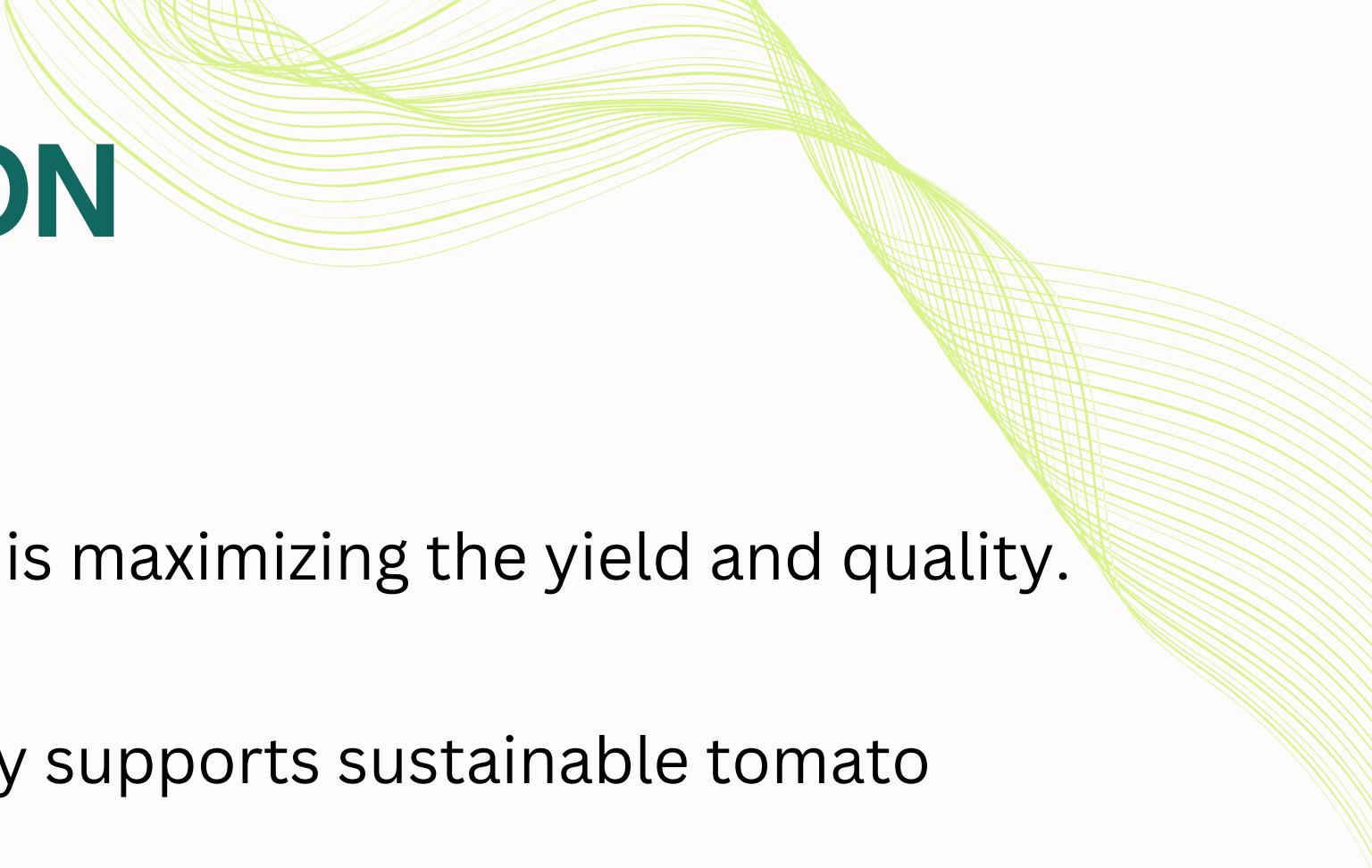


SMART GREEN HOUSES DECISION SUPPORT SYSTEMS FOR TOMATO CULTIVATION

GROUP ID: 24-25J-064



INTRODUCTION



- Maintaining optimal conditions in tomato greenhouses is maximizing the yield and quality.
- Managing watering, fertilizing, and harvesting efficiently supports sustainable tomato farming.
- Machine learning can predict watering needs, fertilization, harvest times, disease identification and treatment recommendations.
- Combining IoT and ML, the project aims to make greenhouse management more efficient

OBJECTIVES OF MEMBERS

Optimize Irrigation control & Watering Schedules :

- Determine the best timing and frequency for watering to ensure healthy growth and efficient water use.

Optimize Fertilizing Schedules and Types :

- Identify the most effective fertilizing routines and products to enhance nutrient availability and promote robust tomato plants.

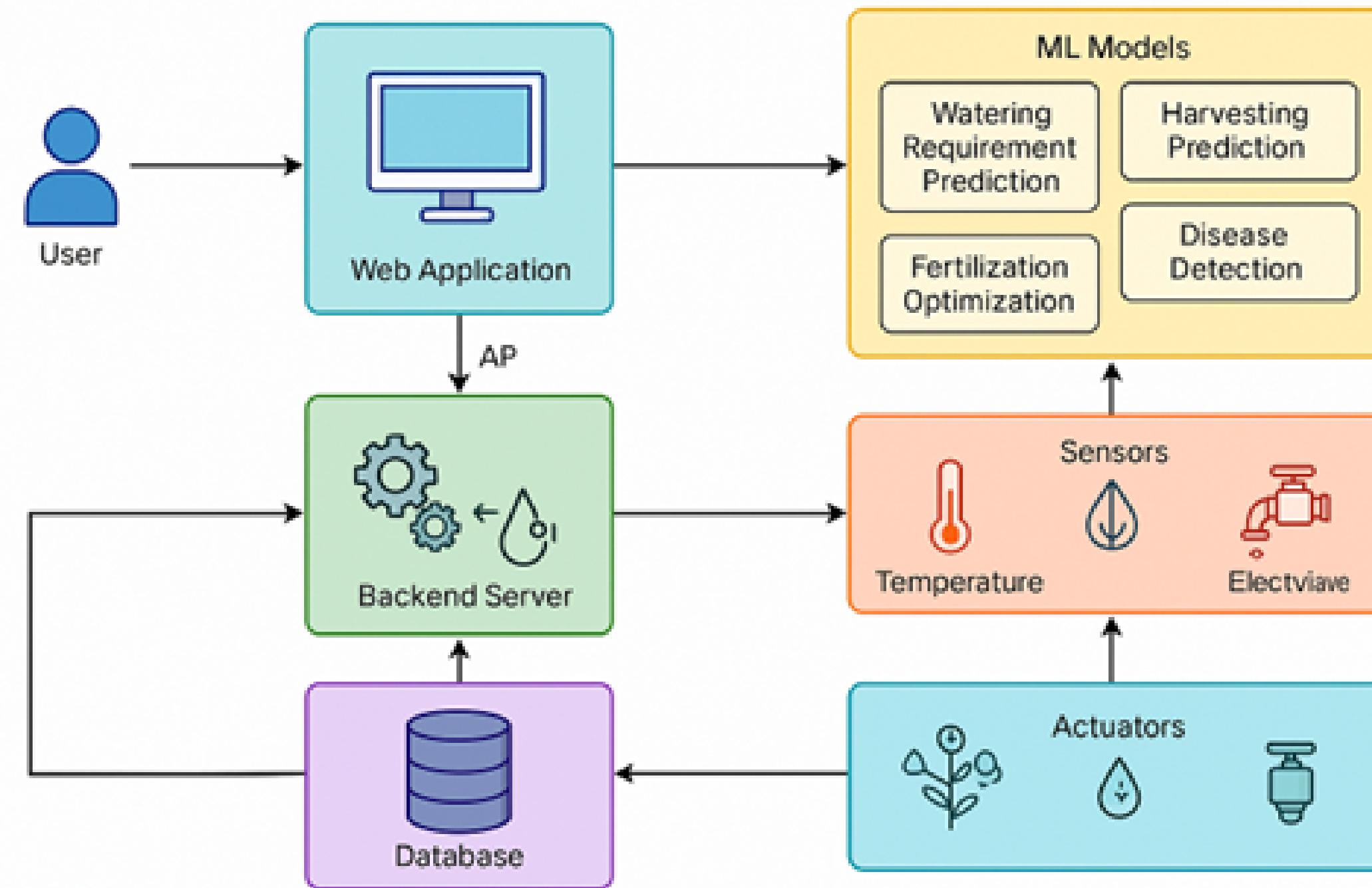
Optimize Harvesting Schedules :

- Establish the ideal timing for harvesting tomatoes to maximize yield and quality.

Disease identification and treatment recommendations

- Develop ml model to identify, prevent, and treat common tomato diseases and provide recommendations to maintain plant health.

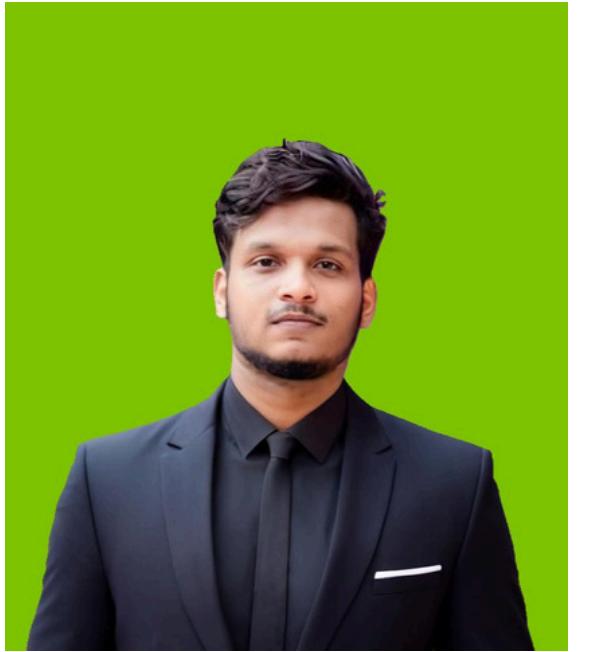
FULL SYSTEM ARCHITECTURE DIAGRAM



GROUP MEMBERS AND SUPERVISORS DETAILES

1. THRIMAVITHANA .V .D - IT21181160
2. ASARDEEN .A - IT21231896
3. NAJAS .M .N .M - IT21186592
4. JAYANETHTHI .I .H .N .S - IT21231414

SUPERVISOR – Mrs. Geethanjali Wimalaratne
CO-SUPERVISOR – Mr. Samantha Thelijjagoda



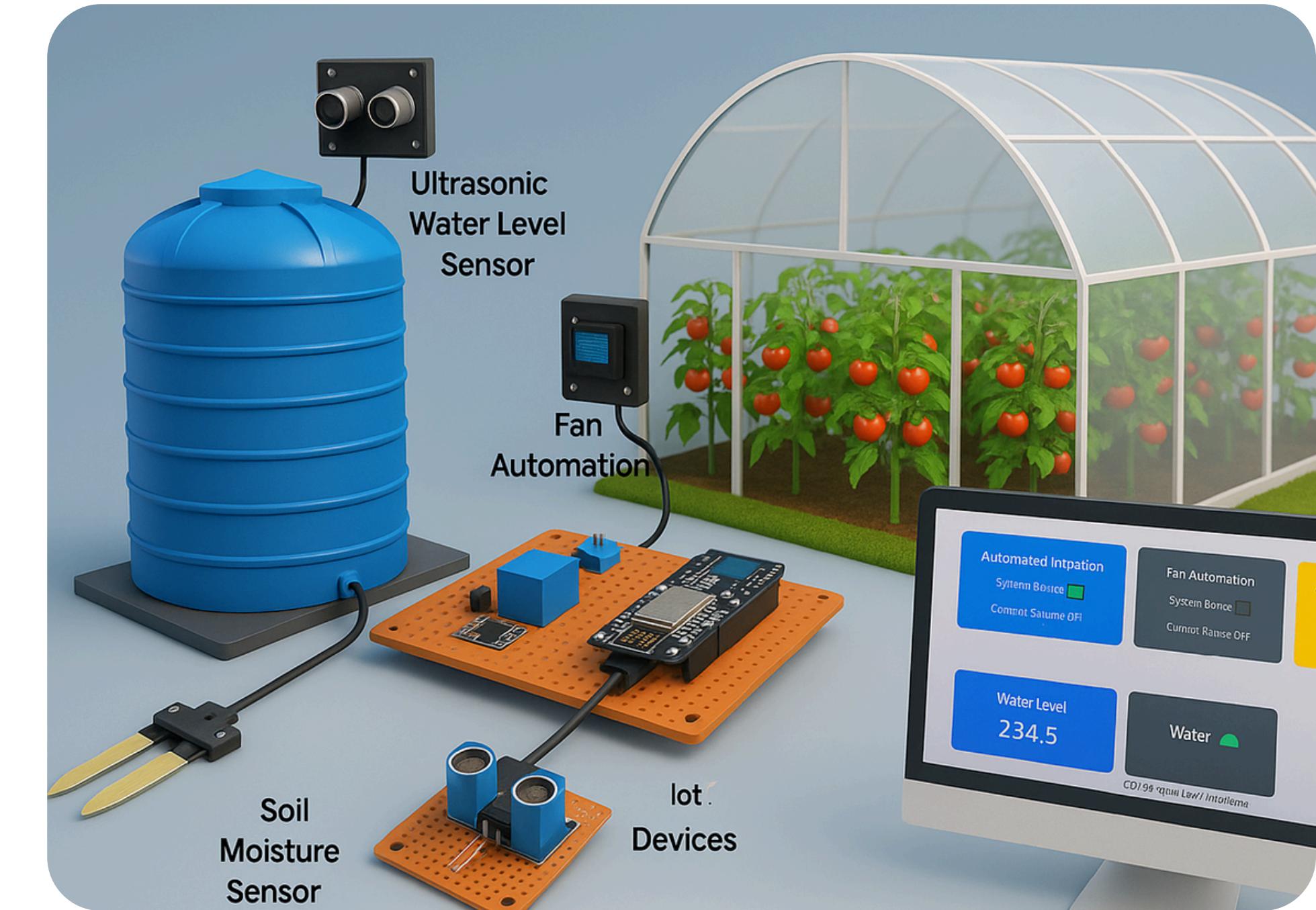
IT21231896 | ASARDEEN .A

Information technology



MY COMPONENT

**IoT and Machine Learning-Based
Water Requirement Prediction
and Automation Control System
for Greenhouse Irrigation**



INTRODUCTION BACKGROUND

- **Water management** is vital for tomato greenhouses.
- Environmental factors like **temperature, humidity, light, and soil moisture** directly impact plant water needs.
- **Photosynthesis** efficiency depends on:
 - Adequate **light, CO₂, temperature, and water**.
 - Poor irrigation leads: **limited energy production**, affecting growth & yield.
- **IoT + ML models** monitor real-time data to maintain optimal conditions that **enhance photosynthesis, improve crop health, and boost resource efficiency**.

Problem Statement:

Manual and static irrigation leads to:

- Over or under watering.
- Poor yield and water wastage.
- High labor costs.

Existing systems lack:

- Real-time prediction.
- Integration of multiple environmental factors.

21°C - 27°C



RESEARCH QUESTIONS

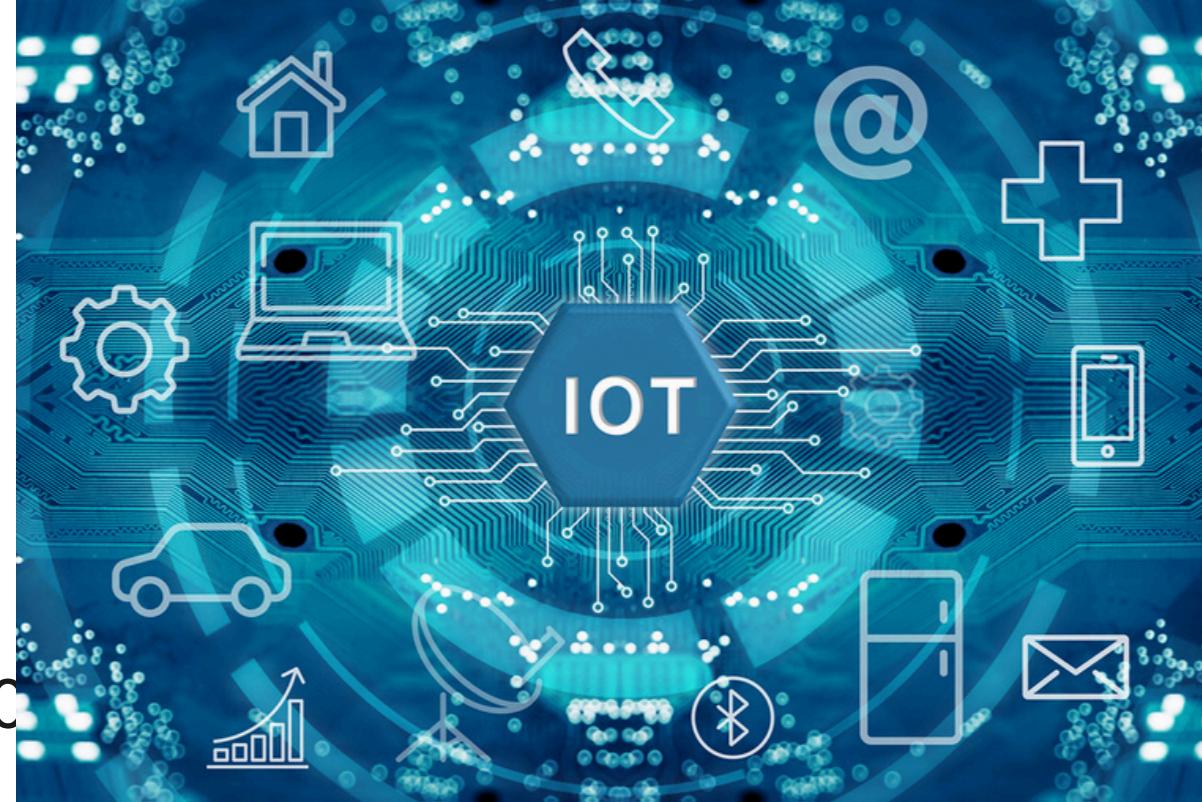
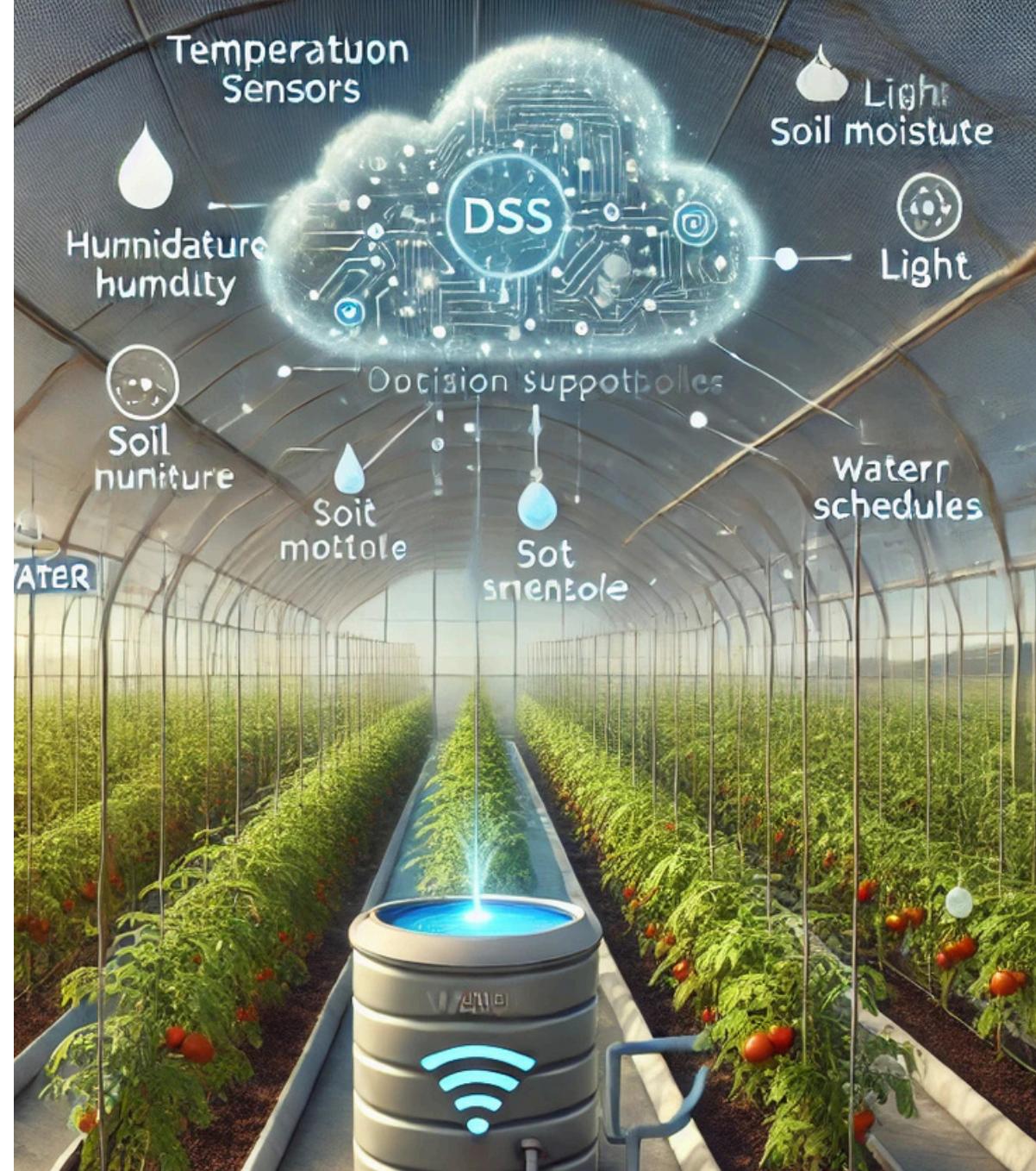
How can we develop a decision support system that accurately predicts optimal water requirements and ensures water availability based on real-time environmental conditions?



