Electric Motor Temperature Prediction Using Machine Learning

Abstract

This project predicts the Permanent Magnet (PM) surface temperature of an electric motor using machine learning. A Random Forest Regressor is trained on the Kaggle Electric Motor Temperature dataset and deployed using a Flask web application.

Problem Statement

Overheating in electric motors can cause failures and downtime. This project aims to predict motor temperature using sensor data instead of physical temperature sensors.

Dataset Description

Dataset Source: Kaggle Electric Motor Temperature Dataset

Tools & Technologies

Python, Google Colab, VS Code, Flask, Scikit-learn, Pandas, NumPy, Joblib

Tools & Technologies Used

1. Python

Description:

Python is a high-level, interpreted programming language widely used in data science and machine learning. It provides simplicity, readability, and a vast ecosystem of libraries that make data preprocessing, model training, and deployment efficient. In this project, Python is used as the core programming language for data handling, machine learning model development, and backend logic of the web application.

Role in Project:

- Data preprocessing and feature selection
- Machine learning model training and evaluation
- Backend development using Flask

---

2. Google Colab

Description:
Google Colab is a cloud-based Jupyter notebook environment provided by Google. It allows users to write and execute Python code directly in the browser without any local setup. Google Colab provides free access to computing resources and is ideal for machine learning experiments and model training.

In this project, Google Colab was used to load the Kaggle dataset, perform exploratory data analysis (EDA), preprocess data, train multiple machine learning models, and save the best-performing model and scaler.

Role in Project:

- Dataset loading and visualization
- Feature scaling and preprocessing
- Model training and evaluation
- Saving trained model and scaler files

---

3. VS Code (Visual Studio Code)

Description:
Visual Studio Code (VS Code) is a lightweight yet powerful source code editor developed by Microsoft. It supports Python development, Flask applications, and virtual environments with ease. VS Code was used to integrate the trained machine learning model into a Flask web application and run the project locally.

Role in Project:

- Flask application development
- Integration of trained model and scaler

- Running and testing the web application

---

4. Flask

Description:
Flask is a lightweight Python web framework used to build web applications. It is simple, flexible, and easy to integrate with machine learning models. In this project, Flask is used to create a web interface where users can input electric motor parameters and obtain predicted temperature values.

Role in Project:

- Backend web framework
- Handling user input from HTML form
- Sending data to the trained ML model for prediction
- Displaying predicted output on the web page

---

5. Scikit-learn

Description:
Scikit-learn is a popular machine learning library in Python that provides simple and efficient tools for data analysis and modeling. It supports various regression algorithms, preprocessing techniques, and evaluation metrics.

In this project, Scikit-learn is used to train multiple regression models and evaluate their performance.

Algorithms Used:

- Linear Regression
- Decision Tree Regressor
- Random Forest Regressor
- Support Vector Machine (SVR)

Role in Project:

- Model training and testing
- Feature scaling (MinMaxScaler)
- Performance evaluation (MAE, RMSE, R^2)

---

6. Pandas

Description:
Pandas is a powerful Python library used for data manipulation and analysis. It provides data structures such as DataFrames that make it easy to clean, preprocess, and analyze large datasets.

Role in Project:

- Reading and handling dataset
- Data cleaning and transformation
- Feature selection and column operations

---

7. NumPy

Description:
NumPy is a fundamental Python library for numerical computations. It provides support for arrays, mathematical operations, and efficient numerical processing. NumPy is heavily used internally by machine learning libraries.

Role in Project:

- Numerical operations
- Array transformations
- Supporting machine learning computations

---

8. Joblib

Description:
Joblib is a Python library used for saving and loading machine learning models efficiently. In this project, Joblib is used to store the trained Random Forest model and the MinMaxScaler, enabling easy reuse during deployment.

Role in Project:

- Saving trained ML model (model.save)
- Saving scaler (transform.save)
- Loading model and scaler in Flask application

Model Building

Multiple models were evaluated including Linear Regression, Decision Tree, Random Forest, and SVM. Random Forest Regressor achieved the best performance.

