual 7-Day Calendar, allowing farmers to easily track the Task breakdown day by day.

5.1.3 Output and Display

The results returned from groq are strictly standardized in JSON Array format. Each element in the array corresponds to one day, containing information fields: Task Title, Detailed Technical Description, Warning Level, and Representative Icon. The Frontend application parses this JSON string to visually display it as a list of Task Cards for 7 days, helping farmers easily monitor and implement.

5.2 Building the Agricultural Virtual Assistant Chatbot

5.2.1 Intelligent Routing Architecture

Unlike traditional chatbots that operate based on static scripts, AgriWeather's Chatbot is designed as a Context-Aware virtual assistant. The special feature of this module is the ability to automatically classify user intent (Intent Classification) through "System Instructions" to route the answer into one of three specialized processing flows:

- ***Flow 1***: Weather Consultation (Weather Flow): When users ask questions related to environmental indicators (e.g., "Will it rain tomorrow?", "What is the current temperature?"), the Chatbot will directly retrieve real-time weather data stored in the working session (Session Context). This ensures the answer always has absolute accuracy based on data, avoiding the situation where the model "fabricates" information (Hallucination).

- ***Flow 2***: Agricultural Technique (Agriculture Flow): When the question relates to farming techniques (pests, fertilizers, soil preparation), the Chatbot will activate the Agricultural Engineer role. Based on the crop information (Input: Rice) and Season (Input: Winter-Spring) that the user entered, the Chatbot will provide specific advice. For example, for the same question about fertilization, but with the Winter-Spring crop, the Chatbot will prioritize advising Phosphorus and Potassium fertilization to help plants harden against the cold, different from the process for the Summer-Autumn crop.

- ***Flow 3***: General Communication (General Flow): For social interactions or questions outside the scope of agriculture, the system is programmed to respond politely and skillfully navigate the user back to farming support topics.

5.2.2 Integration and User Experience

Regarding the interface, the Chatbot is integrated as a Popup window operating in parallel with the Farming Schedule screen. This design creates a seamless experience: users can view the list of suggested tasks for 7 days while asking Follow-up questions to the Chatbot immediately. The entire context regarding the farm and weather is implicitly sent (injected) into each chat turn, helping users avoid repeating context information, creating the feeling of chatting with a real expert who understands the current status of the farm.

5.3 System Integration Processing Workflow

To ensure consistency between forecasting and recommendation functions, the AgriWeather system operates on a multi-layer pipeline architecture. The process is designed to combine the computational power of Deep Learning models in weather forecasting with the semantic reasoning capabilities of Large Language Models (LLM).

The overall processing workflow is divided into 4 main phases as follows:

5.3.1 Initialization and Positioning (Phase 1)

This is a mandatory preprocessing step to define the spatial context for the entire system.

- Trigger: User enters city/location name.

- Processing: The system calls the OpenWeatherMap Geocoding API to convert the location name into geographic coordinates (latitude, longitude).

- Storage: Coordinates are saved to the Global State for shared use by subsequent modules, avoiding repeated geographic API calls.

