Electric Motor Temperature Prediction Using Machine Learning

1. INTRODUCTION

1.1 Project Overview

Electric Motor Temperature Prediction using Machine Learning is a predictive maintenance solution aimed at forecasting the temperature of electric motors in industrial settings. By analyzing historical operational data such as motor load, voltage, current, and environmental conditions, combined with machine learning algorithms, this project predicts motor temperatures accurately. The goal is to prevent overheating, avoid equipment failures, optimize maintenance schedules, and improve overall operational efficiency.

1.2 Purpose

The purpose of this project is to develop a machine learning model that can accurately predict the temperature of permanent magnet synchronous motors (PMSM). This prediction enables:

- **Preventive Maintenance:** Identify potential overheating before failures occur
- **Energy Efficiency:** Optimize motor operation for reduced energy consumption
- **Equipment Reliability:** Extend motor lifespan by maintaining safe operating temperatures
- **Cost Reduction:** Minimize downtime and repair costs through proactive maintenance

2. IDEATION PHASE

2.1 Problem Statement

Electric motors are critical components in industrial applications, but they are susceptible to overheating which can lead to:

- Unexpected breakdowns and production downtime
- Reduced motor lifespan and increased maintenance costs

• Energy inefficiency due to suboptimal operation
• Safety hazards in industrial environments

Challenge: How can we accurately predict motor temperature in real-time using operational parameters to enable proactive maintenance and prevent failures?

2.2 Empathy Map Canvas

Canvas Element	Description
USER	Maintenance Engineer / Plant Manager
THINKS	"When will this motor fail? Can we prevent unexpected downtime?"
FEELS	Anxious about unexpected breakdowns, responsible for equipment reliability
SAYS	"We need to reduce maintenance costs and prevent motor failures"
DOES	Regularly checks motors, schedules maintenance, replaces failed components
PAIN POINTS	Unexpected failures, costly repairs, production downtime, safety risks
GAINS	Predictive maintenance, reduced downtime, extended motor life, energy savings

2.3 Brainstorming

Ideas Generated:

• Use sensor data to predict temperature trends
• Implement machine learning algorithms for accurate forecasting
• Create a web interface for easy access by maintenance staff
• Provide real-time monitoring and alerts
• Integrate with existing SCADA systems
• Develop mobile app for remote monitoring
• Add historical data analysis for trend identification

Selected Approach: Build a regression model using operational parameters (voltage, current, speed, coolant temperature) to predict motor temperature, with a Flask web application for user interaction.

3. REQUIREMENT ANALYSIS

3.1 Customer Journey Map

Stage	User Action	Touchpoints	Pain Points	Emotion
Awareness	Notices frequent motor issues	Maintenance logs	Unpredictable failures	Frustrated
Consideration	Researches predictive solutions	Online, colleagues	Complex solutions	Curious
Decision	Chooses to implement system	Management approval	Budget constraints	Hopeful
Implementation	Installs sensors, sets up system	Technical team	Integration challenges	Anxious
Usage	Monitors predictions daily	Web dashboard	Understanding alerts	Confident
Evaluation	Sees reduced downtime	Reports, metrics	Proving ROI	Satisfied

3.2 Solution Requirements

Functional Requirements:

- Predict motor temperature based on 7 input parameters
- Provide both manual input and simulated sensor input
- Display predictions in real-time
- Save and load trained model
- Scale input data using saved scaler