IT21231896

ASARDEEN .A

24-25J-C



RESEARCH GAP

Existing Systems / Research	Environmental Monitoring	Water Availability Monitoring & Control	Machine learning integration	Overflow Detection	Water recruitment prediction	Automation Features & Control	Real time dashboard
IoT-Based Precision Irrigation System	✓	✓	✗	✗	✗	✓	✗
Smart Tomato Cultivation System	✓	✓	✗	✗	✗	✓	✓
Indoor Seed Germination System	✓	✗	✗	✗	✗	✗	✗
LoRaWAN-Based IoT Monitoring	✓	✗	✗	✗	✗	✗	✓
Proposed System (This Study)	✓	✓	✓	✓	✓	✓	✓

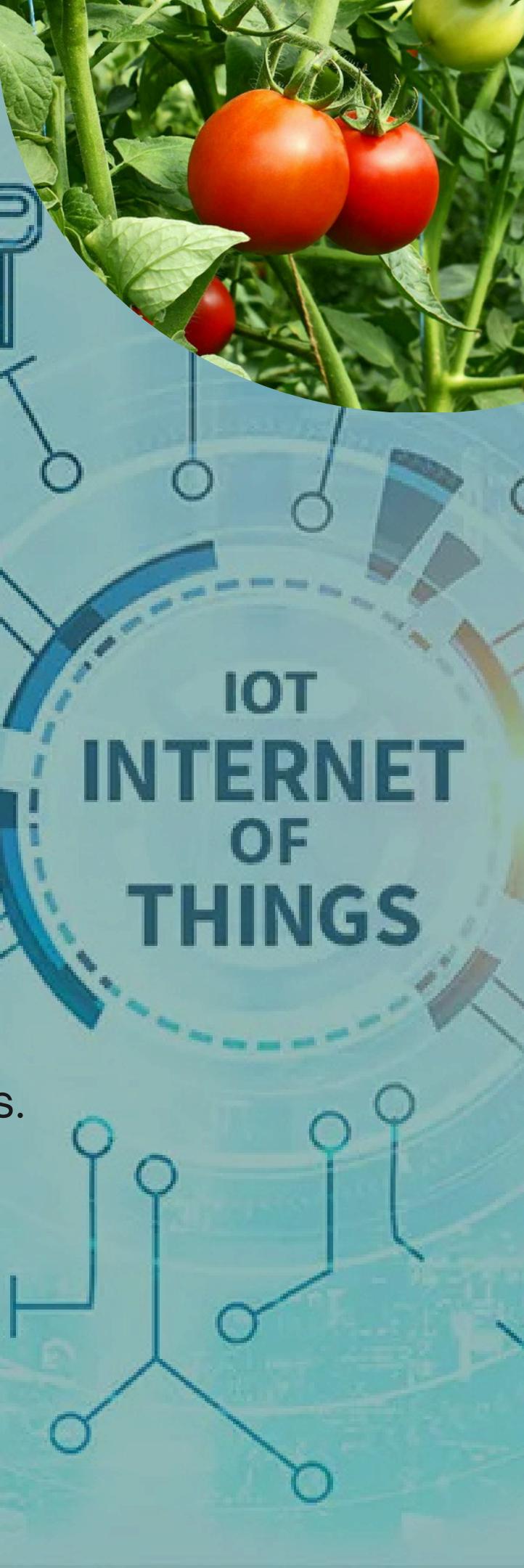
MAIN & SPECIFIC OBJECTS

Main objectives

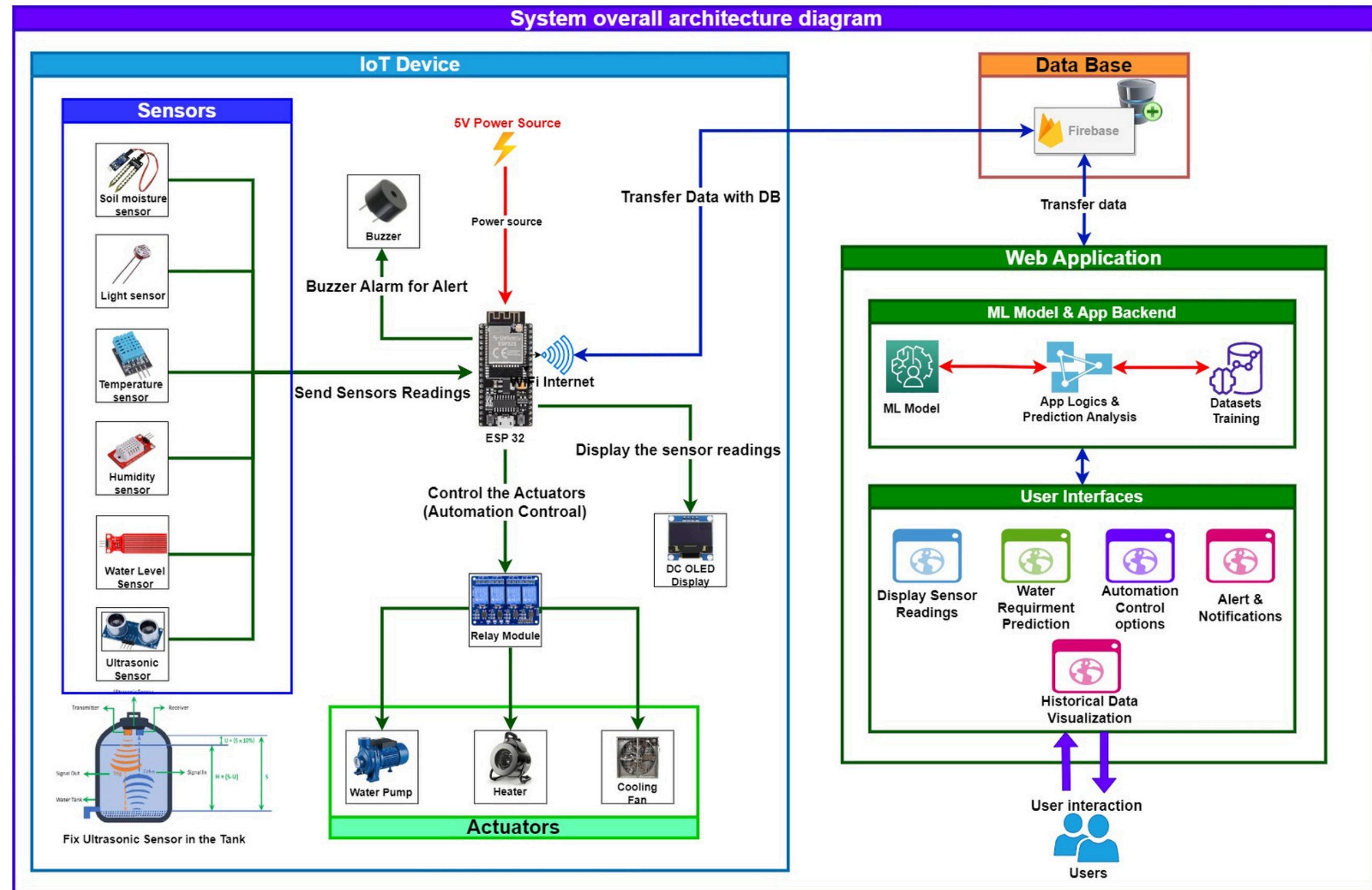
To design a smart, IoT-based decision support system that utilizes machine learning to predict precise watering needs in tomato greenhouses, automate environmental controls, and provide real-time monitoring, control, and historical analysis.

Sub objectives

-  **Develop ML Models to Predict Irrigation Needs**
Utilize environmental data and ML models to predict water required per plant.
-  **Develop IoT Device with Integrated Real-Time Sensors**
 - ▶ Capture live temperature, humidity, light, soil moisture, and water level data via Firebase
 - ▶ Monitor real time water availability using an ultrasonic sensor.
-  **Enable Automated Controls**
Control irrigation, fans, Ventilation and alarms, based on live conditions and logic thresholds.
-  **Data Logging & Visualization**
Maintain and display historical water predictions using charts and tables.
-  **User-Friendly & Responsive Web Dashboard**
Provide authenticated access for users to view, control, and download system data.



METHODOLOGY - DIAGRAM



Sensors → ESP32 → Firebase → ML model → Web Dashboard



100% COMPLETION OF THE PROJECT

🧠 ML Model Development,
Trained & Validation



📡 IoT Device Development &
Firebase Integration



🌐 Web Application with
Real-Time Dashboard



⚙ Automation Features
Implementation



📈 History Logging &
Data Visualization



🔗 Complete System
Integration & Testing



ABOUT THE IOT DEVELOPMENT

IoT Code

```
// Pin definitions for sensors
#define DHTPIN 15
#define DHTTYPE DHT11
#define TRIG_PIN 27
#define ECHO_PIN 18
#define SIGNAL_PIN 34
#define LIGHT_SENSOR_PIN 33

// OLED display parameters
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
#define OLED_I2C_ADDRESS 0x3C

// Initialize DHT sensor and OLED
DHT dht(DHTPIN, DHTTYPE);
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &OLED_I2C_ADDRESS);

// Read DHT sensor data
float temperature = dht.readTemperature();
float humidity = dht.readHumidity();

// Read ultrasonic sensor
digitalWrite(TRIG_PIN, LOW);
delayMicroseconds(2);
digitalWrite(TRIG_PIN, HIGH);
delayMicroseconds(10);
digitalWrite(TRIG_PIN, LOW);
long duration = pulseIn(ECHO_PIN, HIGH);
float distance = (duration * 0.034) / 2;

// Read soil moisture sensor
int soilMoisture = analogRead(SIGNAL_PIN);

// Read light sensor
int lightLevel = analogRead(LIGHT_SENSOR_PIN);

// Display data on OLED
display.clearDisplay();
display.setCursor(0, 0);
display.setTextSize(1);
display.setTextColor(SSD1306_WHITE);
display.print("Temp: ");
display.print(temperature);
display.println(" C");
display.print("Hum: ");
display.print(humidity);
display.println(" %");
display.print("Dist: ");
display.print(distance);
display.println(" cm");
display.print("Soil: ");
display.print(soilMoisture);
display.println();
display.print("Light: ");
display.print(lightLevel);
display.display();

// Log data to serial
Serial.print("Temperature: ");
Serial.print(temperature);
Serial.println("°C");
Serial.print("Humidity: ");
Serial.print(humidity);
Serial.println("%");
Serial.print("Distance: ");
Serial.print(distance);
Serial.println(" cm");
Serial.print("Soil Moisture: ");
Serial.print(soilMoisture);
Serial.println();
Serial.print("Light Level: ");
Serial.println(lightLevel);
```

IoT Firebase Configurations

```
// Initialize Firebase
config.api_key = API_KEY;
config.database_url = DATABASE_URL;

if (Firebase.signup(&config, &auth, "", "")) {
    Serial.println("Firebase authentication succeeded");
    signupOK = true;
} else {
    Serial.printf("Firebase signup failed: %s\n", config.signer.s);
}
Firebase.begin(&config, &auth);
```

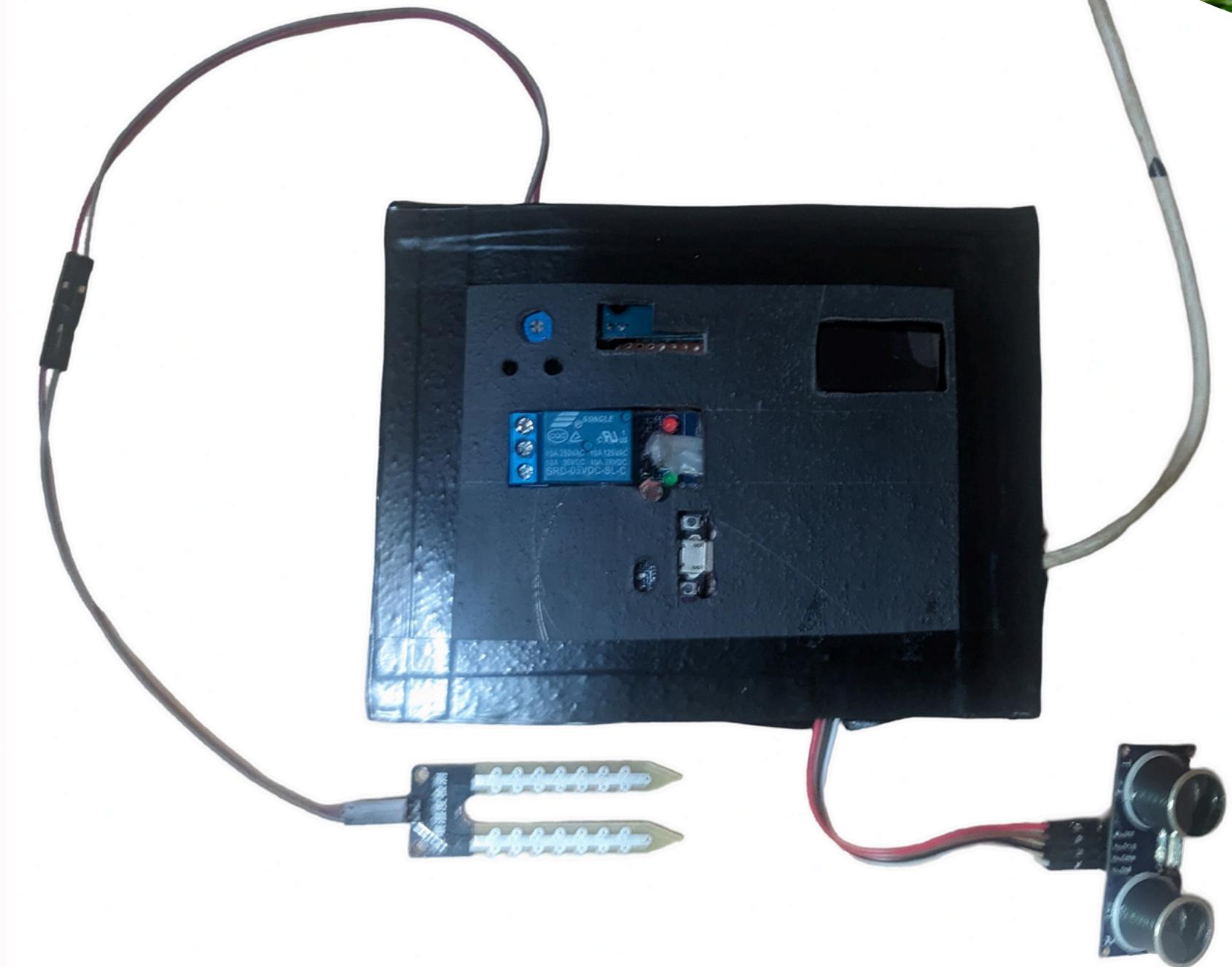
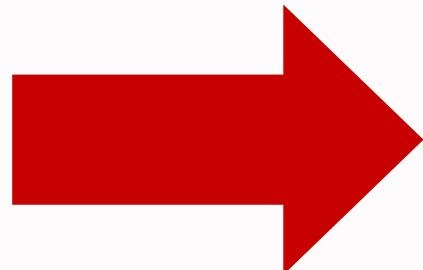
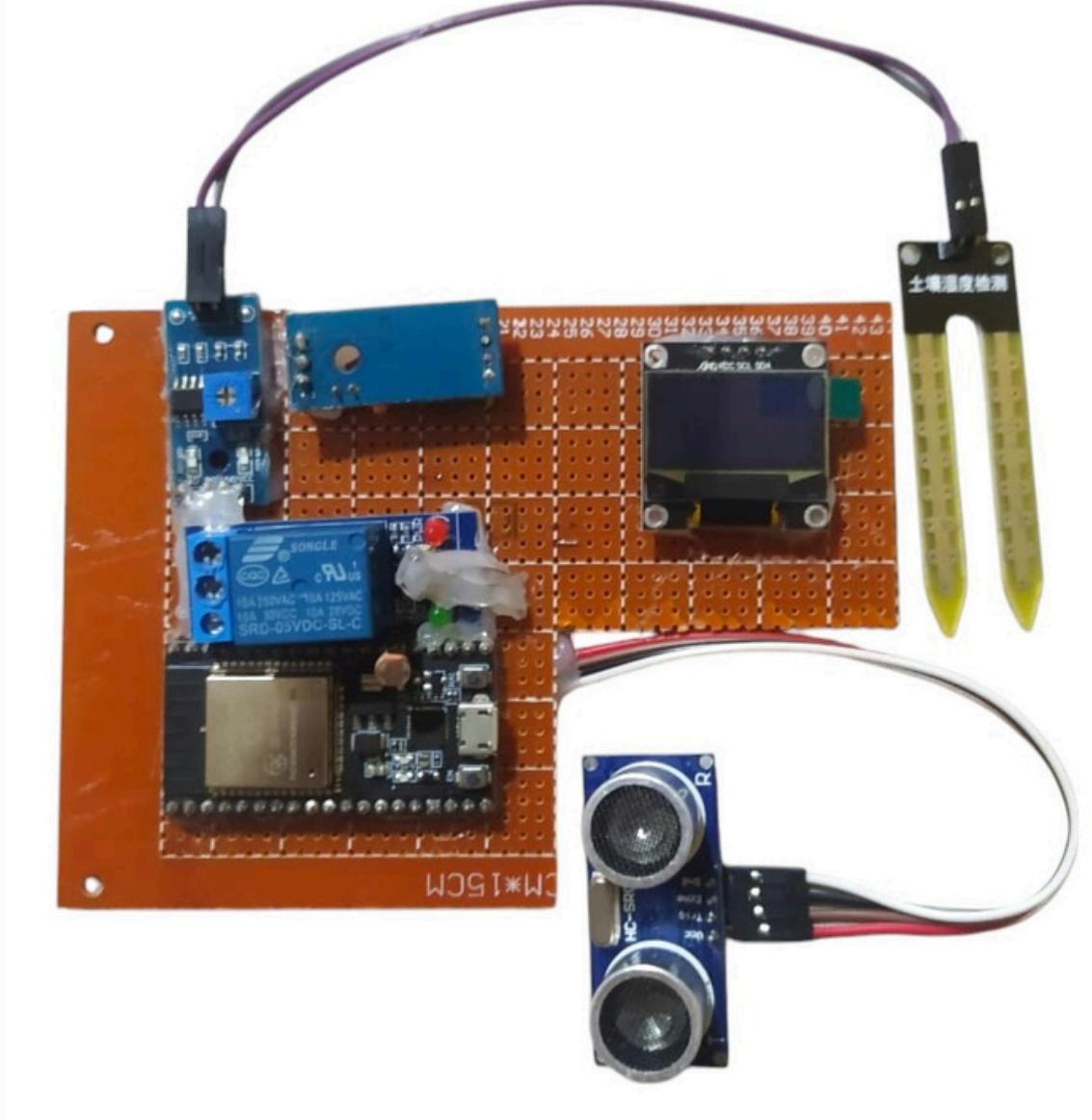
```
// Upload data to Firebase
if (Firebase.ready() && signupOK && millis() - sendDataPrevMillis > 2000) {
    sendDataPrevMillis = millis();

    Firebase.RTDB.setFloat(&fbdo, "/sensors/temperature", temperature);
    Firebase.RTDB.setFloat(&fbdo, "/sensors/humidity", humidity);
    Firebase.RTDB.setFloat(&fbdo, "/sensors/distance", distance);
    Firebase.RTDB.setInt(&fbdo, "/sensors/soilMoisture", soilMoisture);
    Firebase.RTDB.setInt(&fbdo, "/sensors/lightLevel", lightLevel);

    if (fbdo.httpCode() != 200) {
        Serial.print("Firebase failed: ");
        Serial.println(fbdo.errorReason());
    } else {
        Serial.println("Data uploaded to Firebase");
    }
}
```