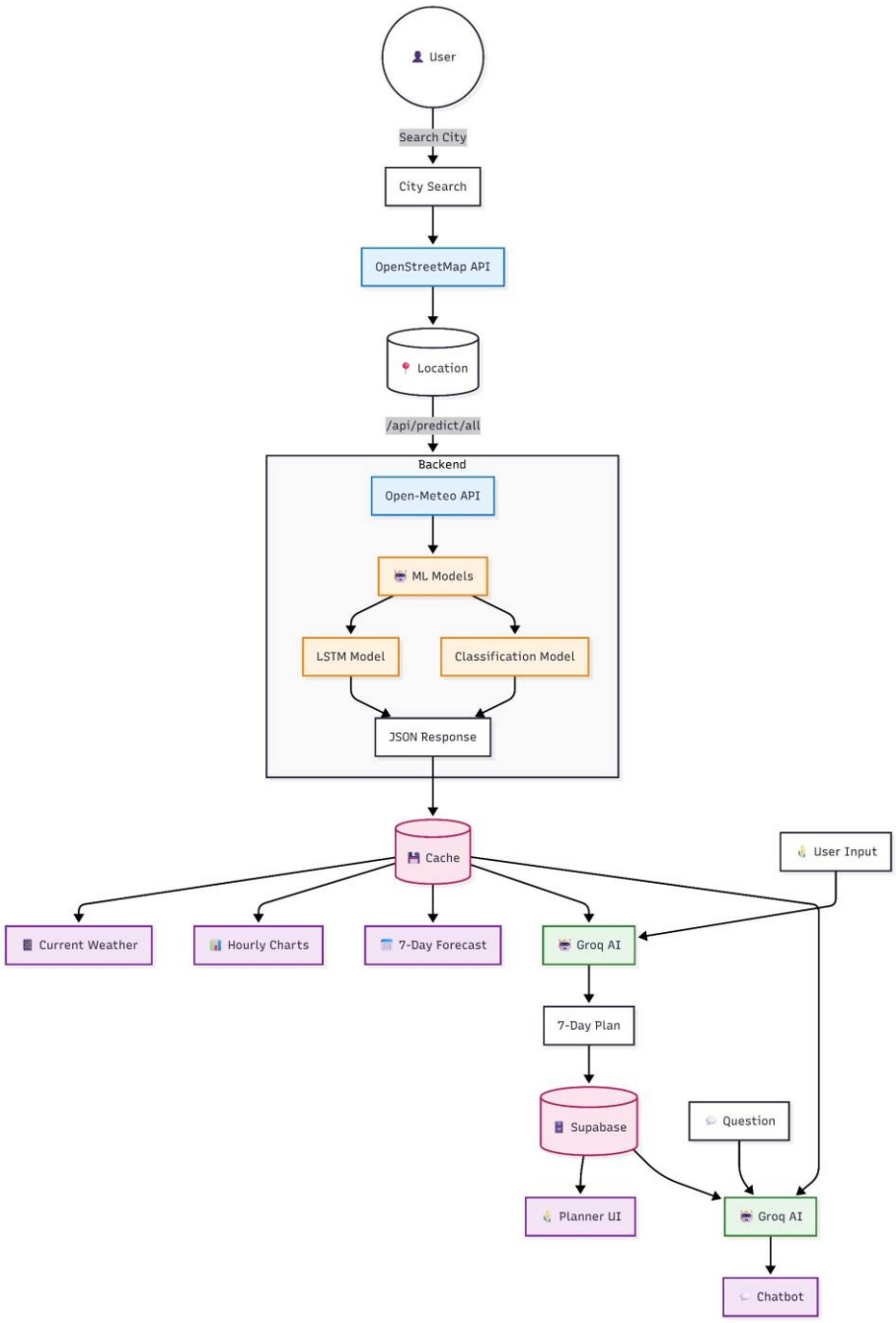


Figure 5.1: Pipeline of the AgriWeather system

5.3.2 Weather Forecasting Core (Phase 2)

This is the central processing layer using trained Deep Learning models to generate weather data.

- Data Collection: The system calls the API (Open-Meteo Historical) to retrieve past 30-day data at the defined coordinates.

- 7-Day Prediction (Multi-Model Architecture): The system deploys 03 specialized models (Hourly, Daily, and 7-day Forecast) to process distinct data layers. Historical data is fed into model_7day (LSTM/Dense architecture) to forecast quantitative metrics for the next 7 days. The prediction_raw output is stored for reuse by the recommendation module.

- Weather Classification (Classification Task): The system loops through the prediction_raw result, using the votingC classification model to assign weather labels (Rainy, Sunny, Cloudy...) for each day.

- UI Sync:

+ Tab 3 (7-Day Forecast): Displays charts from prediction and icons from classification labels.

+ Tab 1 (Current Weather): Extracts data from the first day (index 0) from the forecast sequence to display "Real-time" current weather (simulated from AI).

5.3.3 Agricultural Recommendation System (Phase 3)

This phase performs the conversion of numerical data from Phase 2 into agricultural advisory language via GenAI.

- Input Aggregation: The system joins prediction_raw data (output of Phase 2) with farming information from the user (Crop, Season). Using the exact forecast result of model_7day as input for Groq ensures that agricultural advice perfectly matches the weather chart the user sees.

- Processing (Generative Process):

+ Create Prompt containing CSV weather context and request JSON output.

+ Send request to Groq API.

- Output: Receive JSON list of tasks (Tasks) and render onto the "7-Day Schedule" interface.

5.3.4 Interactive Virtual Assistant (Phase 4)

This module runs in parallel and listens for events from the user.

- Context Injection Mechanism: Whenever a user sends a message, the system silently packages the current state (Location, Today's Weather from Tab 1, Agricultural Plan from Phase 3) into the System Instruction.

- Routing: groq analyzes the question and answers according to 3 flows (Weather/Agriculture/General) as designed in section 5.2.

5.3.5 Execution Technology Architecture

To realize the complex multi-layer processing workflow mentioned above, the system is built based on a modern Client-Server model, clearly separating responsibilities between Frontend and Backend:

Data Infrastructure (Supabase)

Serves as the persistent storage layer and manages Authentication. Supabase stores user profiles, farming plans, and chat history, ensuring data integrity across the system.

Backend (Python & FastAPI)

Acts as the central coordinator, connecting Deep Learning models with external APIs (groq, OpenWeather). Python is suitable for AI/ML, while FastAPI provides high-performance RESTful APIs with asynchronous processing capabilities, ensuring fast response and not blocking the main forecast flow.