Model Evaluation For Random Forest Regression

MAE: 2.00
RMSE: 3.58
R^2 Score: 0.964

Deployment of Electric Motor Temperature Prediction System

1. Overview of Deployment

Deployment is the process of **making a trained machine learning model available for real-world usage**.
In this project, the trained **Random Forest Regression model** and the **MinMaxScaler**

- Downloaded and transferred to **VS Code**
- Integrated into a **Flask web application**
- Used to predict **Permanent Magnet (PM) temperature** based on user inputs through an **HTML interface**
- Trained and saved in **Google Colab**

---

2. Deployment Architecture

Flow of the System

User → HTML Web Page → Flask App → Scaler → ML Model → Prediction → Web Page

Components

Component	Purpose
Google Colab	Model training and saving
Joblib	Model & scaler serialization
VS Code	Application development
Flask	Backend web framework
HTML/CSS	User interface
Random Forest Model	Temperature prediction

---

3. Model Training and Saving (Google Colab)

3.1 Training the Model

- Dataset loaded from Kaggle
- Features selected (ambient, coolant, u_d, i_d, i_q)
- Target variable: pm
- MinMaxScaler applied
- Random Forest Regressor trained

3.2 Saving Model and Scaler

After training, both the model and scaler are saved using **Joblib**:

```
import joblib

joblib.dump(rf_model, "model.save")

joblib.dump(mm, "transform.save")
```

3.3 Downloading Files from Colab

```
from google.colab import files

files.download("model.save")

files.download("transform.save")
```

📸 TRAINING & TESTING

```
# ==========================
# STEP 1: Imports
# ==========================
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
import pandas as pd
import joblib

# ==========================
# STEP 2: Feature & Target Selection
# ==========================
features_to_scale = ['ambient', 'coolant', 'u_d', 'i_d', 'i_q']
target_column = 'pm'

# Input features and target
X_data = df1[features_to_scale]
y_data = df1[target_column]

# ==========================
# STEP 3: Train-Test Split
# ==========================
X_train, X_test, y_train, y_test = train_test_split(
    X_data,
    y_data,
    test_size=0.2,
    random_state=42
)

# ==========================
# STEP 4: MinMax Scaling
# ==========================
mm = MinMaxScaler()

# Fit ONLY on training data
X_train_scaled = mm.fit_transform(X_train)

# Transform test data using SAME scaler
X_test_scaled = mm.transform(X_test)

# Convert back to DataFrame
X = pd.DataFrame(X_train_scaled, columns=features_to_scale)
X_df1_test = pd.DataFrame(X_test_scaled, columns=features_to_scale)

# ==========================
# STEP 5: Target Variables
# ==========================
y = y_train.reset_index(drop=True)
y_df1_test = y_test.reset_index(drop=True)

# ==========================
# STEP 6: Save Scaler
# ==========================
joblib.dump(mm, 'transform.save')

print("✅ Train-test split completed")
print("✅ Features scaled using MinMaxScaler")
print("✅ Scaler saved as transform.save")
```

```
✅ Train-test split completed
✅ Features scaled using MinMaxScaler
✅ Scaler saved as transform.save
```

RANDOM FOREST ALGORITHM

```
from sklearn.ensemble import RandomForestRegressor

rf_model = RandomForestRegressor(
    n_estimators=100,        # reduced
    max_depth=15,            # reduced
    min_samples_split=10,    # increased
    min_samples_leaf=4,      # increased
    random_state=42,
    n_jobs=-1
)

rf_model.fit(X, y)
```

```
RandomForestRegressor(max_depth=15, min_samples_leaf=4, min_samples_split=10,
                      n_jobs=-1, random_state=42)
```