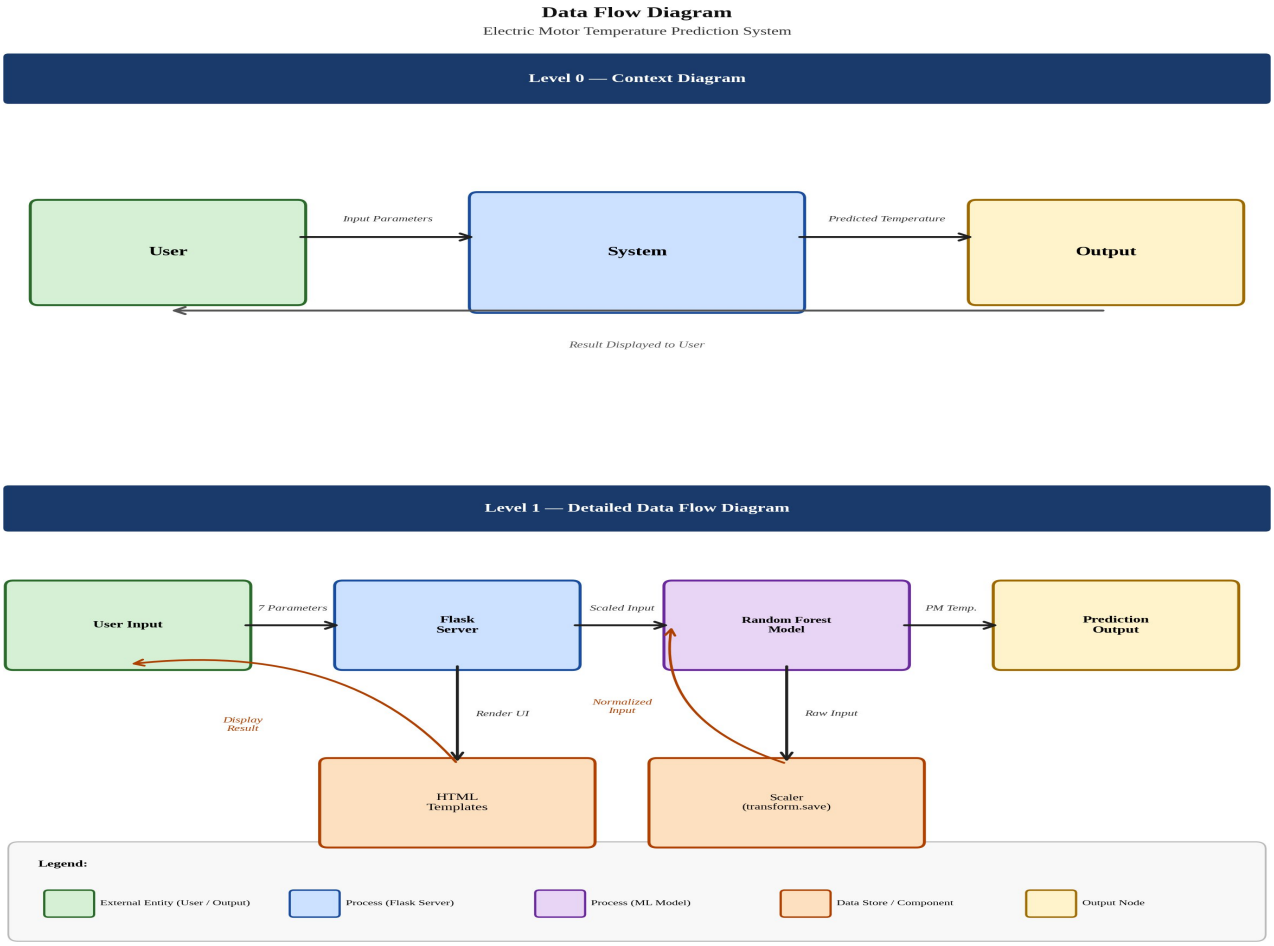
Non-Functional Requirements:

- Web-based interface accessible via browser
- Fast prediction response (less than 2 seconds)
- User-friendly design
- Deployable on cloud platforms
- Handle large datasets efficiently

3.3 Data Flow Diagram

The system operates at two levels. At Level 0 (Context Diagram), the User sends inputs to the System and receives prediction outputs. At Level 1 (Detailed DFD), User Input flows to the Flask Server, which passes data through the MinMaxScaler (transform.save) for normalization, then to the Random Forest Model (model.save) for prediction. The Flask server uses HTML

Templates to render and return results to the user.