Frontend (React, TypeScript & Tailwind CSS)

React is responsible for building the interface according to component architecture and managing UI state. TypeScript helps ensure data type safety when processing complex weather data, while Tailwind CSS supports modern, responsive, and consistent dashboard design across multiple devices.

5.3.6 Demo Website AgriWeather

- Registration Interface.

Figure 5.2: Registration Interface

- Login Interface.

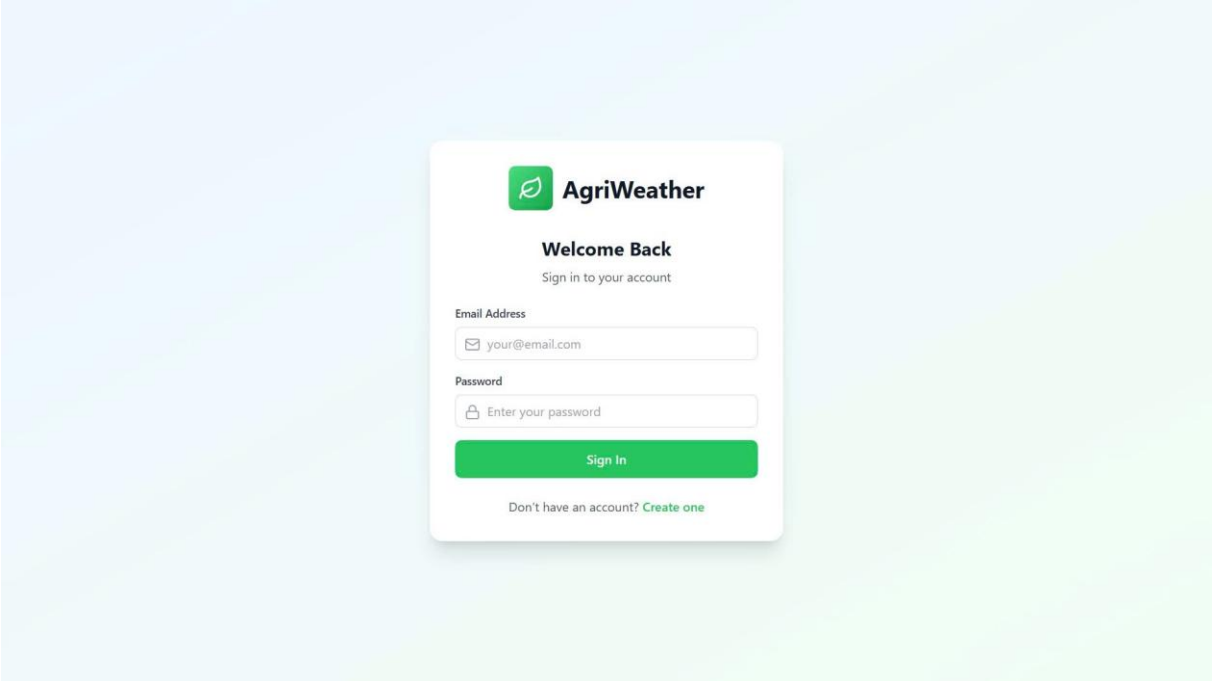


Figure 5.3: Login Interface

- Weather Forecast Interface - Tab Weather Today.