PERFORMANCE OF MODEL && IMPORTING MODEL,TRANSOFORM.SAVE TO VSCODE

```
y_pred = rf_model.predict(X_df1_test)

mae = mean_absolute_error(y_df1_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_df1_test, y_pred))
r2 = r2_score(y_df1_test, y_pred)

print("MAE :", mae)
print("RMSE:", rmse)
print("R²  :", r2)
```

```
MAE : 2.003360502530888
RMSE: 3.5835339248473734
R²  : 0.9644516828541994
```

```
accuracy = r2 * 100
print(f"Model Accuracy (R²): {accuracy:.2f}%")
```

```
Model Accuracy (R²): 96.45%
```

```
joblib.dump(rf_model, "model.save")
```

```
['model.save']
```

```
from google.colab import files

files.download("model.save")
files.download("transform.save")
```

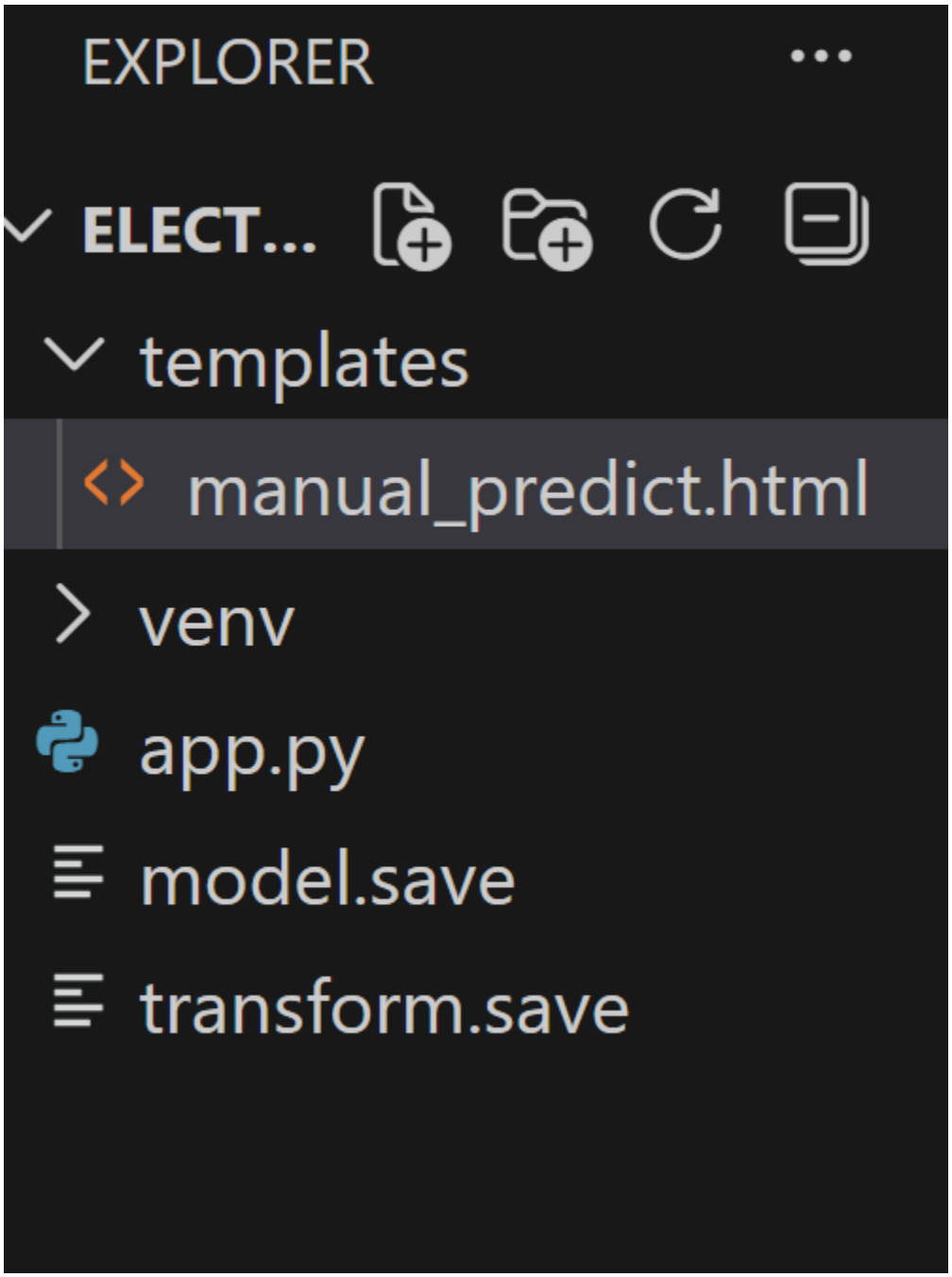
---

4. Project Structure in VS Code

Once downloaded, the files are moved to the Flask project directory.

Project Folder Structure

📸 **PROJECT FOLDER**



---

5. Flask Application (Backend)

5.1 Flask App Initialization

```
from flask import Flask, request, render_template
import joblib
import numpy as np
app = Flask(__name__)
model = joblib.load("model.save")
scaler = joblib.load("transform.save")
```

5.2 Home Route