IOT DEVICE

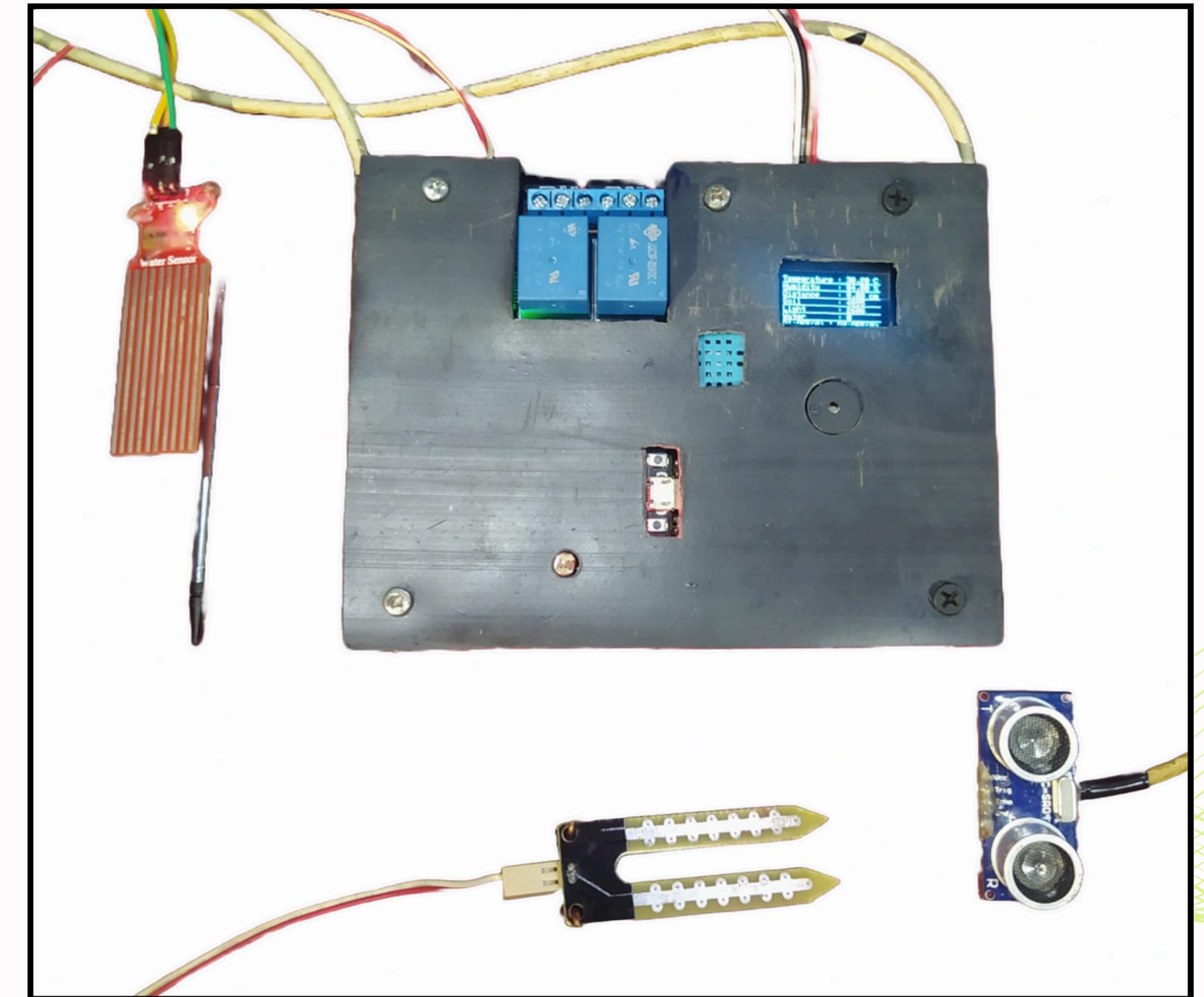
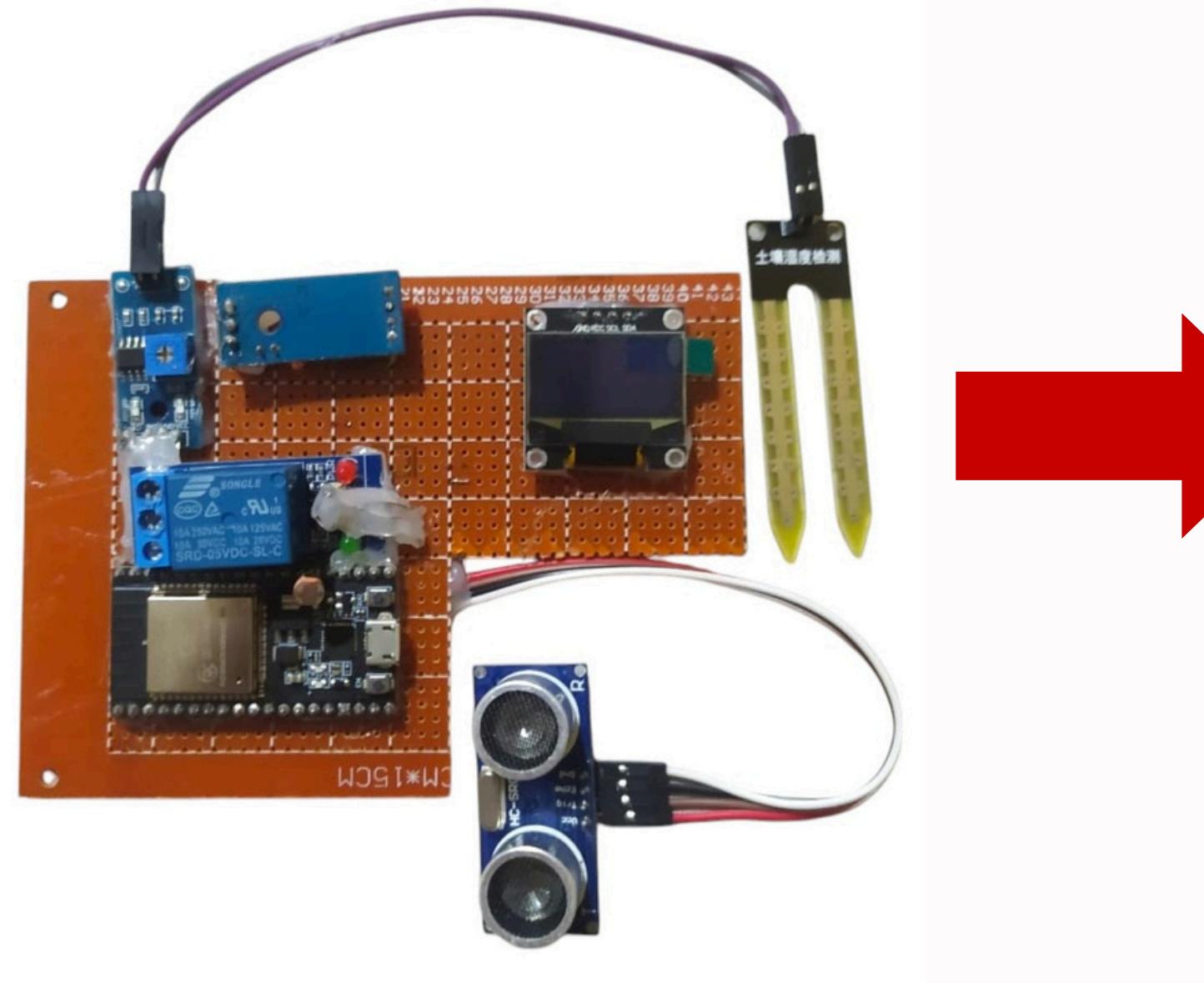
Previous version of the IoT device



IOT DEVICE

Updated Version of The IoT

IoT device being successfully developed and covered with a casing for safety.



ABOUT THE DATASET

The dataset captures real-time environmental factors affecting tomato cultivation, including Timestamp, Temperature (°C), Humidity (%), Soil Moisture (%), Light Level (lux), and Water Needed (Liters), to optimize watering schedules in the Smart Greenhouse Decision Support System.

Timestamp	Temperature (°C)	Humidity (%)	Soil Moisture (%)	Light Level (lux)	Water Level Needed (Liters)
11/1/2024 8:00	27.5	69.6	10.5	903	4.8
11/1/2024 9:00	39	69.5	36.5	1696	11.2
11/1/2024 10:00	34.6	34.6	17.1	966	6.9
11/1/2024 11:00	32	54.7	48.4	1183	12.9
11/1/2024 11:00	23.1	32.9	15.9	517	5.5
11/1/2024 13:00	23.1	57.5	26.6	609	7.6
11/1/2024 14:00	21.2	52.1	13.4	1089	4.8
11/1/2024 15:00	37.3	74.4	49.9	1220	13.7
11/1/2024 16:00	32	47.5	30.1	1400	9.2
11/1/2024 17:00	34.2	35.9	33.8	937	10.2
11/1/2024 18:00	20.4	37.1	12.7	1542	4.6
11/1/2024 19:00	39.4	68.1	40	1790	11.9
11/1/2024 20:00	36.6	60.9	18.4	1670	7.3
11/1/2024 21:00	24.2	35.1	45.9	559	11.6
11/1/2024 22:00	23.6	34.2	18.2	1221	6
11/1/2024 23:00	23.7	65	17.6	657	5.9
11/2/2024 0:00	26.1	33.6	11.5	863	4.9
11/2/2024 1:00	30.5	71.1	28.9	1980	8.8

DATA PREPROCESSING & FEATURE ENGINEERING

```
15  
16     # Preprocess the data  
17     data['Timestamp'] = pd.to_datetime(data['Timestamp'])  
18  
19     # Feature Engineering  
20     data['Hour'] = data['Timestamp'].dt.hour  
21     data['DayOfWeek'] = data['Timestamp'].dt.dayofweek  
22     data['Month'] = data['Timestamp'].dt.month  
23     data['Year'] = data['Timestamp'].dt.year  
24
```

```
2     # Load dataset  
3     file_path = "irrigation_dataset.csv"  
4     data = pd.read_csv(file_path)  
5  
6     # Preprocess the data  
7     data['Timestamp'] = pd.to_datetime(data['Timestamp'])  
8  
9     # Feature Engineering  
10    data['Hour'] = data['Timestamp'].dt.hour  
11    data['DayOfWeek'] = data['Timestamp'].dt.dayofweek  
12    data['Month'] = data['Timestamp'].dt.month  
13    data['Year'] = data['Timestamp'].dt.year  
14  
15    # Handle missing values  
16    imputer = SimpleImputer(strategy='median')  
17    data[['Temperature (°C)', 'Humidity (%)', 'Soil Moisture (%)', 'Light Level (lux)']] = imputer.fit_transform(  
18        data[['Temperature (°C)', 'Humidity (%)', 'Soil Moisture (%)', 'Light Level (lux)']])  
19
```

```
61  
62     # Feature Scaling and Polynomial Expansion  
63     # Feature Scaling  
64     scaler = StandardScaler()  
65     features = ['Temperature (°C)', 'Humidity (%)', 'Soil Moisture (%)', 'Light Lev  
66     data[features] = scaler.fit_transform(data[features])  
67  
68     # Polynomial Feature Expansion  
69     poly = PolynomialFeatures(degree=2, include_bias=False)  
70     X_poly = poly.fit_transform(data[features])  
71
```

```
  Data columns (total 10 columns):  
   #   Column           Non-Null Count  Dtype     
   --  --     
   0   Timestamp       420 non-null    datetime64[ns]  
   1   Temperature (°C) 420 non-null    float64  
   2   Humidity (%)    420 non-null    float64  
   3   Soil Moisture (%) 420 non-null    float64  
   4   Light Level (lux) 420 non-null    float64  
   5   Water Level Needed (Liters) 420 non-null    float64  
   6   Hour            420 non-null    int64  
   7   DayOfWeek       420 non-null    int64  
   8   Month           420 non-null    int64  
   9   Year            420 non-null    int64  
   dtypes: datetime64[ns](1), float64(5), int64(4)  
   memory usage: 32.9 KB
```

MODEL TRAINING

```
1 # Split data into training
2 X = X_poly
3 y = data['Water Level Needed (Liters)']
4 X_train, X_temp, y_train, y_temp = train_test_split(*arrays: X, y, test_size=0.3, random_state=42)
5 X_val, X_test, y_val, y_test = train_test_split(*arrays: X_temp, y_temp, test_size=0.5, random_state=42)
6
7 # Hyperparameter Tuning
8 param_grid = {
9     'n_estimators': [100, 200, 300],
10    'max_depth': [10, 20, 30, None],
11    'min_samples_split': [2, 5, 10],
12    'min_samples_leaf': [1, 2, 4],
13    'bootstrap': [True, False]
14 }
15
16 grid_search = GridSearchCV(estimator=RandomForestRegressor(random_state=42),
17                             param_grid=param_grid,
18                             cv=5,
19                             n_jobs=-1,
20                             verbose=2,
21                             scoring='neg_mean_squared_error')
22
23 grid_search.fit(X_train, y_train)
```

```
# Best parameters
print("Best Hyperparameters found:", grid_search.best_params_)

# Train the model
best_model = grid_search.best_estimator_
y_val_pred = best_model.predict(X_val)

# performance metrics
val_rmse = mean_squared_error(y_val, y_val_pred, squared=False)
val_r2 = r2_score(y_val, y_val_pred)
val_mae = mean_absolute_error(y_val, y_val_pred)

print(f"Validation RMSE: {val_rmse:.2f}")
print(f"Validation R2 Score: {val_r2:.2f}")
print(f"Validation MAE: {val_mae:.2f}")
💡
```

```
134
135 # Save Models
136 joblib.dump(best_model, filename="irrigation_model.pkl")
137 joblib.dump(scaler, filename="scaler.pkl")
138 joblib.dump(poly, filename="poly.pkl")
139 print("Best model, scaler, and poly transformer saved successfully.")
140 💡
```

PERFORMANCE MATRIX ACCURECY :

```
Validation RMSE: 0.19
Validation R2 Score: 0.99
Validation MAE: 0.15
```

MODEL SELECTION & ACCURACY COMPARISON

🧪 Models Tested:

1. ✗ Linear Regression
 - ▶ Very simple, but low accuracy
 - ▶ R² Score: ~0.61
2. ✗ Decision Tree Regression
 - ▶ Showed overfitting on training data
 - ▶ Poor generalization

✓ Final Model Chosen: **Polynomial Regression**

- ✓ Captures non-linear patterns between environment and water needs
- ✓ Well-suited for agriculture sensor data
- ✓ Best accuracy and real-time performance