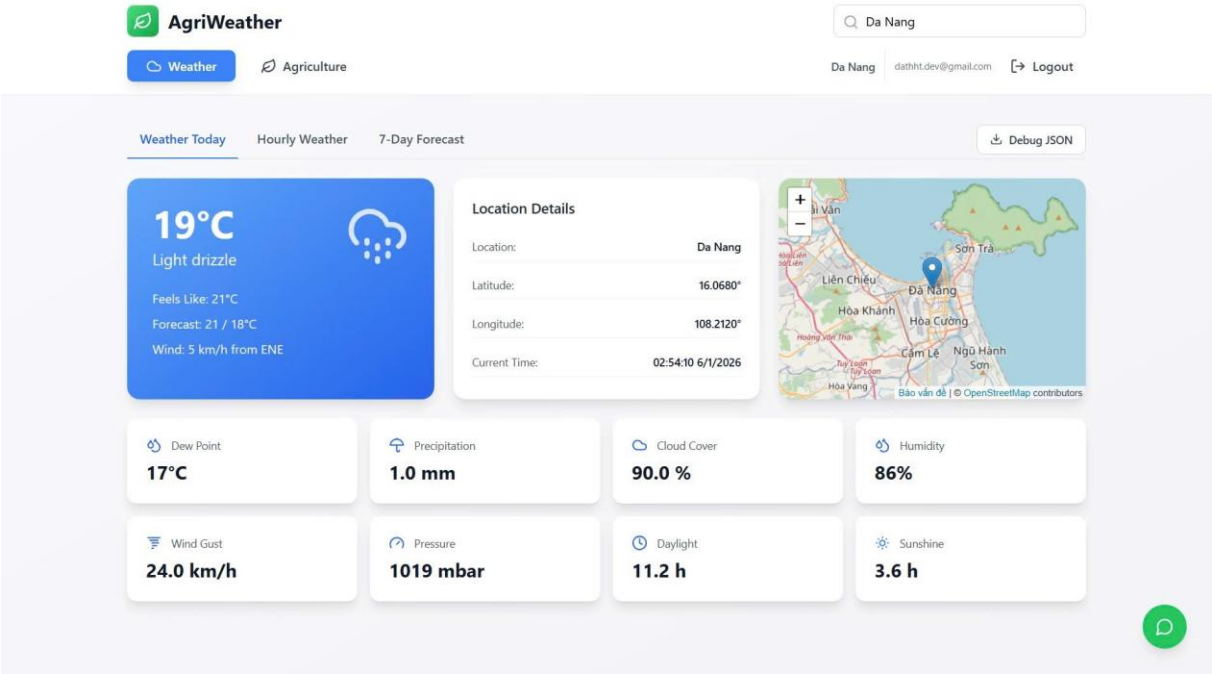


Figure 5.4: Weather Forecast Interface - Tab Weather Today

- Weather Forecast Interface - Tab Hourly Weather.

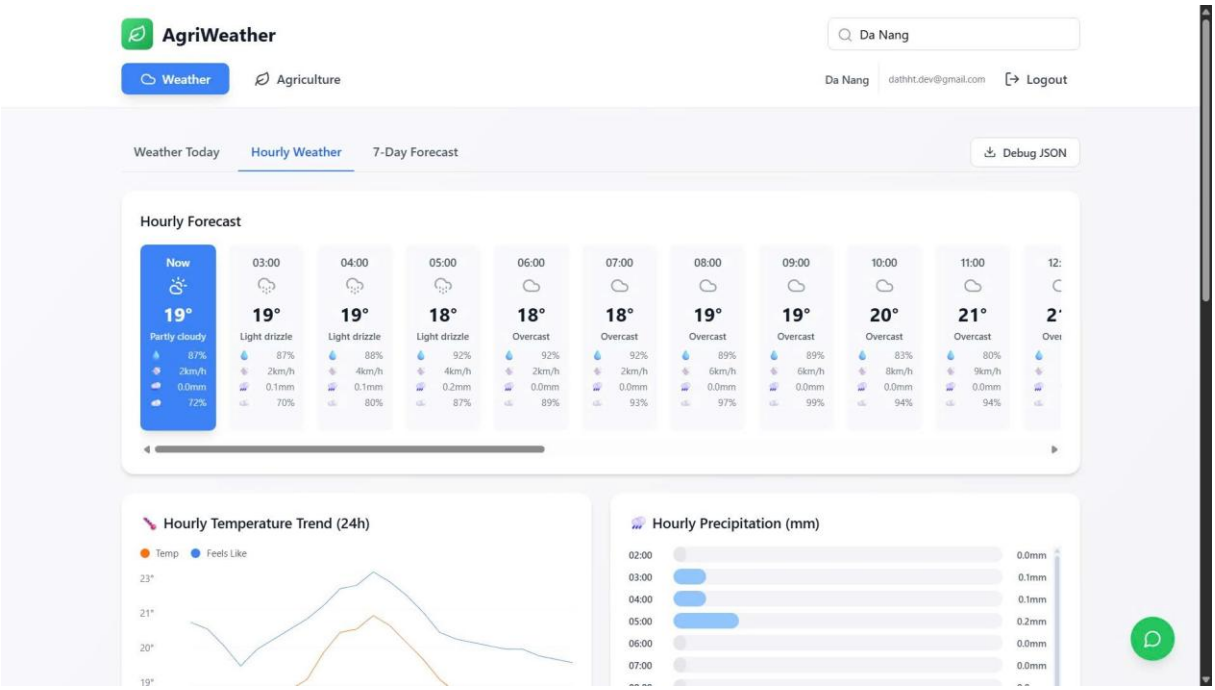


Figure 5.5: Weather Forecast Interface - Tab Hourly Weather 1

- Weather Forecast Interface - Tab Hourly Weather - Temperature and Precipitation chart.