Displays the input form:

```
@app.route('/')
def home():
    return render_template('Manual_predict.html')
```

5.3 Prediction Route

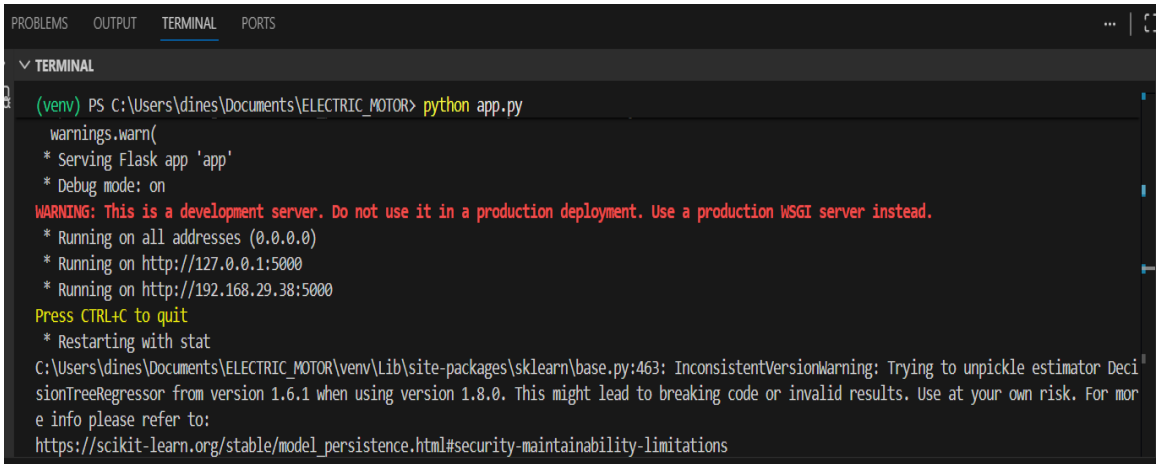
Handles user input, scaling, and prediction:

```
@app.route('/y_predict', methods=['POST'])
def y_predict():
    values = [float(x) for x in request.form.values()]
    x_test = np.array(values).reshape(1, -1)
    x_test_scaled = scaler.transform(x_test)
    prediction = model.predict(x_test_scaled)
    return render_template(
        'Manual_predict.html',
        prediction_text=f"Permanent Magnet Surface Temperature: {prediction[0]:.2f} °C"
    )
```

5.4 Running the Application

```
if __name__ == "__main__":
    app.run(debug=True)
```

TERMINAL OF VSCODE



---

6. HTML Page (Frontend)

Manual_predict.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Electric Motor Temperature Prediction</title>
  <style>
    * {
      margin: 0;
      padding: 0;
      box-sizing: border-box;
    }
```

```
body {
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
    min-height: 100vh;
    display: flex;
    align-items: center;
    justify-content: center;
    padding: 20px;
}