📈 Final Validation Results:

- ◆ R² Score: 0.99
- ◆ RMSE: 0.19
- ◆ MAE: 0.15

PERFORMANCE MATRIX ACCURACY :

⬇️	Validation RMSE: 0.19
🖨️	Validation R2 Score: 0.99
🗑️	Validation MAE: 0.15

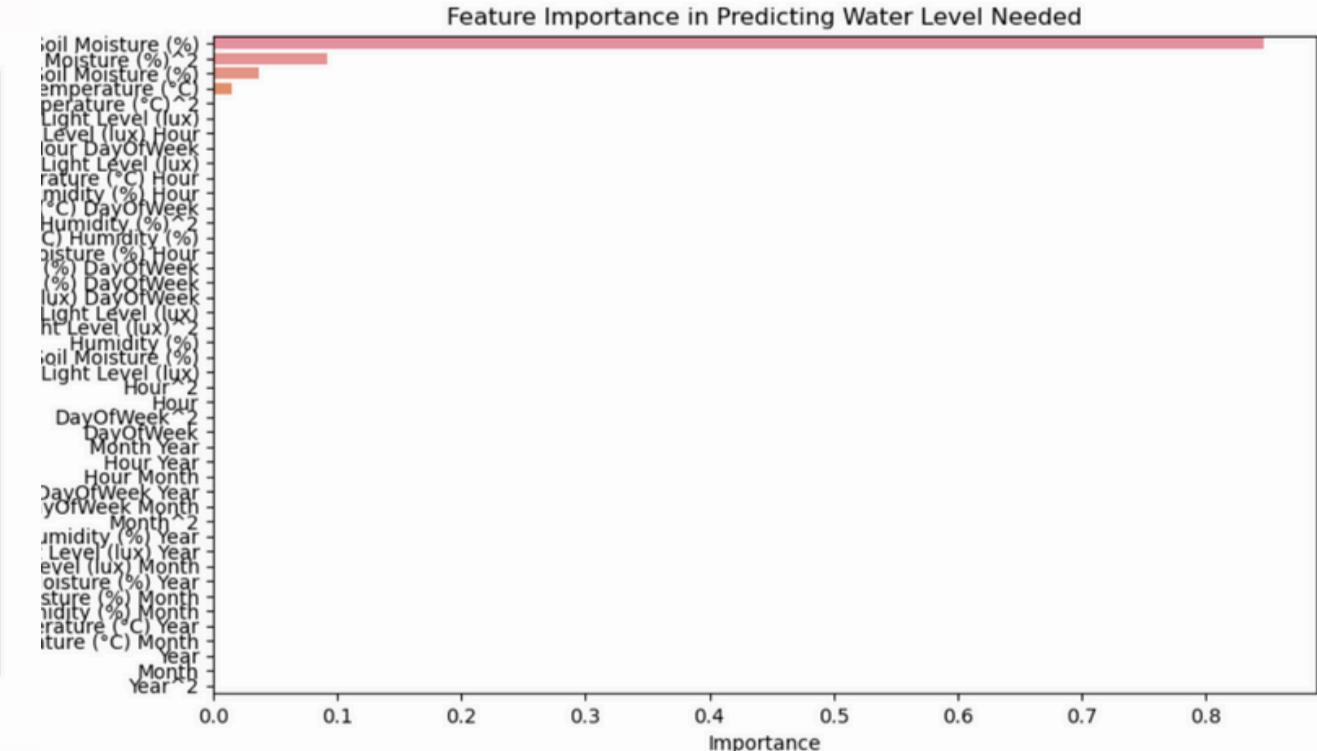
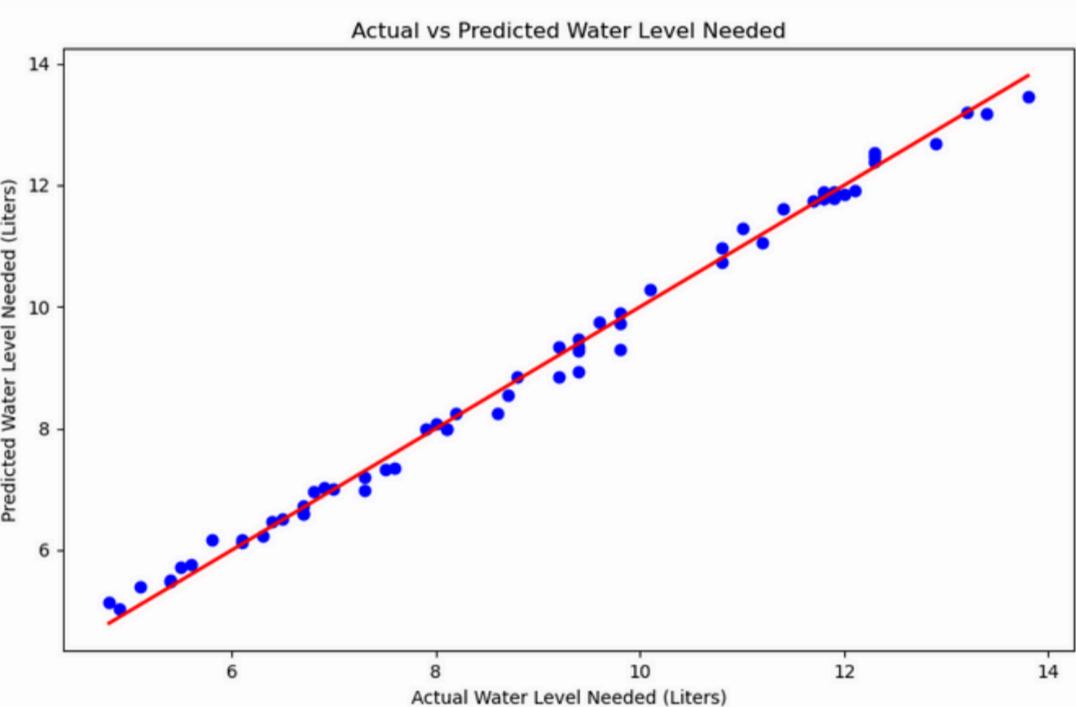
- Polynomial Regression offered the perfect balance between accuracy, speed, and simplicity for real-time irrigation prediction.

MODEL PERFORMANCE EVALUATION

```
#plot the model Performance
plt.figure(figsize=(10, 6))
plt.scatter(y_val, y_val_pred, color='blue')
plt.plot(*args: [min(y_val), max(y_val)], [min(y_val), max(y_val)], color='red')
plt.title('Actual vs Predicted Water Level Needed')
plt.xlabel('Actual Water Level Needed (Liters)')
plt.ylabel('Predicted Water Level Needed (Liters)')
plt.show()

# Find the Important feature(x data) for water need prediction
feature_importances = best_model.feature_importances_
features_list = poly.get_feature_names_out(features)
importance_df = pd.DataFrame({
    'Feature': features_list,
    'Importance': feature_importances
})
importance_df = importance_df.sort_values(by='Importance', ascending=False)

plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=importance_df)
plt.title('Feature Importance in Predicting Water Level Needed')
plt.show()
```



```
# predict watering needs

def predict_watering(temperature, humidity, soil_moisture, light_level, timestamp): 1 usage
    timestamp = pd.to_datetime(timestamp)
    hour = timestamp.hour
    day_of_week = timestamp.dayofweek
    month = timestamp.month
    year = timestamp.year

    # DataFrame input
    input_data = pd.DataFrame([temperature, humidity, soil_moisture, light_level, hour, day_of_week,
                               columns=['Temperature (°C)', 'Humidity (%)', 'Soil Moisture (%)', 'Light Level (%)', 'Hour', 'Day of Week']])

    # Normalize the input
    input_data_scaled = scaler.transform(input_data)

    # Apply polynomial
    input_data_poly = poly.transform(input_data_scaled)

    # prediction
    predicted_water_level = model.predict(input_data_poly)

    return predicted_water_level[0]
```

```
# prediction
predicted_water_level = model.predict(input_data_poly)

return predicted_water_level[0]

# data live
temperature = 25.5
humidity = 65.0
soil_moisture = 25.3
light_level = 1500
timestamp = "2024-11-01 10:00:00"

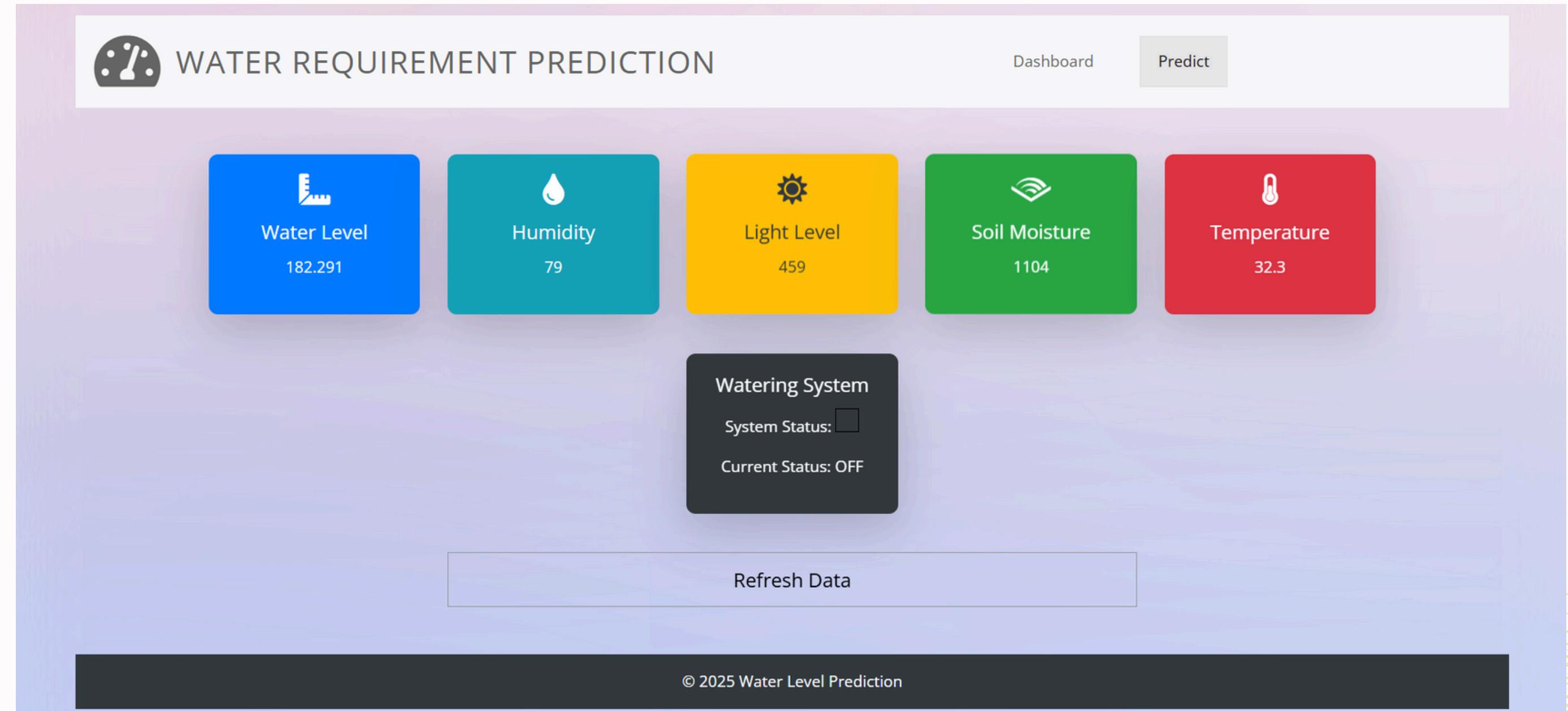
predicted_water_level = predict_watering(temperature, humidity, soil_moisture, light_level)
print(f"Predicted Water Level Needed: {predicted_water_level:.2f} Liters")
```

```
Run Preidct_watering ×

C:\Users\moham\Downloads\Compressed\Python38_01
C:\Users\moham\Downloads\Compressed\Python38_01
    warnings.warn(
Predicted Water Level Needed: 7.62 Liters
Process finished with exit code 0
```

REAL-TIME MONITORING DASHBOARD

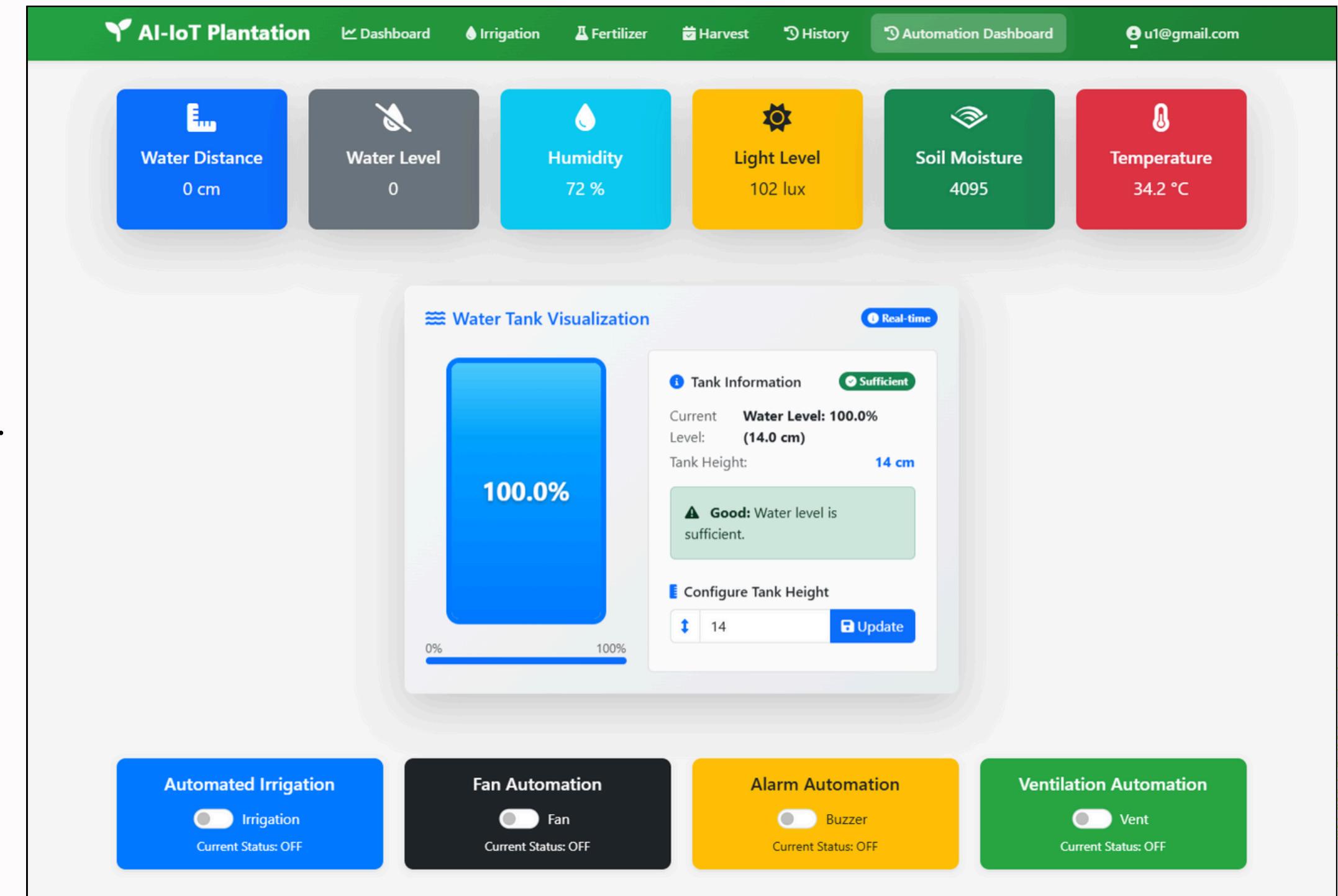
My previous version of
the reading UI



REAL-TIME MONITORING & CONTROL DASHBOARD

- Water Tank Visualization - configurable height and level alerts
- Automation Controls for irrigation, fan, ventilation and alarm systems
- Live Alerts based on threshold conditions (e.g. low water, high heat)
- Real-Time Updates via Firebase

↗ This UI Helps users monitor, control greenhouse conditions effectively.



REAL-TIME AUTOMATION CONTROLS

Automated Irrigation Control

Problem in Existing Systems:

- Traditional irrigation systems rely only on **soil moisture** or **time-based triggers**, often leading to **over-irrigation** and **water wastage**.

Proposed Enhancement :

- The system automatically stops irrigation when **overflow is detected**, preventing **water wastage** and **root diseases** in tomato plants.