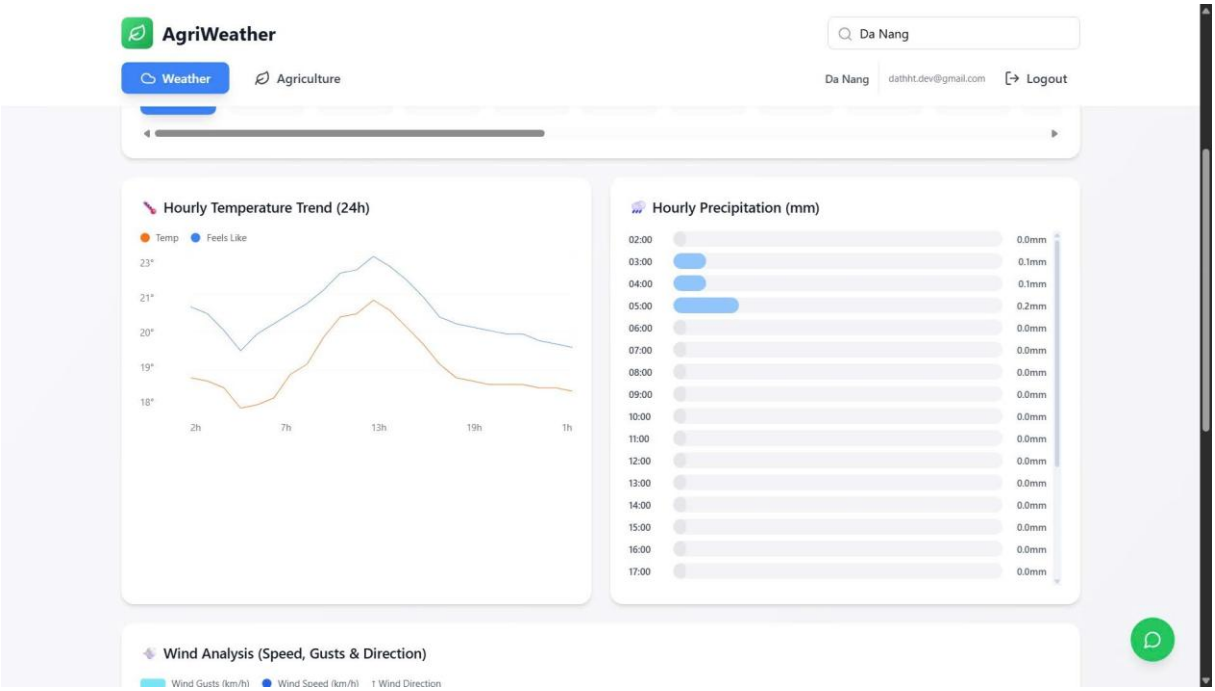


Figure 5.6: Weather Forecast Interface - Tab Hourly Weather 2

- Weather Forecast Interface - Tab Hourly Weather – Wind and Clould, Humidity chart.

Figure 5.7: Weather Forecast Interface - Tab Hourly Weather 3

- Weather Forecast Interface - Tab 7-day Forecast.