.container {
    background: white;
    border-radius: 15px;
    box-shadow: 0 20px 60px rgba(0, 0, 0, 0.3);
    padding: 40px;
    max-width: 500px;
    width: 100%;
    animation: slideIn 0.5s ease-out;
}

@keyframes slideIn {
    from {
        opacity: 0;
        transform: translateY(-20px);
    }
```

```
    to {
        opacity: 1;
        transform: translateY(0);
    }
}

h1 {
    text-align: center;
    color: #333;
    margin-bottom: 10px;
    font-size: 28px;
}

.subtitle {
    text-align: center;
    color: #666;
    margin-bottom: 30px;
    font-size: 14px;
}

form {
    display: flex;
    flex-direction: column;
    gap: 15px;
}
```

```
.form-group {
    display: flex;
    flex-direction: column;
}

label {
    font-weight: 600;
    color: #333;
    margin-bottom: 8px;
    font-size: 14px;
}

input[type="text"] {
    padding: 12px 15px;
    border: 2px solid #e0e0e0;
    border-radius: 8px;
    font-size: 15px;
    transition: all 0.3s ease;
    font-family: inherit;
}

input[type="text"]:focus {
    outline: none;
    border-color: #667eea;
    box-shadow: 0 0 0 3px rgba(102, 126, 234, 0.1);
    background-color: #fafafa;
```

```
}

input[type="text"]::placeholder {
    color: #999;
}

button {
    padding: 13px 30px;
    background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
    color: white;
    border: none;
    border-radius: 8px;
    font-size: 16px;
    font-weight: 600;
    cursor: pointer;
    transition: all 0.3s ease;
    margin-top: 10px;
    text-transform: uppercase;
    letter-spacing: 0.5px;
}

button:hover {
    transform: translateY(-2px);
    box-shadow: 0 10px 25px rgba(102, 126, 234, 0.4);
}
```

```
button:active {
    transform: translateY(0);
}

.prediction-result {
    margin-top: 30px;
    padding: 20px;
    background: #f0f4ff;
    border-left: 4px solid #667eea;
    border-radius: 8px;
    display: none;
}

.prediction-result.show {
    display: block;
    animation: fadeIn 0.5s ease-out;
}

@keyframes fadeIn {
    from {
        opacity: 0;
    }
    to {
        opacity: 1;
    }
}
```

```
      .prediction-result h2 {
        color: #667eea;
        font-size: 18px;
        margin: 0;
      }
    </style>
</head>
<body>
    <div class="container">
      <h1> ⚡ Motor Temperature Prediction</h1>
      <p class="subtitle">Predict motor temperature using ML model</p>

      <form action="/y_predict" method="post">
        <div class="form-group">
          <label for="ambient">Ambient Temperature (°C)</label>
          <input type="text" id="ambient" name="ambient" placeholder="e.g., 25" required>
        </div>

        <div class="form-group">
          <label for="coolant">Coolant Temperature (°C)</label>
          <input type="text" id="coolant" name="coolant" placeholder="e.g., 30" required>
        </div>