Trigger Conditions:

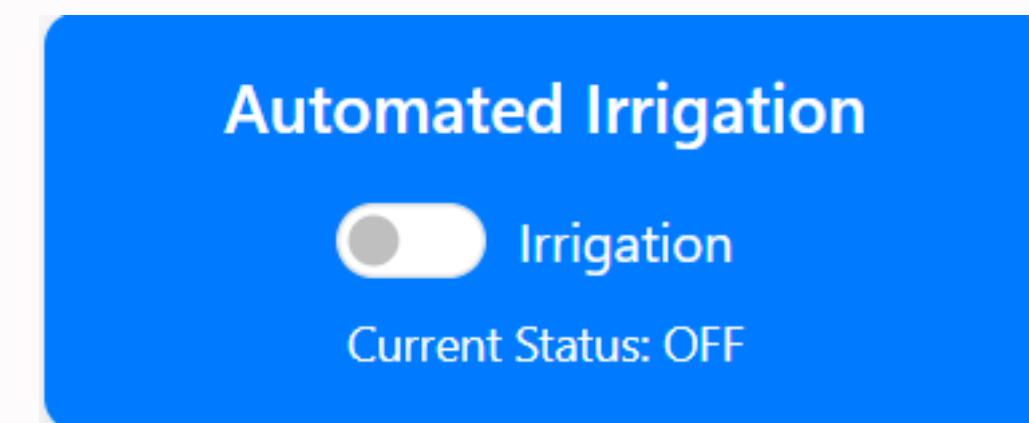
- When $\text{SoilMoisture} > 2500 \rightarrow \text{Soil moisture is dry}$
- $\text{Tank} > 10\% \rightarrow \text{Tank water availability}$
- $\text{WaterLevel} \leq 1000 \rightarrow \text{Water overflow detected}$

System Benefits:

- ✓ Prevents overwatering & diseases
- ✓ Saves water & electricity



Soil Moisture Sensor Value Range:
0 (wet) to ~4095 (dry)



REAL-TIME AUTOMATION CONTROLS

Smart Climate Ventilation Control

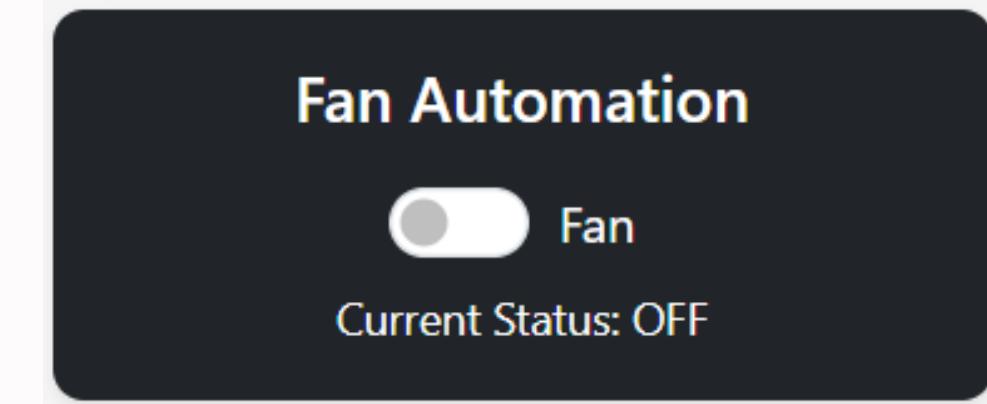


Trigger Conditions:

- Humidity > 70 or
- Temperature > 35°C

Benefits:

- Prevents plant diseases caused by Humidity (Ex: **fungal risk** - Botrytis, Powdery Mildew)
- Control temperature for optimal plant health
- Prevents entry of **pests, dust, and airborne diseases**

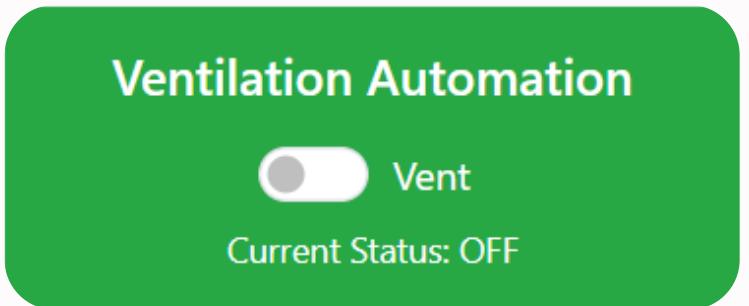


REAL-TIME AUTOMATION CONTROLS

Smart Servo-Based Ventilation Control

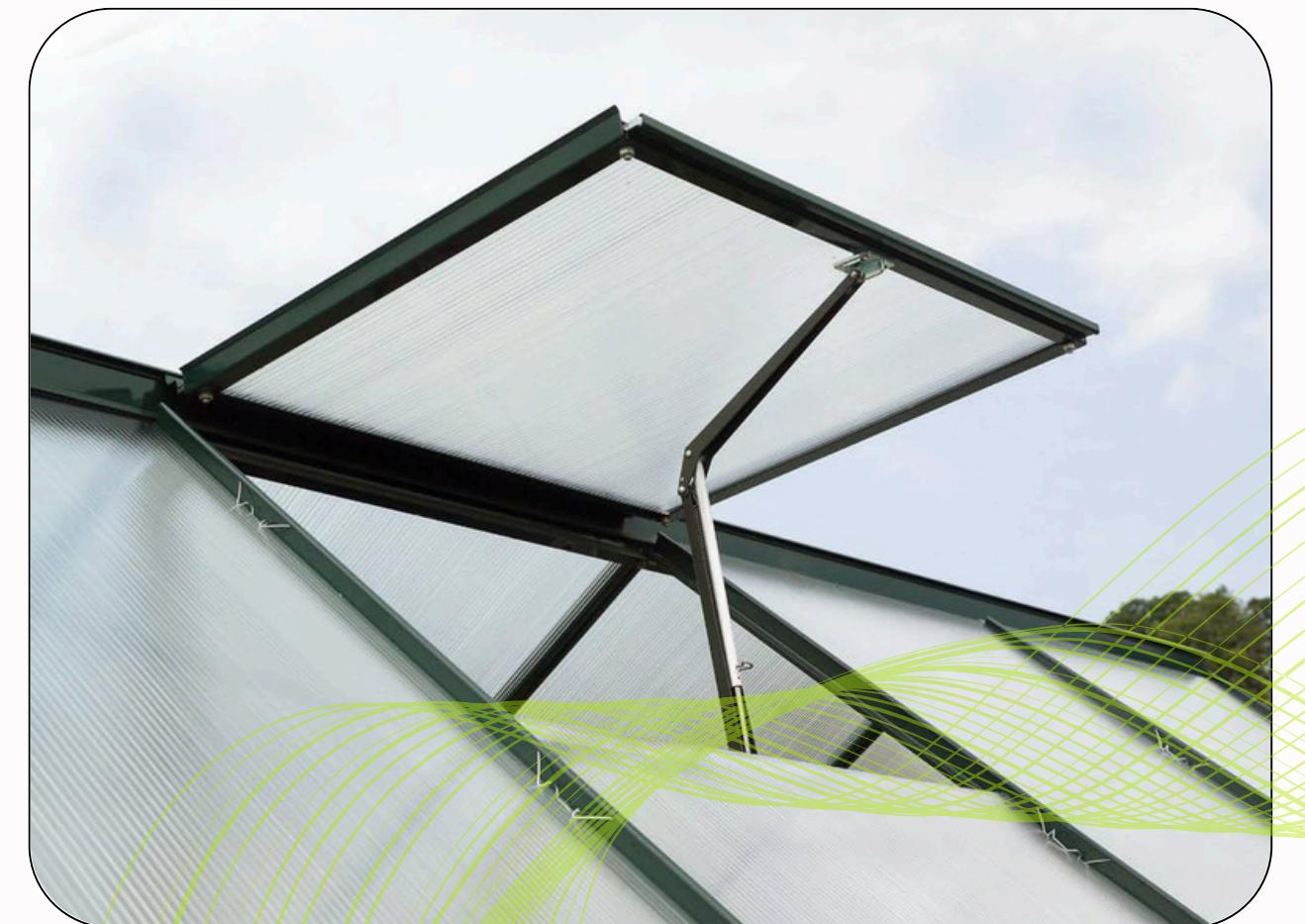
🔧 Trigger Condition:

- 🌡️ Temperature > 35°C or Humidity > 70 → Ventilation windows automatically open



🌿 Benefits:

- Reduces **heat stress** by enabling natural air circulation.
- Maintains ideal temperature for **photosynthesis** and growth.
- **Low energy** use – Minimizing energy use compared to fans.
- Best when **external conditions are safe (pest, birds, airborne diseases like fungal)**
- Improves plant **CO₂ exchange**



REAL-TIME AUTOMATION CONTROLS

Critical Alert System – Buzzer Automation

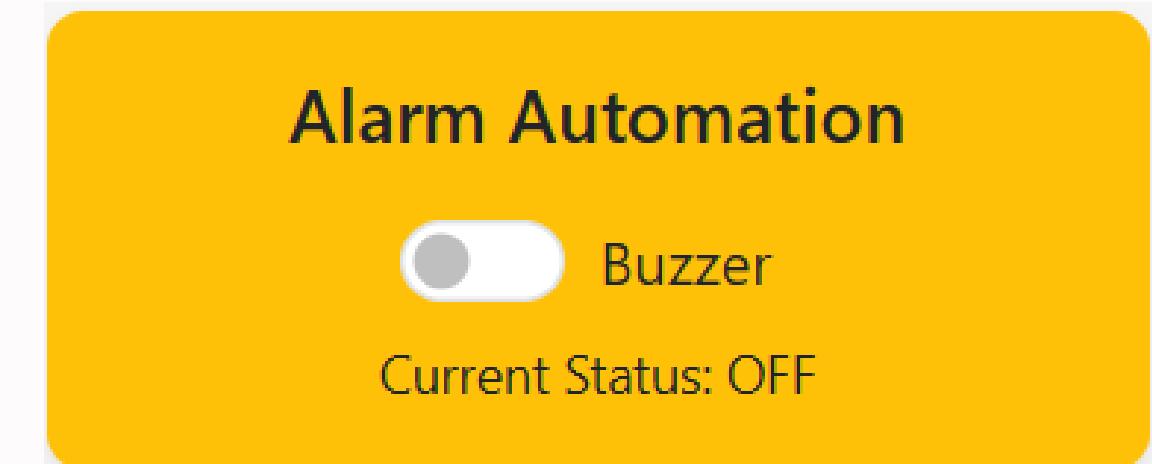
Triggers (any one):

- Water Level > 1000 → **Water Overflow**
- Tank Level < 10% → **Water Availability is low**
- Temperature > 40°C → **Over Temperature**
- Soil Moisture > 2500 → **When the Soil Moisture dry**

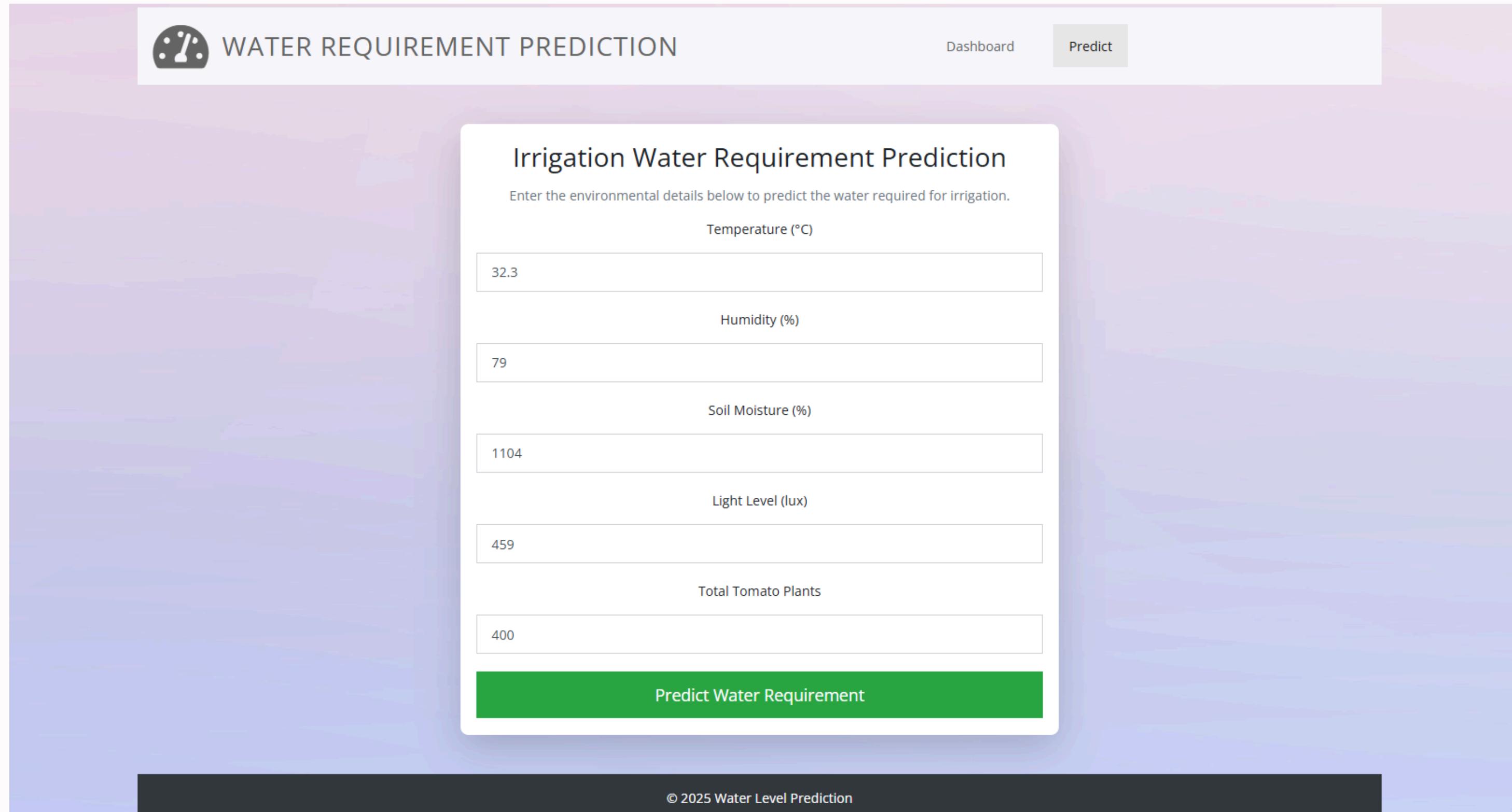


Benefits of this Automation System:

- Immediate **physical alert** to critical risks
- Reduces reliance on dashboard monitoring
- **Failsafe safety Feature** if web access or internet is down
- Fully automated



WATER REQUIREMENT PREDICTION UI



The image shows a user interface for water requirement prediction. At the top, there is a navigation bar with a water drop icon, the text "WATER REQUIREMENT PREDICTION", and two buttons: "Dashboard" and "Predict". The main area is titled "Irrigation Water Requirement Prediction" and contains a form for entering environmental details. The fields are: Temperature (°C) with value 32.3, Humidity (%) with value 79, Soil Moisture (%) with value 1104, Light Level (lux) with value 459, and Total Tomato Plants with value 400. A green button at the bottom of the form says "Predict Water Requirement". At the bottom of the page, there is a footer bar with the text "© 2025 Water Level Prediction".

WATER REQUIREMENT PREDICTION

Dashboard Predict

Irrigation Water Requirement Prediction

Enter the environmental details below to predict the water required for irrigation.

Temperature (°C)

32.3

Humidity (%)

79

Soil Moisture (%)

1104

Light Level (lux)

459

Total Tomato Plants

400

Predict Water Requirement

© 2025 Water Level Prediction

WATER REQUIREMENT PREDICTION UI

User Inputs:

- Temperature (°C), Humidity (%), Soil Moisture (%), Light Level (lux)
- Plant Count: Can input in two methods
 - Direct input the Plant Counts
 - Auto-calculate from greenhouse area & spacing

Outputs:

- Per Plant Water Need (L)
- Total Plant Count (if 2nd method selected)
- Total Water Requirement (L)
- Prediction Reason (e.g., high temperature, dry soil)

Irrigation Water Requirement Prediction

Enter the environmental details below to predict the water required for irrigation.

Temperature (°C)	Humidity (%)
35.2	75
Soil Moisture (%)	Light Level (lux)
4095	0

Plant Count Calculation Method

Direct Plant Count
Enter exact number of plants

Calculate from Area
Based on greenhouse size

Total Tomato Plants: 400 plants

Calculate Water Requirement

WATER REQUIREMENT PREDICTION UI

Introduced a new method called ‘Calculation From Area’ for total plant count calculation based on the greenhouse’s area (square feet) and plant gap.

sample output:

if area=1000 and the Gap=1.5 → plant counts=444



Irrigation Water Requirement Prediction

Enter the environmental details below to predict the water required for irrigation.

Temperature (°C)	Humidity (%)
30.8	91.0
Soil Moisture (%)	Light Level (lux)
4095.0	327.0

Plant Count Calculation Method

Direct Plant Count
Enter exact number of plants

Calculate from Area
Based on greenhouse size

Greenhouse Area m² **Plant Spacing**

System will calculate plant count automatically based on area and spacing

Calculate Water Requirement

Per Plant Water Need **1.58** Liters **Total Plant Count** **444** plants

Total Water Requirement For All Plants **701.77** Liters

WATER REQUIREMENT PREDICTION OUTPUT

 Calculate Water Requirement

Per Plant Water Need

1.58

Liters

Total Plant Count

444

plants

Total Water Requirement For All Plants

701.77 Liters

Reason for Prediction

Alert: Elevated temperature detected. Consider providing shade and increasing watering frequency.

Intelligent Plantation System

Harnessing AI & IoT for smarter agriculture

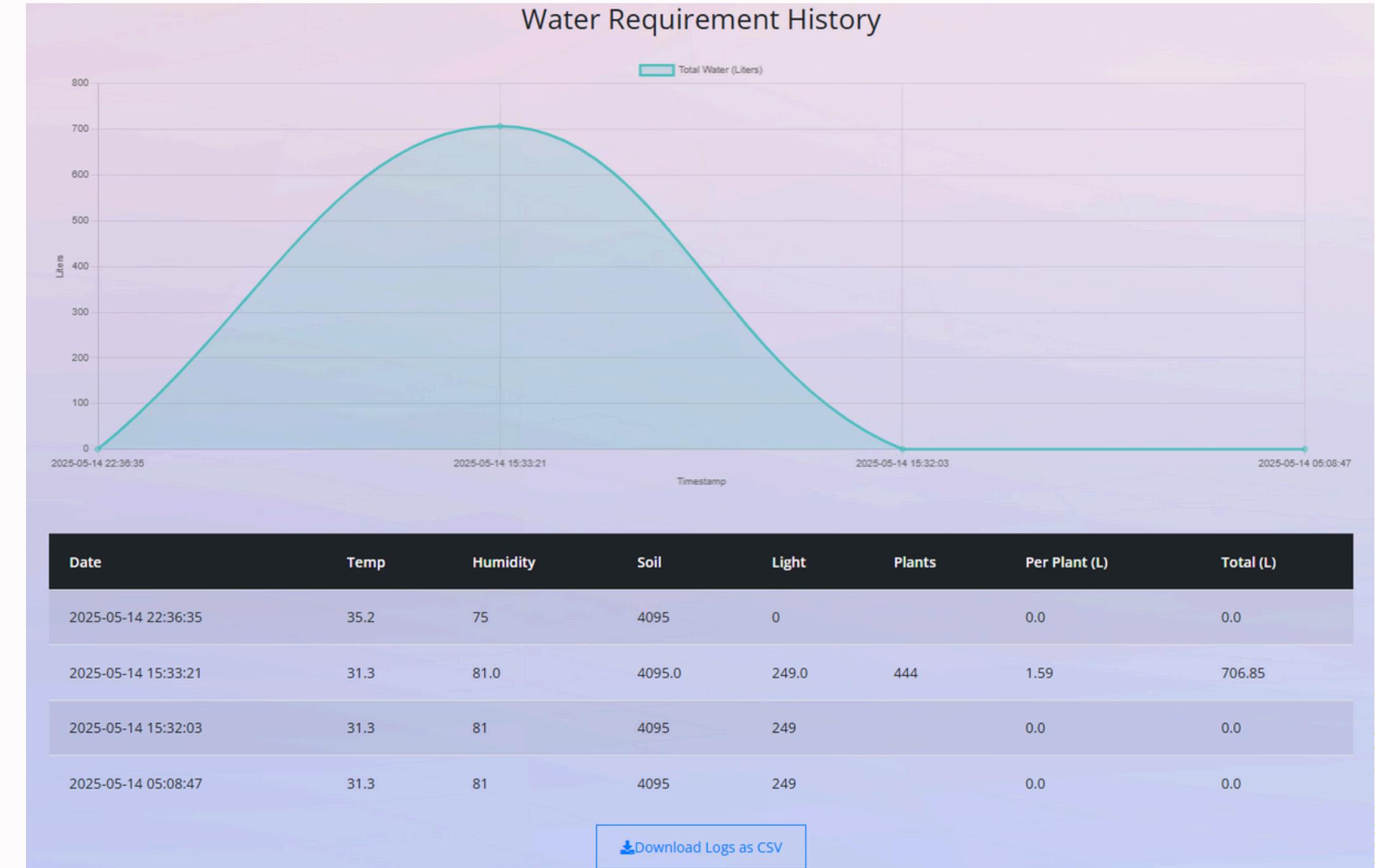
WATER REQUIREMENT HISTORY DASHBOARD UI

Water Requirement History UI

- Logs each prediction with environmental inputs() & water output
- Line chart shows total water usage trends over time
- Table view for detailed record inspection
- [Download as CSV](#) for offline tracking