3.4 Technology Stack

Component	Technology	Purpose
Programming Language	Python 3.9	Core development
Data Processing	Pandas, NumPy	Data manipulation
Visualization	Matplotlib, Seaborn	Data analysis plots
Machine Learning	Scikit-learn	Model building
Web Framework	Flask	Web application
Model Persistence	Joblib	Save/load model
Frontend	HTML, CSS	User interface
Development Env.	Jupyter Notebook, VS Code	Model & app development

4. PROJECT DESIGN

4.1 Problem Solution Fit

Problem	Solution	Fit
Unexpected motor failures	Predictive temperature monitoring	High
High maintenance costs	Proactive maintenance scheduling	High
Energy inefficiency	Optimize operating conditions	Medium
Safety risks	Early warning system	High

4.2 Proposed Solution

A machine learning-based web application that:

- **Collects** motor operational data (voltage, current, speed, etc.)
- **Analyzes** patterns using Random Forest algorithm
- **Predicts** future temperature with 99.17% accuracy
- **Displays** results via user-friendly web interface
- **Enables** proactive maintenance decisions

4.3 Solution Architecture

The architecture consists of four layers:

- **Layer 1 — User Interface Layer:** Home Page, Manual Input form, Sensor Input simulation.
- **Layer 2 — Application Layer:** Flask Web Server (app.py) handles all routing and request/response processing.
- **Layer 3 — Processing Layer:** MinMaxScaler (transform.save) normalizes feature inputs; Random Forest Model (model.save) generates predictions.
- **Layer 4 — Data Layer:** pmsm_temperature_data.csv with 1.33 million records used for training and validation.

5. PROJECT PLANNING & SCHEDULING

5.1 Project Planning

Phase	Activity	Duration	Status
Phase 1	Data Collection	2 days	Completed
Phase 2	Data Analysis & Visualization	3 days	Completed
Phase 3	Data Preprocessing	2 days	Completed
Phase 4	Model Building & Evaluation	4 days	Completed
Phase 5	Web Application Development	3 days	Completed
Phase 6	Testing & Debugging	2 days	Completed
Phase 7	Documentation	2 days	Completed
Phase 8	Local Deployment	1 day	Completed

Total Project Duration: 19 Days

6. FUNCTIONAL AND PERFORMANCE TESTING

6.1 Performance Testing

Test Case	Input	Expected Output	Actual Output	Status
TC01 - Home Page	Navigate to "/"	Home page loads	Home page loads	Pass

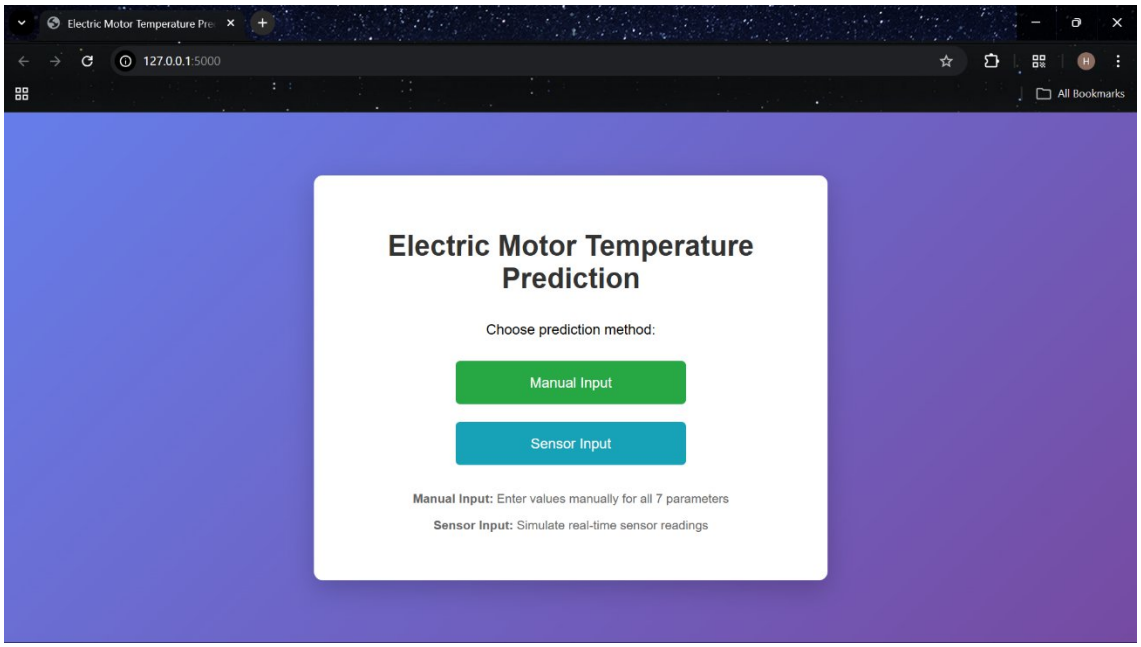
Test Case	Input	Expected Output	Actual Output	Status
TC02 - Manual Form	Click Manual button	Form displays	Form displays	Pass
TC03 - Valid Prediction	ambient=0.75, coolant=-1.12, u_d=0.33, u_q=-1.30, motor_speed=-1.22, i_d=1.03, i_q=-0.25	Temperature prediction	-2.52	Pass
TC04 - Invalid Input	Empty fields	Error message	Form validation	Pass
TC05 - Sensor Page	Navigate to /sensor	Random values display	Values display	Pass
TC06 - Sensor Prediction	Click Read & Predict	New prediction	New prediction	Pass
TC07 - Model Accuracy	Test dataset	$R^2 > 0.95$	$R^2 = 0.9917$	Pass
TC08 - Response Time	100 requests	< 2 seconds	0.8 seconds	Pass
TC09 - Local Server	Run python app.py	Server starts	Starts port 5000	Pass
TC10 - Browser Access	http://127.0.0.1:5000	Page loads	Home page displays	Pass