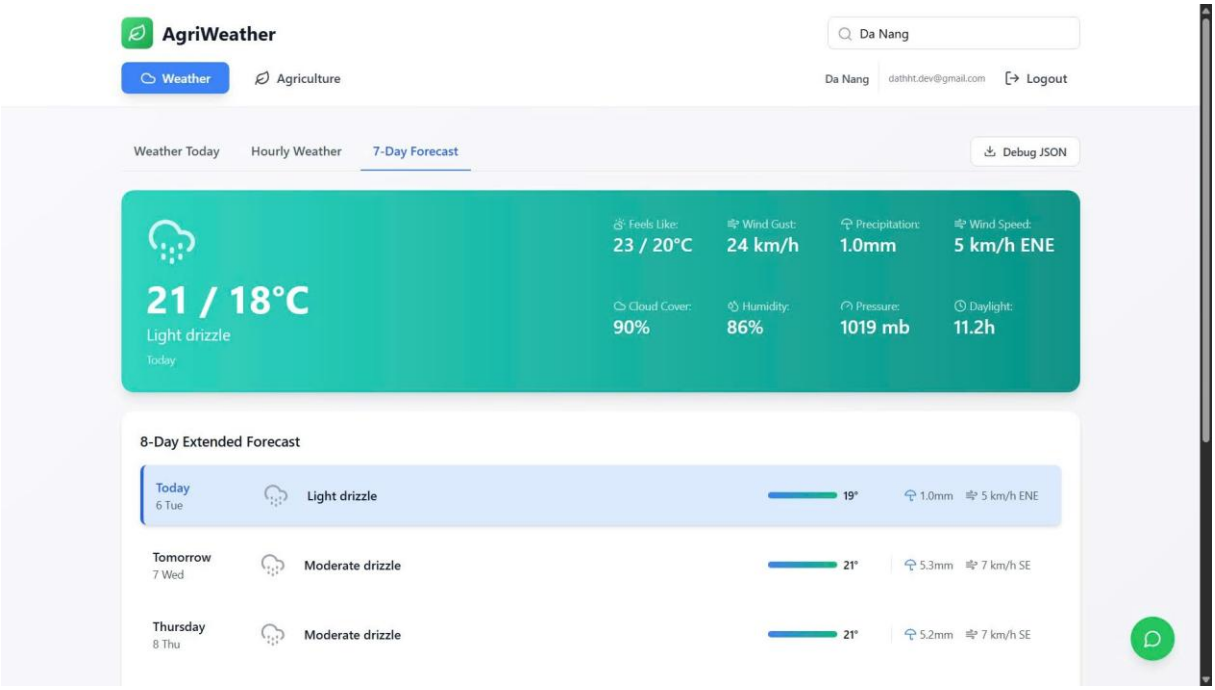


Figure 5.8: Weather Forecast Interface - Tab 7-day Forecast

- Agricultural Recommendation Interface.

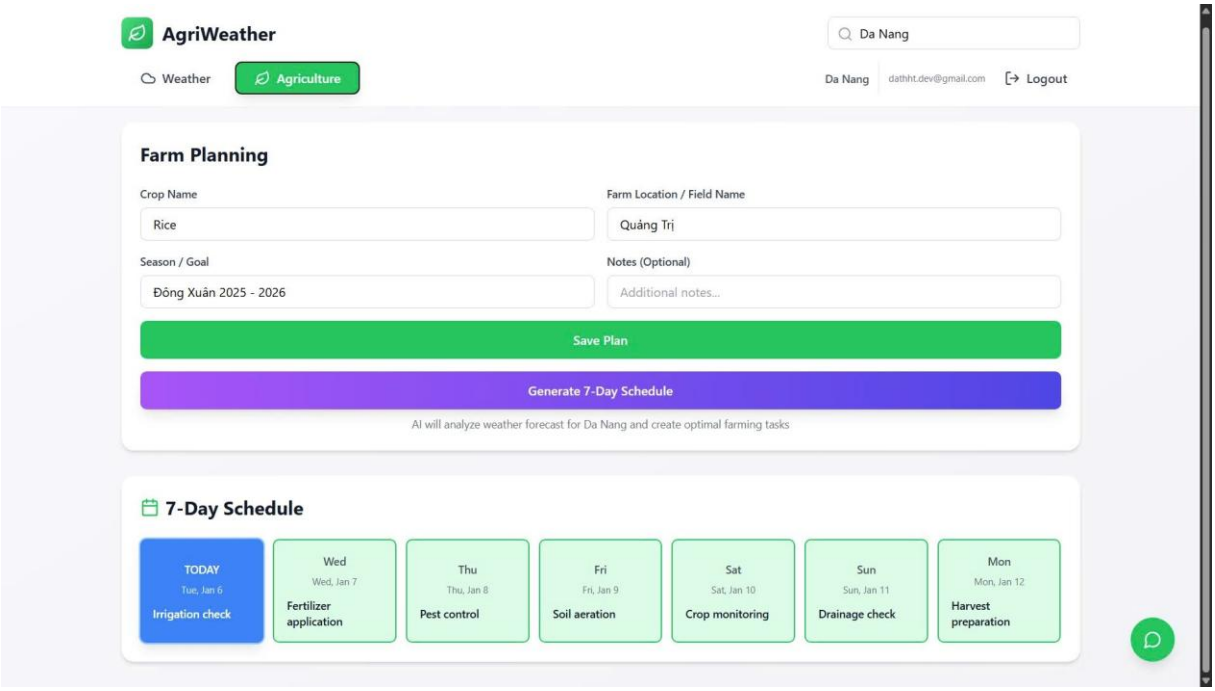


Figure 5.9: Agricultural Recommendation Interface

- Specific Agricultural Recommendation Interface.