Benefits:

- Tracks irrigation performance
- Supports data-driven decisions
- Aids in optimizing water usage



REFERENCES

- [1] Tao, W., Zhao, L., Wang, G. & Liang, R. (2021). Review of the internet of things communication technologies in smart agriculture and challenges, *Computers and Electronics in Agriculture*, Vol. 189, 2021, 106352, ISSN0168-1699, <https://doi.org/10.1016/j.compag.2021.106352>.
- [2] Food and Agriculture Organization of the United Nations, Global agriculture towards 2050, https://www.fao.org/fileadmin/templates/wsfs/docs/Issues_papers/HLEF2050_Global_Agriculture.pdf.
- [3] Saggi, M.K. & Jain, S. A. (2022). Survey Towards Decision Support System on Smart Irrigation Scheduling Using Machine Learning approaches. *Archives of Computational Methods in Engineering*, 29, 4455–4478. <https://doi.org/10.1007/s11831-022-09746-3>.
- [4] Salim, O., Fouad, K. & Hassan, B. (2022). Dual-Level Sensor Selection with Adaptive Sensor Recovery to Extend WSNs' Lifetime, *Human-centric Computing and Information Sciences*, Vol. 12, Article number: 18, <https://doi.org/10.22967/HCIS.2022.12.018>.
- [5] Goap, A., Sharma, D., Shukla, A.K. & Krishna, R. (2018). An IoT based smart irrigation management system using Machine learning and open source technologies, *Computers and Electronics in Agriculture*, Vol. 155, PP: 41-49, ISSN0168-1699, <https://doi.org/10.1016/j.compag.2018.09.040>.
- [6] Fouad, K. & Elbably, D. (2020). Intelligent approach for large-scale data mining. *Int. J. Computer Applications in Technology*, Vol. 63, Nos. 1/2, PP: 93-112.
- [7] Fouad, K., Hassan, B. & Salim, O. (2022). Hybrid Sensor Selection Technique for Lifetime Extension of Wireless Sensor Networks. *Computers, Materials & Continua* 2022, 70(3), 4965-4985. <https://doi.org/10.32604/cmc.2022.020926>.

REFERENCES

- [1] Tao, W., Zhao, L., Wang, G. & Liang, R. (2021). Review of the internet of things communication technologies in smart agriculture and challenges, *Computers and Electronics in Agriculture*, Vol. 189, 2021, 106352, ISSN0168-1699, <https://doi.org/10.1016/j.compag.2021.106352>.
- [2] Food and Agriculture Organization of the United Nations, Global agriculture towards 2050, https://www.fao.org/fileadmin/templates/wsfs/docs/Issues_papers/HLEF2050_Global_Agriculture.pdf.
- [3] Saggi, M.K. & Jain, S. A. (2022). Survey Towards Decision Support System on Smart Irrigation Scheduling Using Machine Learning approaches. *Archives of Computational Methods in Engineering*, 29, 4455–4478. <https://doi.org/10.1007/s11831-022-09746-3>.
- [4] Salim, O., Fouad, K. & Hassan, B. (2022). Dual-Level Sensor Selection with Adaptive Sensor Recovery to Extend WSNs' Lifetime, *Human-centric Computing and Information Sciences*, Vol. 12, Article number: 18, <https://doi.org/10.22967/HCIS.2022.12.018>.
- [5] Goap, A., Sharma, D., Shukla, A.K. & Krishna, R. (2018). An IoT based smart irrigation management system using Machine learning and open source technologies, *Computers and Electronics in Agriculture*, Vol. 155, PP: 41-49, ISSN0168-1699, <https://doi.org/10.1016/j.compag.2018.09.040>.
- [6] Fouad, K. & Elbably, D. (2020). Intelligent approach for large-scale data mining. *Int. J. Computer Applications in Technology*, Vol. 63, Nos. 1/2, PP: 93-112.
- [7] Fouad, K., Hassan, B. & Salim, O. (2022). Hybrid Sensor Selection Technique for Lifetime Extension of Wireless Sensor Networks. *Computers, Materials & Continua* 2022, 70(3), 4965-4985. <https://doi.org/10.32604/cmc.2022.020926>.

REFERENCES

- [1] Tao, W., Zhao, L., Wang, G. & Liang, R. (2021). Review of the internet of things communication technologies in smart agriculture and challenges, *Computers and Electronics in Agriculture*, Vol. 189, 2021, 106352, ISSN0168-1699, <https://doi.org/10.1016/j.compag.2021.106352>.
- [2] Food and Agriculture Organization of the United Nations, Global agriculture towards 2050, https://www.fao.org/fileadmin/templates/wsfs/docs/Issues_papers/HLEF2050_Global_Agriculture.pdf.
- [3] Saggi, M.K. & Jain, S. A. (2022). Survey Towards Decision Support System on Smart Irrigation Scheduling Using Machine Learning approaches. *Archives of Computational Methods in Engineering*, 29, 4455–4478. <https://doi.org/10.1007/s11831-022-09746-3>.
- [4] Salim, O., Fouad, K. & Hassan, B. (2022). Dual-Level Sensor Selection with Adaptive Sensor Recovery to Extend WSNs' Lifetime, *Human-centric Computing and Information Sciences*, Vol. 12, Article number: 18, <https://doi.org/10.22967/HCIS.2022.12.018>.
- [5] Goap, A., Sharma, D., Shukla, A.K. & Krishna, R. (2018). An IoT based smart irrigation management system using Machine learning and open source technologies, *Computers and Electronics in Agriculture*, Vol. 155, PP: 41-49, ISSN0168-1699, <https://doi.org/10.1016/j.compag.2018.09.040>.
- [6] Fouad, K. & Elbably, D. (2020). Intelligent approach for large-scale data mining. *Int. J. Computer Applications in Technology*, Vol. 63, Nos. 1/2, PP: 93-112.
- [7] Fouad, K., Hassan, B. & Salim, O. (2022). Hybrid Sensor Selection Technique for Lifetime Extension of Wireless Sensor Networks. *Computers, Materials & Continua* 2022, 70(3), 4965-4985. <https://doi.org/10.32604/cmc.2022.020926>.



IT21181160 | Thrimavithana V. D.

Information technology



INTRODUCTION TO PREDICTING FERTILIZING SCHEDULES

Tomatoes are a vital crop, but their cultivation is challenged by climate change, which affects soil nutrients.

Proper fertilization is essential to ensure healthy growth, but mismanagement can harm plants and the environment.



KEY CHALLENGES

- **Over-Fertilization:** Causes nutrient leaching, pollution, and plant toxicity.
- **Under-Fertilization:** Leads to poor growth and low yields.



ENVIRONMENTAL FACTORS

- **High temperatures** increase nutrient demand but risk imbalances.
- **Low temperatures** slow nutrient uptake, risking deficiencies.

RESEARCH QUESTION

How can we develop a decision support system for tomato greenhouses that optimizes fertilizing schedules and types that ensures nutrient balance based on real-time plant and environmental conditions?

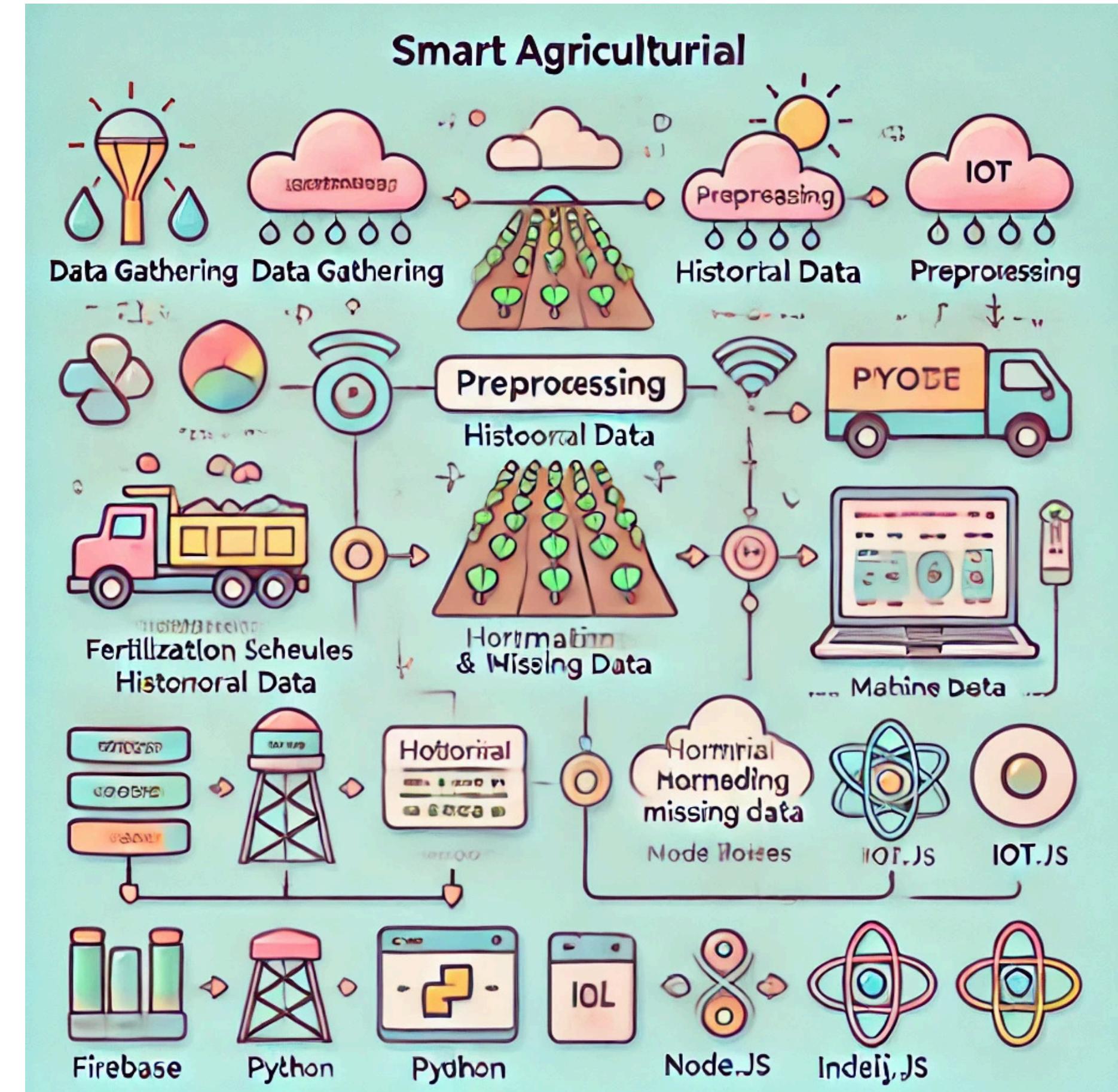


RESEARCH GAP

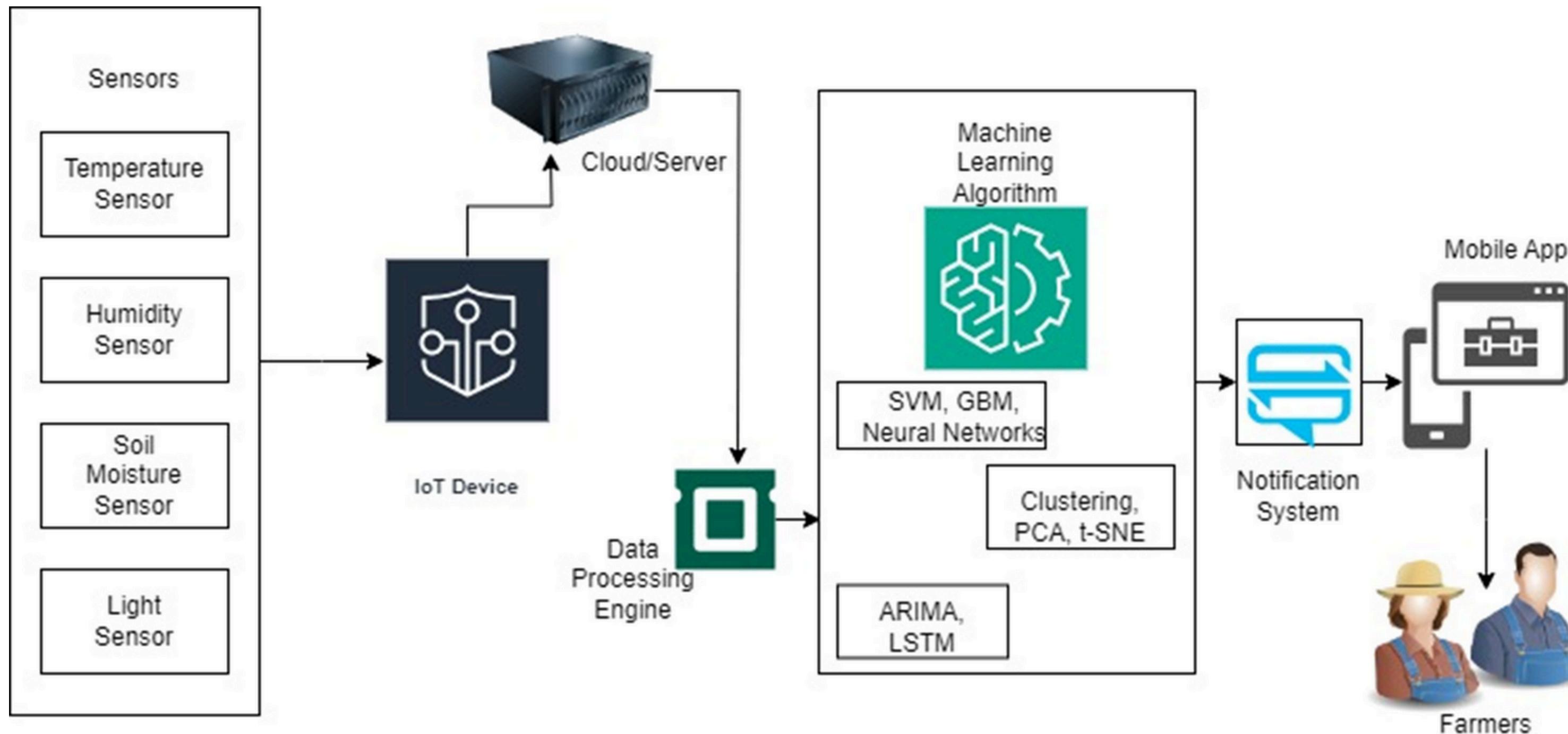
Existing Products / Research	Features						
	Automated Fertilizer Application	Real-Time Nutrient Monitoring	Precision Nutrient Delivery	Environmentally Sustainable Practices	Integration with IoT and Sensors	Impact on Fruit Quality	Predictive Analytics for Nutrient Needs
IoT-Based Precision Fertilization System	✓	✗	✓	✗	✓	✗	✗
Smart Fertigation Management System	✓	✓	✗	✗	✗	✓	✗
Integrated Hydroponic Nutrient Solution	✗	✓	✓	✓	✗	✗	✓
Proposed System	✓	✓	✓	✓	✓	✓	✓

METHODOLOGY

- 1.Gather soil, nutrient data and environmental conditions.
- 2.Collect historical fertilization schedules and crop yield data.
- 3.Preprocess the data to handle missing values and normalize it.
- 4.Choose appropriate hybrid ML models.
- 5.Train the models using the preprocessed data to optimize fertilization schedules.