        <div class="form-group">
          <label for="u_d">Voltage d-component (V)</label>
```

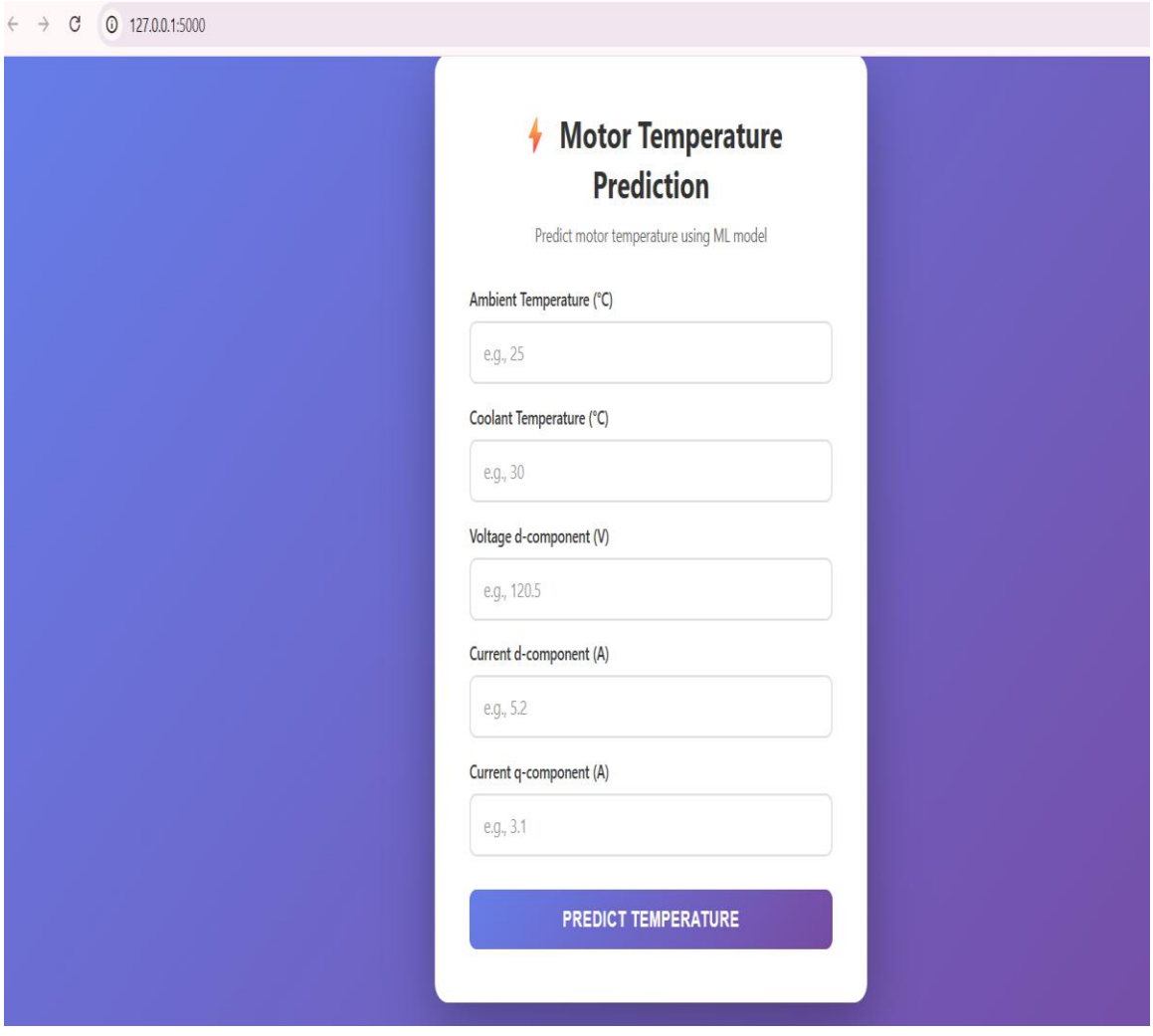
```
            <input type="text" id="u_d" name="u_d" placeholder="e.g., 120.5" required>
        </div>

        <div class="form-group">
            <label for="i_d">Current d-component (A)</label>
            <input type="text" id="i_d" name="i_d" placeholder="e.g., 5.2" required>
        </div>

        <div class="form-group">
            <label for="i_q">Current q-component (A)</label>
            <input type="text" id="i_q" name="i_q" placeholder="e.g., 3.1" required>
        </div>