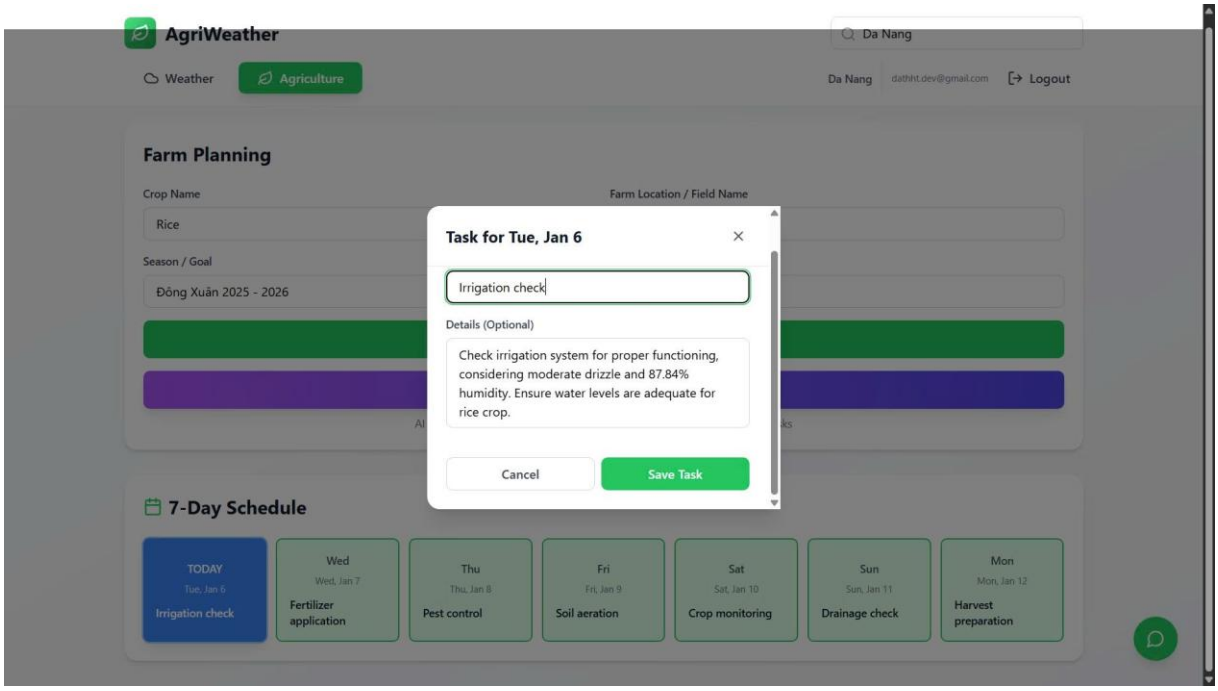


Figure 5.10: Specific Agricultural Recommendation Interface

- Chatbot Interface.

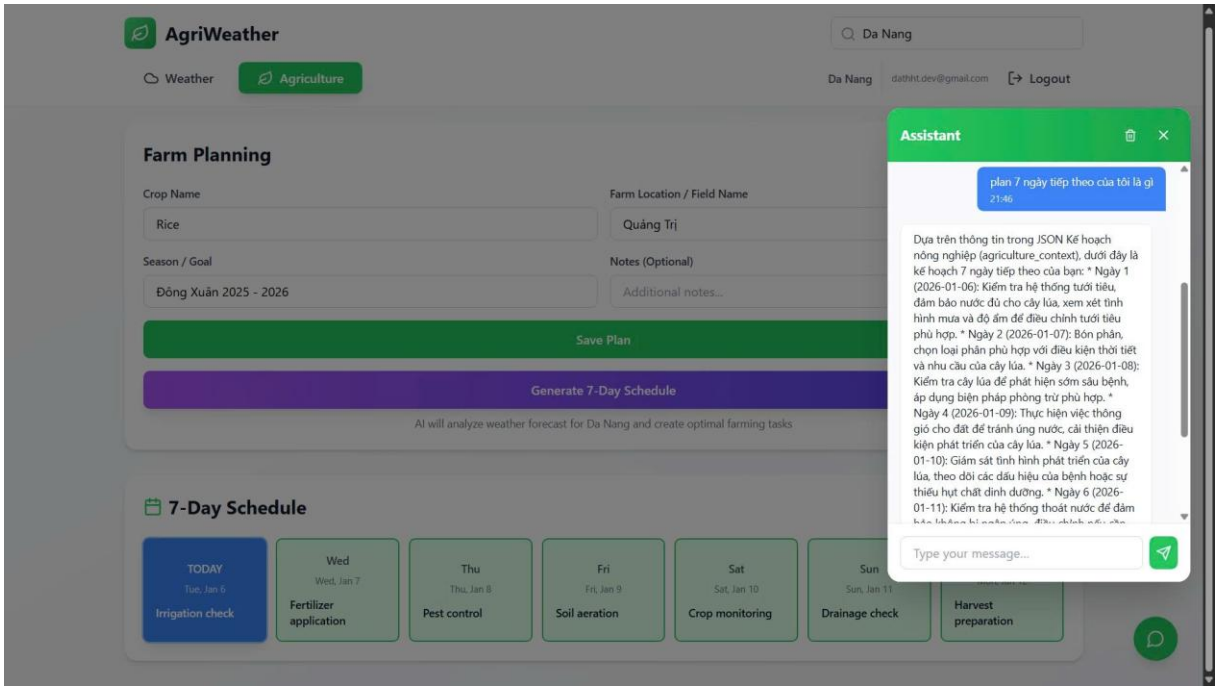


Figure 5.11: Chatbot Interface

CONCLUSION

1. Results Achieved

The thesis has successfully built a multivariate weather forecasting and intelligent agricultural recommendation system, applying machine learning, deep learning, and large language model techniques. The system processes weather data as multivariate time series, performing forecasts across two timeframes: daily and hourly, thereby providing detailed information close to practical usage needs.