SYSTEM DIAGRAM



DATASET

	A	B	C	D	E	F	G	H
1	Organic Matter	Humidity (%)	Temperature (°C)	Fertilizer Type	Fertilizer Quantity (kg/ha)	Fertilizer Timing (Week)	Crop Yield (tons/ha)	Historical Yield (tons/ha)
2	2.749080238	75.83158087	20.13094457	Potash	180.4636872	8	4.675482342	7.215243009
3	3.901428613	75.79236286	26.63537372	Ammonium Nitrate	176.9410612	7	8.222115865	4.788313252
4	3.463987884	61.82412206	21.78035967	DAP	218.7758391	11	7.24419086	3.632189534
5	3.197316968	69.88840609	29.61070317	Potash	97.97520052	6	4.579206821	7.69094318
6	2.312037281	61.1511752	21.48662728	Urea	94.57535091	7	5.45136297	6.864895329
7	2.311989041	70.99057765	24.14624124	Potash	181.4074344	5	4.074424218	3.289116074
8	2.116167224	68.83061003	20.85349668	Ammonium Nitrate	160.9741684	5	6.812607793	5.160302795
9	3.732352292	77.75408366	29.96874252	Ammonium Nitrate	118.3824933	6	5.807591565	8.740798893
10	3.202230023	67.01830025	25.0219501	DAP	91.11871166	4	7.590144906	6.002406874
11	3.416145156	62.34134033	25.95385017	Potash	91.99215494	8	5.783426846	5.596374816
12	2.041168989	62.85983364	20.67076477	Urea	205.1867225	7	5.799514669	5.746197631
13	3.939819704	75.23021263	27.4996047	Urea	106.8614255	5	8.459159192	4.253296344
14	3.664885282	72.36436127	22.09905593	NPK	125.2720187	4	4.288857016	5.212243025
15	2.424678221	62.02245352	28.98054289	Urea	111.7318963	6	9.417369971	5.218879536
16	2.363649934	61.68213612	22.0513964	DAP	129.6994828	10	9.113583246	3.314130121
17	2.36680902	74.01938263	21.90687721	DAP	89.71933798	9	8.006828099	7.605405771
18	2.608484486	61.45526013	20.36549668	Urea	152.6683707	10	7.559329421	5.499026591
19	3.049512863	76.43720119	24.72066945	Urea	89.4657589	10	9.353814989	7.933079321
20	2.863890037	74.12484454	25.64841133	NPK	192.0499111	5	5.111981077	8.102088007
21	2.58245828	61.62697561	20.65708639	Urea	112.7196915	11	4.473814973	4.271964328
22	3.223705789	61.69675428	27.75527617	NPK	155.601668	11	5.437061019	6.944120893
23	2.278987721	79.73279157	24.53288835	NPK	203.2110723	6	8.767469672	5.833690825
24	2.584289297	67.48541592	25.24390269	Urea	171.1228317	7	4.2080216	8.280940428
25	2.732723687	67.41284294	24.40762747	Ammonium Nitrate	154.6140901	11	7.496841011	4.294444075
26	2.912139968	76.25599135	24.00763061	Ammonium Nitrate	125.4067254	10	9.972625097	7.066879202
27	3.570351923	78.94497155	25.59640331	NPK	126.6202678	11	9.134176563	6.646512734
28	2.399347564	79.72002128	21.55240246	Urea	173.7281737	10	7.128674134	4.771808865

< > fertilization_data +

DATA PREPROCESSING

```
9     # Load the dataset
10    file_path = "fertilization_data.csv"
11    df = pd.read_csv(file_path)
12
13    # Preprocess
14    df = pd.get_dummies(df, columns=['Fertilizer Type'], drop_first=True)
15
16    # Split data
17    X = df.drop(['Fertilizer Quantity (kg/ha)', 'Fertilizer Timing (Week)'], axis=1) # Features
18    y = df[['Fertilizer Quantity (kg/ha)', 'Fertilizer Timing (Week)']] # Targets
19
20    # Train-test split
21    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
22
23    # Normalize
24    scaler = StandardScaler()
25    X_train_scaled = scaler.fit_transform(X_train)
26    X_test_scaled = scaler.transform(X_test)
27
```

MODEL TRAINING

```
28 # Train the models
29 rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
30 gb_model = GradientBoostingRegressor(n_estimators=100, random_state=42)
31 rf_model.fit(X_train_scaled, y_train['Fertilizer Quantity (kg/ha)'])
32 gb_model.fit(X_train_scaled, y_train['Fertilizer Timing (Week)'])
33 rf_predictions = rf_model.predict(X_test_scaled)
34 gb_predictions = gb_model.predict(X_test_scaled)
35
36 # Evaluate
37 rf_mae = mean_absolute_error(y_test['Fertilizer Quantity (kg/ha)'], rf_predictions)
38 rf_mse = mean_squared_error(y_test['Fertilizer Quantity (kg/ha)'], rf_predictions)
39 rf_r2 = r2_score(y_test['Fertilizer Quantity (kg/ha)'], rf_predictions)
40
41 gb_mae = mean_absolute_error(y_test['Fertilizer Timing (Week)'], gb_predictions)
42 gb_mse = mean_squared_error(y_test['Fertilizer Timing (Week)'], gb_predictions)
43 gb_r2 = r2_score(y_test['Fertilizer Timing (Week)'], gb_predictions)
44 rf_cv_scores = cross_val_score(rf_model, X_train_scaled, y_train['Fertilizer Quantity (kg/ha)'], cv=5, scoring='neg_mean_absolute_error')
45 gb_cv_scores = cross_val_score(gb_model, X_train_scaled, y_train['Fertilizer Timing (Week)'], cv=5, scoring='neg_mean_absolute_error')
```

SENSOR DATA - FUNCTION

The screenshot shows the PyCharm IDE interface with the following details:

- Project:** MylastFyp (C:\Users\user\Desktop\MylastFyp)
- File Structure:** static, css, img, js, uploads, webfonts, templates (containing index.html, main.html), API.py, App.py, fertilization_data.csv, rf_model.pkl, Predict_Fertilization.py, Train_Fertilization.py, waterering-7b4c7-firebase-adminsdk-ntnzb-ee048fb927.json.
- Code Editor:** App.py (selected tab) showing Python code for sensor data and prediction functions.
- Code Content:**

```
8
9     # Load models
10    rf_model = joblib.load('rf_model.pkl')
11    gb_model = joblib.load('gb_model.pkl')
12    scaler = joblib.load('scaler.pkl')
13    # Initialize Firebase Admin SDK
14    if not firebase_admin._apps:
15        cred = credentials.Certificate("watering-7b4c7-firebase-adminsdk-ntnzb-ee048fb927.json")
16        firebase_admin.initialize_app(cred, options={
17            'databaseURL': 'https://watering-7b4c7-default.firebaseio.com/'
18        })
19
20    # Realtime Database path
21    sensor_ref = db.reference('/sensors')
22
23    # Sensor data function
24    def get_sensor_data(): 1 usage
25        ref = db.reference('/sensors')
26        data = ref.get()
27        return data
28
29
30    # Prediction function
```
- Run Tab:** Run → App
- Output Tab:** Shows log entries:

```
127.0.0.1 - - [18/Mar/2025 11:19:50] "GET /static/css/tooplate.css HTTP/1.1" 304 -
127.0.0.1 - - [18/Mar/2025 11:19:50] "GET /static/js/jquery-3.3.1.min.js HTTP/1.1" 304 -
127.0.0.1 - - [18/Mar/2025 11:19:51] "GET /static/js/bootstrap.min.js HTTP/1.1" 304 -
```
- Status Bar:** 31:16 CRLF UTF-8 4 spaces Python 3.8

PREDICTION FUNCTION

The screenshot shows a PyCharm IDE interface with two code editors open:

- App.py:** This file contains a prediction function and a route handler for the root URL. It uses pandas to process input data, applies a scaler, and makes predictions with two different models (rf_model and gb_model). It also handles sensor data from a POST request and returns the results as JSON.

```
30 # Prediction function
31 def predict(input_data): 7 usages (6 dynamic)
32     input_df = pd.DataFrame(input_data)
33     input_df = pd.get_dummies(input_df, columns=['Fertilizer Type'], drop_first=True)
34     input_df = input_df.reindex(columns=scaler.feature_names_in_, fill_value=0)
35     X_scaled = scaler.transform(input_df)
36     predicted_quantity = rf_model.predict(X_scaled)
37     predicted_timing = gb_model.predict(X_scaled)
38     return predicted_quantity[0], predicted_timing[0]
39
40 @app.route(rule='/', methods=['GET', 'POST'])
41 def index():
42     prediction_result = None
43     sensor_data = get_sensor_data()
44
45     temperature = humidity = soil_moisture = light_level = None
46
47     if sensor_data:
48         temperature = sensor_data.get('temperature', '')
49         humidity = sensor_data.get('humidity', '')
50         soil_moisture = sensor_data.get('soilMoisture', '')
51         light_level = sensor_data.get('lightLevel', '')
52
53     if request.method == 'POST':
54         humidity = float(request.form['humidity'])
55         temperature = float(request.form['temperature'])
56         fertilizer_type = request.form['fertilizer_type']
57
58     input_data = {
59         'Humidity (%)': [humidity],
60         'Temperature (°C)': [temperature],
61         'Fertilizer Type': [fertilizer_type]
62     }
63
64     quantity, timing = predict(input_data)
65     prediction_result = {
66         'quantity': quantity,
67         'timing': timing
68     }
69
70     return render_template(template_name_or_list='index.html', prediction=prediction_result, temperature=temperature, hu
71
72
73 if __name__ == '__main__':
74     app.run(debug=True)
```

- API.py:** This file contains a single route handler for the '/api' endpoint. It retrieves sensor data and performs a prediction, returning the results as JSON.

```
41 def index():
42     light_level = sensor_data.get('lightLevel', '')
43
44     if request.method == 'POST':
45         humidity = float(request.form['humidity'])
46         temperature = float(request.form['temperature'])
47         fertilizer_type = request.form['fertilizer_type']
48
49     input_data = {
50         'Humidity (%)': [humidity],
51         'Temperature (°C)': [temperature],
52         'Fertilizer Type': [fertilizer_type]
53     }
54
55     quantity, timing = predict(input_data)
56     prediction_result = {
57         'quantity': quantity,
58         'timing': timing
59     }
60
61     return jsonify(prediction_result)
```

API S

The screenshot shows a PyCharm IDE interface with the following details:

- Project:** MylastFyp (C:\Users\user\Desktop\MylastFyp)
- File Structure:** MylastFyp folder contains static, css, img, js, uploads, webfonts, templates, and several Python files (App.py, API.py, Predict_Fertilization.py, Train_Fertilization.py) and model files (gb_model.pkl, rf_model.pkl, scaler.pkl).
- API.py Content:**

```
12 def predict(input_data):    7 usages (6 dynamic)
14     input_df = pd.get_dummies(input_df, columns=['Fertilizer Type'], drop_first=True)
15     input_df = input_df.reindex(columns=scaler.feature_names_in_, fill_value=0)
16     X_scaled = scaler.transform(input_df)
17     predicted_quantity = rf_model.predict(X_scaled)
18     predicted_timing = gb_model.predict(X_scaled)
19     return predicted_quantity[0], predicted_timing[0]
20
21 # API
22 @app.route('/predict', methods=['POST'])
23 def predict_api():
24     try:
25
26         input_data = request.get_json()
27         quantity, timing = predict(input_data)
28         return jsonify({
29             'Predicted Fertilizer Quantity (kg/ha)': quantity,
30             'Predicted Fertilizer Timing (Week)': timing
31         })
32
33     except Exception as e:
34         return jsonify({'error': str(e)}), 400
35
36 if __name__ == "__main__":
37     app.run(debug=True)
```
- Run Tab:** App
- Bottom Status Bar:** MylastFyp > API.py, 10:1 CRLF UTF-8 4 spaces Python 3.8

USER INTERFACE

The screenshot displays the AI-IoT Plantation user interface with a green header bar. The header includes the logo "AI-IoT Plantation", navigation links for "Dashboard", "Irrigation", "Fertilizer" (which is highlighted in blue), "Harvest", "History", "Automation Dashboard", and a user account section with the email "u1@gmail.com".

The main content area is titled "Fertilizer Prediction" and contains the sub-section "Calculate the optimal fertilizer quantity and application timing for your crops.". A green callout box shows the message "Last fertilizer application: 2025-05-01" with a calendar icon.

Below this, there is a large input form with the following fields:

- Humidity (%): Input field containing "91".
- Temperature (°C): Input field containing "30.8".
- Fertilizer Type: A dropdown menu currently set to "Ammonium Nitrate".
- Area planted (ha): An input field containing "0.1", which is highlighted with a green border.
- Soil Type: A dropdown menu currently set to "Clay".
- Growth Stage: A dropdown menu currently set to "Seedling".

A blue button labeled "Update Fertilizer Date" with a calendar icon is positioned above the input form. At the bottom of the form is a large green button labeled "Predict Fertilizer Needs" with a bar chart icon.

USER INTERFACE

The screenshot displays the AI-IoT Plantation user interface. At the top, there is a navigation bar with the logo "AI-IoT Plantation", a "Dashboard" link, and a user account section showing "u1@gmail.com". Below the navigation bar, the main content area features a title "Fertilizer Prediction" with a subtitle "Calculate the optimal fertilizer quantity and application time". A large button labeled "Predict Fertilizer Needs" is prominently displayed. In the center, a modal window titled "Update Fertilizer Date" is open, prompting the user to enter the "Last Fertilizer Application Date" in "mm/dd/yyyy" format. The modal includes a "Close" button and a blue "Save Date" button. In the background, there are input fields for environmental parameters: Humidity (%), Temperature (°C), Fertilizer Type (Ammonium Nitrate), Area planted (ha), Soil Type (Clay), and Growth Stage (Seedling). A "Update Fertilizer Date" button is also visible in the background.

AI-IoT Plantation

Dashboard

Automation Dashboard

u1@gmail.com

Fertilizer Prediction

Calculate the optimal fertilizer quantity and application time

Last fertilizer application date

mm/dd/yyyy

Close

Save Date

Update Fertilizer Date

Humidity (%)

91

Temperature (°C)

30.8

Fertilizer Type

Ammonium Nitrate

Area planted (ha)

0.1

Soil Type

Clay

Growth Stage

Seedling

Predict Fertilizer Needs

USER INTERFACE

The screenshot displays the AI-IoT Plantation user interface, specifically the Fertilizer tab. The top navigation bar includes links for Dashboard, Irrigation, Fertilizer (highlighted in green), Harvest, History, Automation Dashboard, and a user account for u1@gmail.com.

In the main content area, there are two input fields: "Fertilizer Type" (set to Ammonium Nitrate) and "Area planted (ha)" (empty). Below these are two dropdown menus: "Soil Type" (set to Clay) and "Growth Stage" (set to Seedling). A large green button labeled "Predict Fertilizer Needs" is centered below the inputs.

The "Prediction Results" section contains two boxes: "Fertilizer Quantity" (114.35 kg/ha) and "Application Timing" (Week 5). At the bottom, a green bar indicates the "Next Application Date: 2025-06-05".

Parameter	Value
Fertilizer Type	Ammonium Nitrate
Area planted (ha)	(Empty)
Soil Type	Clay
Growth Stage	Seedling
Fertilizer Quantity	114.35 kg/ha
Application Timing	Week 5
Next Application Date	2025-06-05

REFERENCES

- [1] "Big Data Analytics for Smart Agriculture" by Rajasekaran, et al. Explores the role of big data analytics in agriculture, including data processing and visualization.
[https://www.researchgate.net/publication/379404505_Role_of_Data_Visualization_and_Big_Data_Analytics_in_Smart_Agriculture]
- [2] Lee, E. Y., & Parker, R. S. (2019). Mobile App Development with React Native. Packt Publishing.
[<https://www.packtpub.com/en-us/product/react-cross-platform-application-development-with-react-native-9781789136081>]
- [3] Lee, G. J., & Lee, M. Y. (2018). Cloud Computing for Smart Agriculture: A Survey. Future Generation Computer Systems, 78, 287-299.
[https://www.researchgate.net/publication/321896167_A_Survey_Smart_agriculture_IoT_with_cloud_computing]
- [4] Kim, Y. H., et al. (2020). Optimizing Cloud Resources for Machine Learning Applications. IEEE Transactions on Cloud Computing, 8(1), 123-135.



IT21231414 | Jayaneththi I.H.N.S

Information technology



RESEARCH GAP

These research gaps contribute to the development of more advanced and effective devices for quality check of fresh tomatoes, ultimately benefit tomato producers, processors, and consumers alike.

According to that research,

Developing advanced algorithms for real-time data analysis to enable immediate feedback on the quality of fresh tomatoes

Understanding the impact of environmental factors on the performance of quality check devices

Research could focus on developing intuitive interfaces that provide clear and actionable insights into the quality of fresh tomatoes.

RESEARCH QUESTION

How can environmental and agronomic factors be integrated into a predictive model to accurately forecast the optimal harvest dates for different tomato varieties?



SPECIFIC AND SUBJECTIVE

Specific Objective

To develop and validate a machine learning-based model that accurately predicts the optimal harvest dates for tomatoes by analyzing key environmental factors (temperature, humidity, light) and agronomic practices (soil fertility, irrigation, and planting schedules).

Sub Objective

- Develop a predictive model using machine learning algorithms
 - Identify and engineer relevant features from the collected data
 - Design and implement a decision support tool that integrates the predictive model

TECHNOLOGIES & ALGORITHMS

1. Real-time data collection and analysis

2. Machine Learning for prediction models

Algorithms

- **Machine Learning Models:** random forest Regression algorithms for predicting schedule

PROGRESS

100% COMPLETION OF THE PROJECT

DATASET

harvest_dataset

Planting Date	Growth Stage (Days)	Temperature (°C)	Humidity (%)	Light Exposure (hrs/day)	Soil Moisture (%)	Pesticide Used (Yes=1, No=0)	Harvest Days (Target)
04/25/2024	52	28.3	67.6	9.6	25.9	1	55
04/25/2024	24	27.2	59.2	10.0	31.0	0	55
04/25/2024	60	29.9	58.9	10.1	33.9	0	55
04/25/2024	51	26.4	59.9	10.1	30.8	0	55
04/25/2024	2	25.0	59.2	9.2	38.6	0	63
04/25/2024	4	32.6	66.0	10.3	38.8	0	58
04/25/2024	87	27.5	68.2	10.7	34.1	0	55
04/25/2024	29	25.6	63.1	11.5	36.6	1	55
04/25/2024	41	26.7	65.8	9.6	35.0	1	55
04/25/2024	44	26.9	65.0	10.5	36.4	0	55
04/25/2024	14	21.8	64.1	10.1	29.7	0	64
04/25/2024	23	24.2	61.0	10.2	35.4	0	76
04/25/2024	42	28.8	61.6	9.2	38.4	1	55
04/25/2024	89	26.5	58.2	11.9	37.9	0	55
04/25/2024	72	30.9	59.2	8.6	27.5	0	55
04/25/2024	88	27.5	59.6	7.1	29.3	1	55
04/26/2024	2	26.6	61.5	9.9	39.8	0	58
04/26/2024	68	24.4	56.7	9.8	39.8	0	55
04/26/2024	16	25.0	57.2	10.1	35.2	1	55
04/26/2024	67	25.6	58.7	9.8	32.8	1	55
04/26/2024	11	26.3	62.8	11.6	33.3	0	69
04/26/2024	2	27.1	62.5	9.8	29.2	1	69
04/26/2024	80	27.5	61.7	9.0	27.4	1	55
04/26/2024	84	28.1	59.2	9.8	31.7	0	55
04/26/2024	33	24.3	59.9	12.0	35.8	0	55
04/26/2024	48	24.3	56.1	11.3	34.5	0	58
04/26/2024	30	28.7	62.0	10.0	33.6	0	62
04/26/2024	17	23.4	58.8	10.5	39.3	1	55
04/26/2024	46	25.3	52.4	9.3	27.5	1	55
04/26/2024	54	25.2	60.8	10.1	38.2	0	55



User Interface

The screenshot shows the 'Harvest' section of the AI-IoT Plantation web application. At the top, there is a navigation bar with the following items: 'AI-IoT Plantation' (with a leaf icon), 'Dashboard' (with a chart icon), 'Irrigation' (with a water drop icon), 'Fertilizer' (with a fertilizer icon), 'Harvest' (with a calendar icon, currently selected), 'History' (with a clock icon), 'Automation Dashboard' (with a gear icon), and a user account icon labeled 'u1@gmail.com'.

The main content area is titled 'Harvest Prediction' with a subtitle 'Predict the optimal harvest time based on current plant conditions.' Below this, there is a form with the following fields:

- Planting Date:** A date input field with placeholder 'mm/dd/yyyy' and a calendar icon.
- Growth Stage (Days) 1-90:** An empty input field.
- Temperature (°C):** An input field containing '41.6'.
- Humidity (%):** An input field containing '46'.
- Light Exposure (hrs/day):** An input field containing '624'.
- Soil Moisture (%):** An input field containing '4095'.
- Pesticide Used:** A dropdown menu showing 'No'.

At the bottom of the form is a large green button with the text 'Predict Harvest Days' and a chart icon.