        <button type="submit">Predict Temperature</button>
    </form>

    {% if prediction_text %}
    <div class="prediction-result show">
        <h2>{{ prediction_text }}</h2>
    </div>
    {% endif %}
  </div>
</body>
</html>
```

7. End-to-End Execution Process

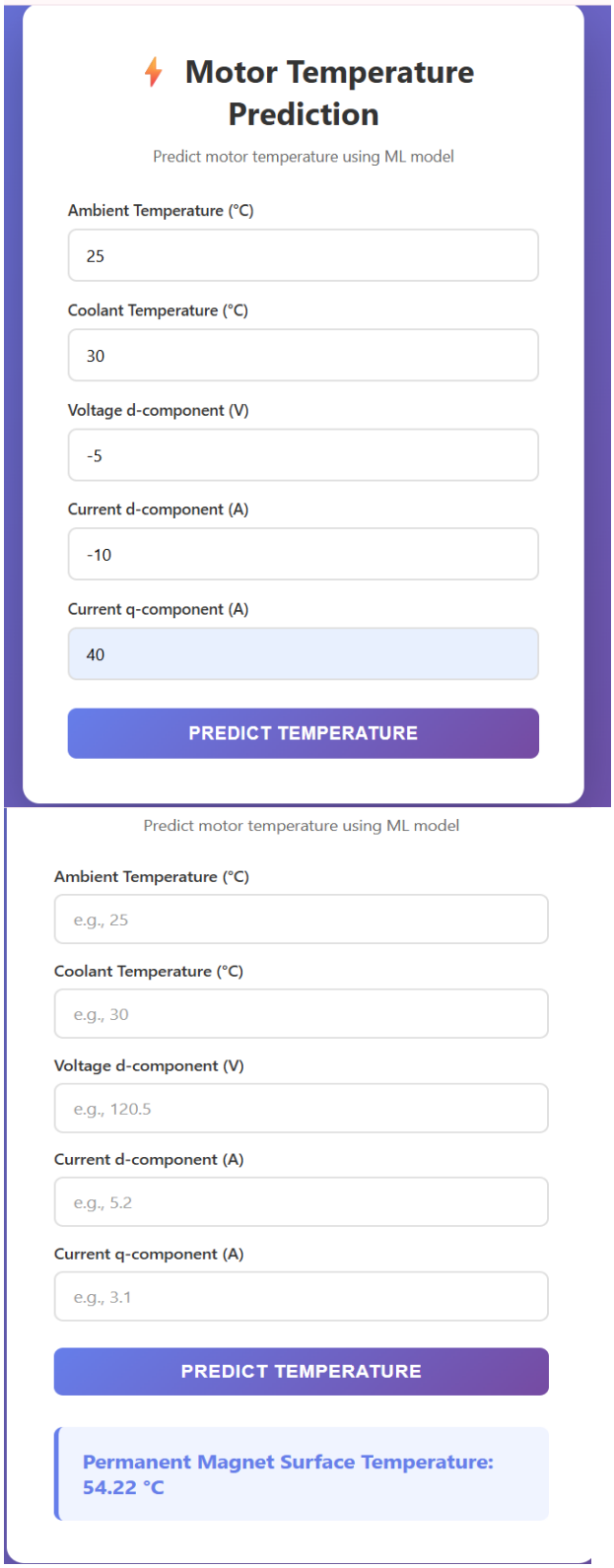
Step-by-Step Execution

1. User enters input values in browser
2. Data sent to Flask via POST request
3. Flask loads scaler and model
4. Input is scaled using MinMaxScaler
5. Random Forest model predicts temperature
6. Result displayed on the same page

WEB PAGE



WEB PAGE OUTPUT



---

8. Advantages of This Deployment Approach

- Lightweight and fast
- Easy to understand and modify
- Suitable for academic and demo projects
- Can be extended to cloud deployment (AWS, Azure)

---

9. Future Enhancements

- Cloud deployment (AWS / Heroku)
- Real-time sensor data integration
- REST API support
- Model retraining pipeline

Conclusion

The trained Random Forest model was successfully deployed using Flask, enabling real-time prediction of electric motor temperature. This deployment bridges machine learning and web development, providing an interactive and practical solution for predictive maintenance.

.