Forecasting models were deployed following a process including data collection, preprocessing, exploratory data analysis, feature engineering, model training, and evaluation. Experimental results show that deep learning models, especially LSTM, effectively exploit time-dependent relationships and interactions between meteorological factors, creating a foundation for subsequent modules.

Based on forecast results, the system built an agricultural recommendation module, allowing the conversion of weather data into specific farming recommendations according to crop type, geographic location, and growth stage. Simultaneously, a virtual assistant chatbot was integrated to support users in looking up information, explaining recommendations, and interacting with the system conveniently. All components were integrated into a complete system with a website demo, demonstrating the feasibility of applying AI in agriculture and daily life.

2. Limitations

Weather Data The weather data used in the system is mainly based on public sources and historical data; it does not yet fully cover spatial, temporal, and related environmental factors, leading to a limited ability to reflect actual conditions in some cases.

Model Accuracy Although deep learning models like LSTM yield promising results, forecast accuracy still depends heavily on the quality and scale of input data, especially under strongly fluctuating or abnormal weather conditions.

Recommendation System Not Yet Verified in Long-term Practice Farming recommendations are currently only evaluated through simulation experiments and have not yet been deployed or monitored for effectiveness in long-term agricultural production practice.

Virtual Assistant Chatbot Lacks Adaptive Learning Capability The Chatbot currently operates based on pre-existing knowledge and rules; it does not yet have a mechanism to learn from user feedback to improve response quality and personalization levels.

System Performance and Scalability are Limited The system has not been comprehensively optimized for processing big data and a high number of concurrent users, causing limitations when deploying at a large scale.

3. Future Development

Expand and diversify weather data Supplement with additional real-time data sources such as IoT sensors, satellite data, and environmental data to increase the coverage and reliability of input data.

Improve the accuracy of forecasting models Apply pre-trained models for time series and weather forecasting (such as deep learning models trained on large datasets),

combined with fine-tuning on local data to improve generalization capabilities and forecast accuracy.

Deploy and evaluate the recommendation system in practice Conduct system testing in actual farming environments, collect feedback, and evaluate the effectiveness of recommendations over a long period to refine the decision-making model.

Develop the chatbot towards adaptive learning Integrate mechanisms for learning from user feedback and maintaining conversation context, helping the chatbot become increasingly accurate and more suitable for each specific user group.

Optimize system performance and scalability Improve system architecture towards microservices, optimize parallel processing, and deployment infrastructure to meet large-scale usage demands.

4. Conclusion

The thesis "Building an intelligent agricultural recommendation system based on multivariate weather forecasting models" has achieved the set research objectives, constructing an integrated system from weather forecasting and agricultural recommendations to user interaction. Research results show great potential for applying artificial intelligence in supporting agricultural decision-making, contributing to improving production efficiency and aiming for sustainable agricultural development in the current context of climate change.

REFERENCES

[1] vnptAI.io, “LSTM Model,” *vnptAI*. [Online]. Available at: *https://vnptai.io/vi/blog/detail/mo-hinh-lstm*

[2] OpenWeatherMap, *OpenWeatherMap API — Weather Data*. [Online]. Available at: *https://openweathermap.org/*

[3] Open-Meteo, *Open-Meteo — Free Weather API*. [Online]. Available at: *https://open-meteo.com/*

[4] scikit-learn.org, *Histogram Gradient Boosting Classifier — scikit-learn*. [Online]. Available at: *https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.HistGradientBoostingClassifier.html*

[5] scikit-learn.org, *VotingClassifier — scikit-learn*. [Online]. Available at: *https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html*

[6] GeeksforGeeks, “Deep Learning | Introduction to Long Short-Term Memory,” *GeeksforGeeks*. [Online]. Available at: *https://www.geeksforgeeks.org/deep-learning/deep-learning-introduction-to-long-short-term-memory/*

[7] FastAPI, “FastAPI — tiangolo.com.” [Online]. Available at: *https://fastapi.tiangolo.com/*

[8] React, *React Developer Documentation*. [Online]. Available at: *https://react.dev/*

[9] Lucide.dev, *Lucide React Guide*. [Online]. Available at: *https://lucide.dev/guide/packages/lucide-react*

[10] Tailwind CSS, *Tailwind CSS Documentation*. [Online]. Available at: *https://tailwindcss.com/*

[11] GROQ Console, *llama-3.3-70b-versatile Model Documentation*. [Online]. Available at: *https://console.groq.com/docs/model/llama-3.3-70b-versatile*

[12] Google Cloud, *Vertex AI — Gemini 2.5 Flash Models Documentation*. [Online]. Available at: *https://docs.cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-5-flash*

[13] Bolt Support, “Cloud Database — Documentation.” [Online]. Available at: *https://support.bolt.new/cloud/database*