User Interface

The screenshot shows the 'Harvest' section of the AI-IoT Plantation web application. At the top, there is a navigation bar with links for Dashboard, Irrigation, Fertilizer, Harvest (which is the active tab), History, Automation Dashboard, and a user account (u1@gmail.com). The main content area is titled 'Harvest Prediction' and includes a subtitle: 'Predict the optimal harvest time based on current plant conditions.' Below this, there is a form with input fields for Planting Date (mm/dd/yyyy), Growth Stage (Days) 1-90, Temperature (°C) (41.6), Humidity (%)(46), Light Exposure (hrs/day) (624), Soil Moisture (%)(4095), and Pesticide Used (No). A green button labeled 'Predict Harvest Days' is present. Below the form, a large callout box displays the predicted harvest days: 'Predicted Harvest Days: 61 days', with the text 'Based on current growth conditions' underneath. A horizontal bar at the bottom of this box is labeled 'Optimal Harvest Window'.

AI-IoT Plantation

Dashboard Irrigation Fertilizer **Harvest** History Automation Dashboard u1@gmail.com

Harvest Prediction

Predict the optimal harvest time based on current plant conditions.

Planting Date: mm/dd/yyyy

Growth Stage (Days) 1-90:

Temperature (°C): 41.6

Humidity (%): 46

Light Exposure (hrs/day): 624

Soil Moisture (%): 4095

Pesticide Used: No

Predict Harvest Days

Harvest Prediction

Predicted Harvest Days: 61 days

Based on current growth conditions

Optimal Harvest Window

REFERENCES

- [1] Gutiérrez, S., Diacono, M., Castrignanò, A., & Sardone, R. (2016). Application of multivariate geostatistical methods for managing precision viticulture. *Computers and Electronics in Agriculture*, 121, 386–396. <https://doi.org/10.1016/j.compag.2016.02.018>
- [2] Hewett, E. W. (2006). An overview of pre-harvest factors influencing postharvest quality of horticultural products. *International Journal of Postharvest Technology and Innovation*, 1(1), 4–15. <https://doi.org/10.1504/IJPTI.2006.009629>
- [3] Saggi, M.K. & Jain, S. A. (2022). Survey Towards Decision Support System on Smart Irrigation Scheduling Using Machine Learning approaches. *Archives of Computational Methods in Engineering*, 29, 4455–4478. <https://doi.org/10.1007/s11831-022-09746-3>.
- [4] Salim, O., Fouad, K. & Hassan, B. (2022). Dual-Level Sensor Selection with Adaptive Sensor Recovery to Extend WSNs' Lifetime, *Human-centric Computing and Information Sciences*, Vol. 12, Article number: 18, <https://doi.org/10.22967/HCIS.2022.12.018>.
- [5] Goap, A., Sharma, D., Shukla, A.K. & Krishna, R. (2018). An IoT based smart irrigation management system using Machine learning and open source technologies, *Computers and Electronics in Agriculture*, Vol. 155, PP: 41–49, ISSN 0168-1699, <https://doi.org/10.1016/j.compag.2018.09.040>.
- [6] Fouad, K. & Elbably, D. (2020). Intelligent approach for large-scale data mining. *Int. J. Computer Applications in Technology*, Vol. 63, Nos. 1/2, PP: 93–112.
- [7] Fouad, K., Hassan, B. & Salim, O. (2022). Hybrid Sensor Selection Technique for Lifetime Extension of Wireless Sensor Networks. *Computers, Materials & Continua* 2022, 70(3), 4965–4985. <https://doi.org/10.32604/cmc.2022.020926>.



IT21186592 | Najas MNM

Information technology



IT21186592 | NAJAS MNM | 24-25J-064

COMPONENT

Detecting Tomato Diseases and Recommending Appropriate Treatment



INTRODUCTION BACKGROUND

- This project aims to develop a comprehensive system that leverages image analysis and machine learning to detect tomato diseases and recommend appropriate treatments.
- The Impact of Diseases on Tomato Crops.
- Challenges in Disease Management.



RESEARCH QUESTIONS

How can we develop a decision support system for tomato greenhouses ?



RESEARCH GAP

Existing Products / Research	Predictive Analytics				
	Image Analysis Tools	Manual Inspection	Disease Specific Detection	Automated Treatment Recommendation	Predictive Analytics
Current Methods	✗	✓	✓	✗	✗
Basic Automated Tools	✓	✓	✗	✗	✗
Proposed System	✓	✗	✓	✓	✓

MAIN & SPECIFIC OBJECTS

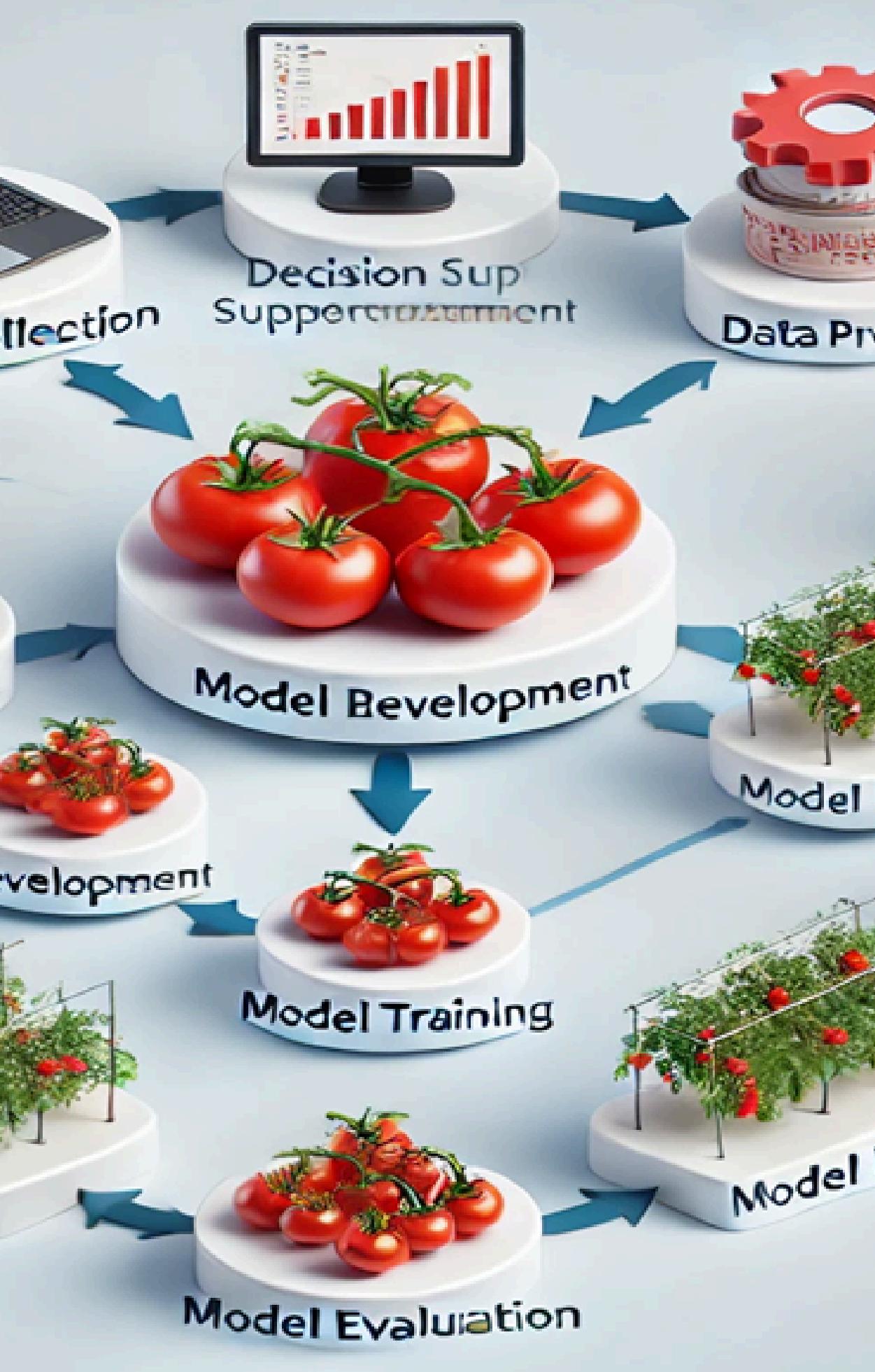
Main objectives

Develop a comprehensive system to detect tomato diseases and recommend appropriate treatments using image analysis and machine learning.

Sub objectives

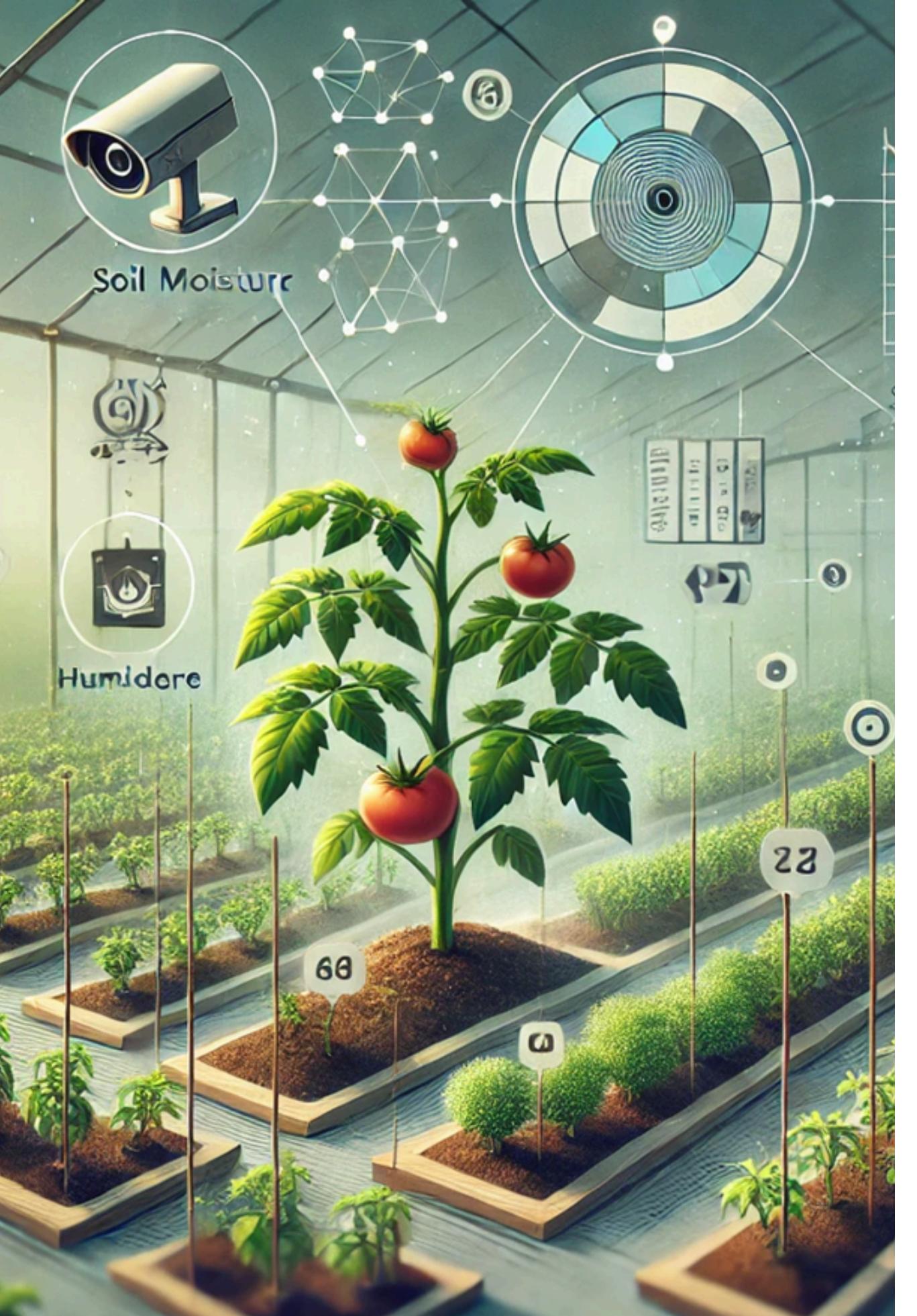
- Select data from Kaggle.
- Label the images with corresponding disease types.
- Preprocess the images (resize, normalize, augment) for model training.
- Develop a model architecture suitable for image classification.
- Train the CNN model on the labeled dataset and evaluate its performance.





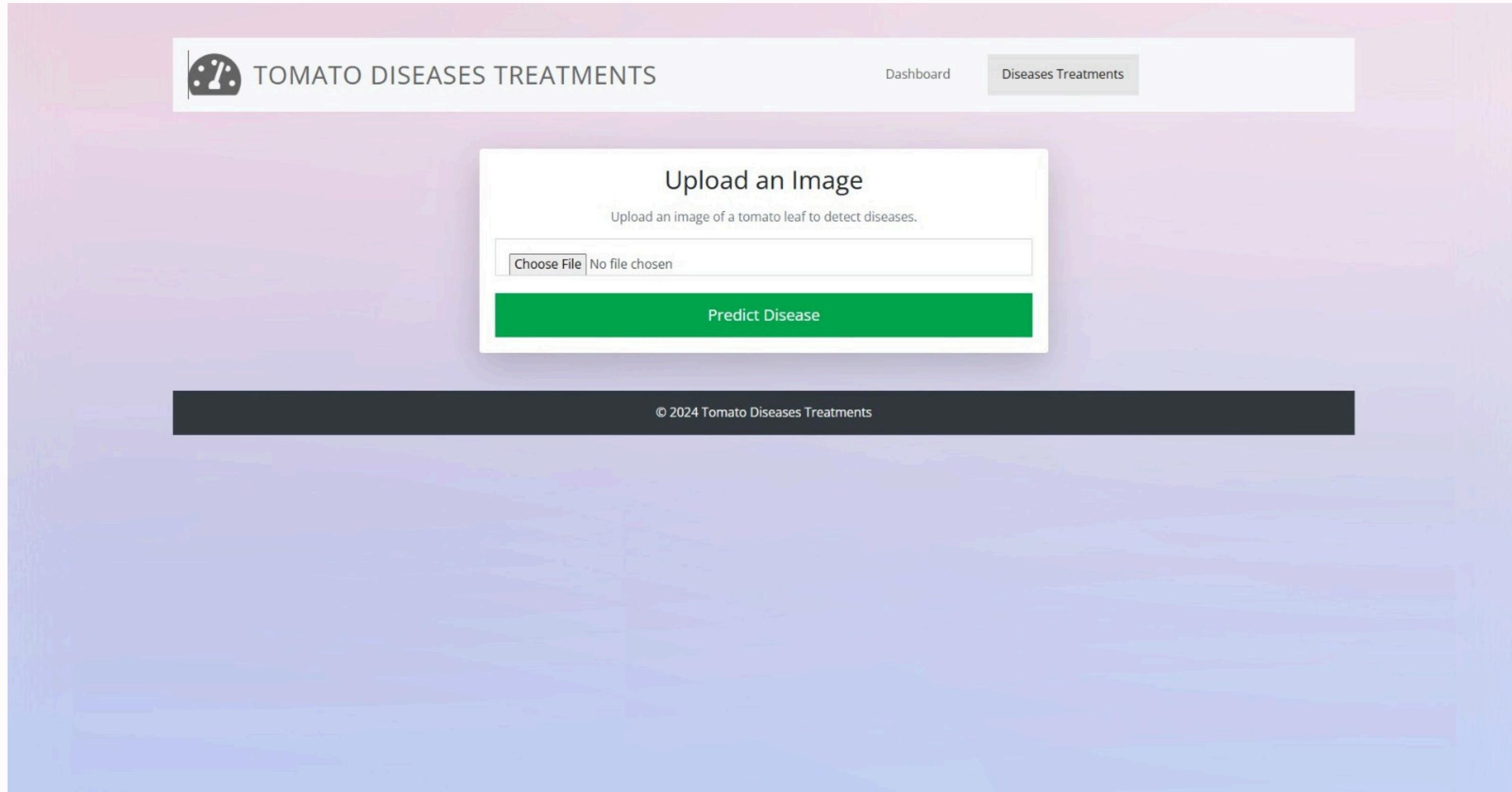
METHODOLOGY

- Python, TensorFlow(Open-Source ML framework), Keras.
 - Image preprocessing techniques.
 - CNN model design and training.
 - Data visualization tools.
- Data Collection and Labeling:** Select and annotate images from Kaggle.
- Image Preprocessing:** Resize, normalize, and augment images.
- CNN Model Development:** Design and train a CNN model for disease classification.
- Performance Evaluation:** Evaluate the model's performance using metrics such as accuracy, precision, and recall.
- Implementation of Recommendation System:** Provide treatment recommendations based on disease detection.



DATA COLLECTION

- Kaggle datasets.



A screenshot of a web application interface titled "TOMATO DISEASES TREATMENTS". The title bar includes a tomato icon and two navigation links: "Dashboard" and "Diseases Treatments". The main content area features a white box with a title "Upload an Image" and a subtitle "Upload an image of a tomato leaf to detect diseases." It contains a file input field labeled "Choose File" with the message "No file chosen" and a green button labeled "Predict Disease". At the bottom of the page is a dark footer bar with the copyright text "© 2024 Tomato Diseases Treatments".

TOMATO DISEASES TREATMENTS

Dashboard Diseases Treatments

Upload an Image

Upload an image of a tomato leaf to detect diseases.

Choose File No file chosen

Predict Disease

© 2024 Tomato Diseases Treatments

OUTPUT

 TOMATO DISEASES TREATMENTS

Dashboard Diseases Treatments

Upload an Image

Upload an image of a tomato leaf to detect diseases.

Choose File No file chosen

Predict Disease

Uploaded Image



Predicted Disease

Tomato Spider mites

Suggested Treatment

Recommended treatment: Use miticides or introduce natural predators like ladybugs.

REQUIREMENTS

System requirements

- High-quality annotated image datasets.
- Efficient CNN model for disease detection.
- Treatment recommendation algorithms.
- User-friendly dashboard for real-time alerts and recommendations.

Non-functional Requirements

- Performance
- Reliability
- Scalability
- Security
- Usability
- Maintainability
- Accuracy

REFERENCES

- Mohanty, S. P., Hughes, D. P., & Salathé, M. (2016). Using deep learning for image-based plant disease detection. *Frontiers in Plant Science*, 7, 1419.
- Zhang, S., Huang, W., Zhang, C., & Wang, X. (2018). Plant disease recognition based on plant leaf image. *Journal of Agricultural Mechanics Research*, 9, 94-99.
- Barbedo, J. G. A. (2016). A review on the main challenges in automatic plant disease identification based on visible range images. *Biosystems Engineering*, 144, 52-60.