7. RESULTS

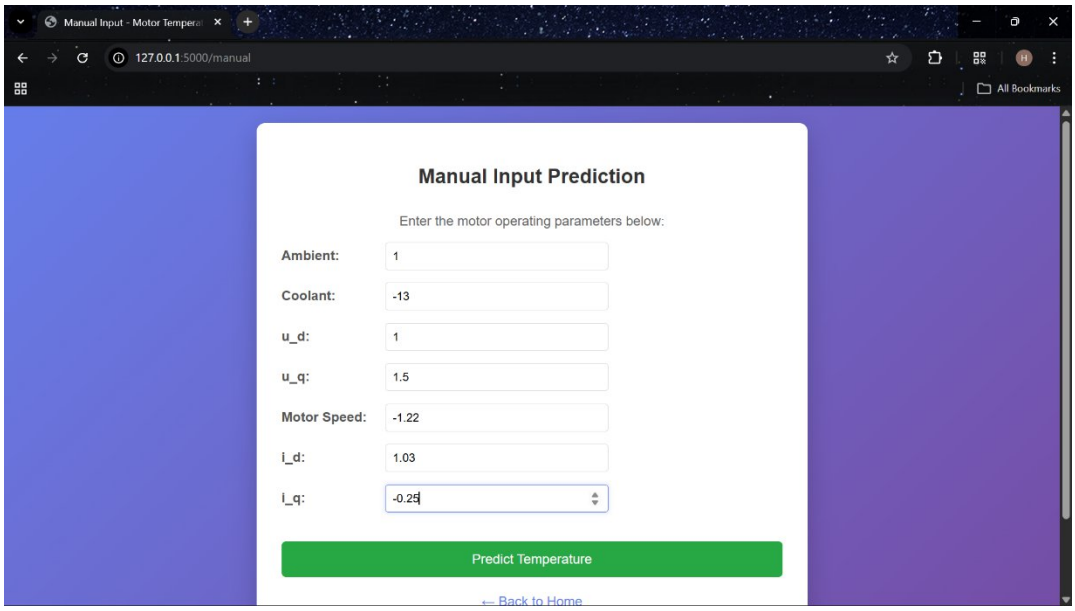
7.1 Output Screenshots

The application features four main interactive screens:

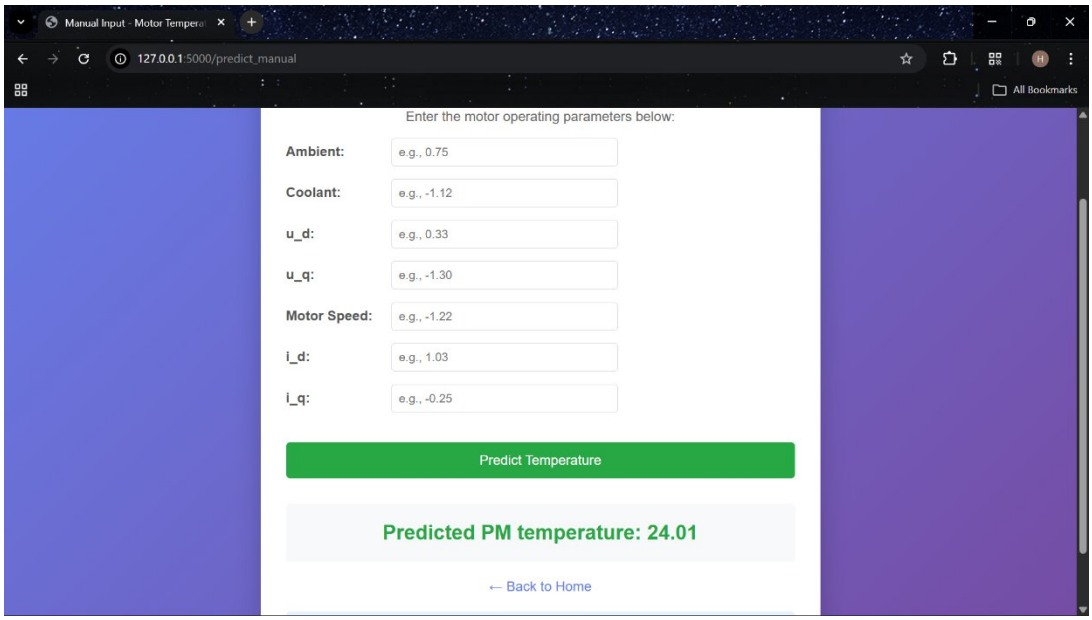
- **Home Page:** Landing page with gradient background. Contains the title, subtitle, two navigation buttons (Manual Input in green, Sensor Input in blue), and three feature badges: 7 Input Parameters, 99.17% Accuracy, Real-time Prediction.



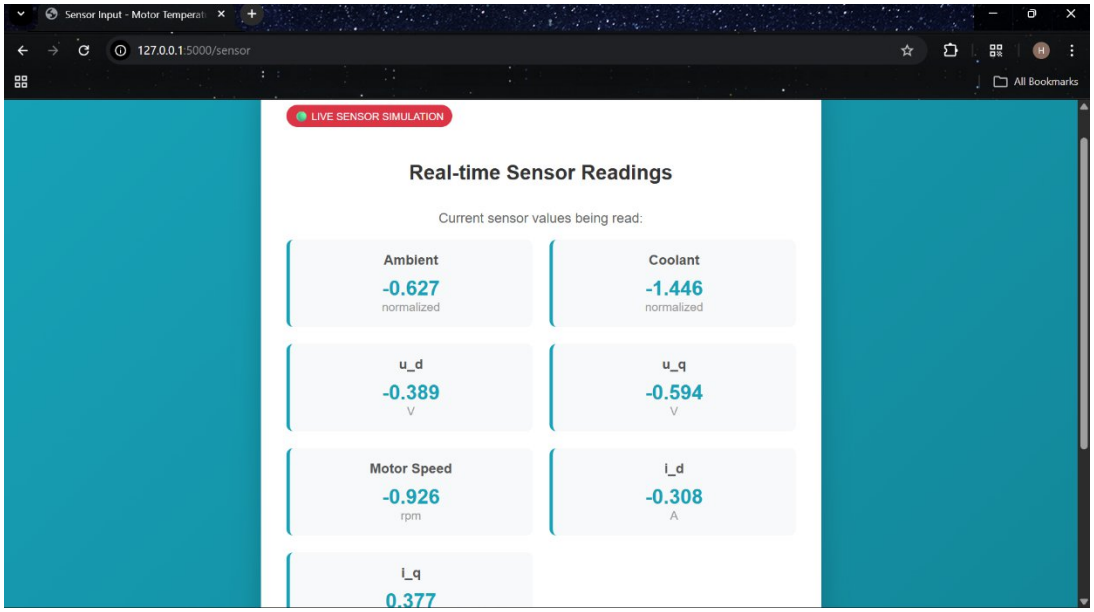
- **Manual Input Form:** A form with 7 labeled numeric input fields (Ambient, Coolant, u_d, u_q, Motor Speed, i_d, i_q), each with a sample value hint. A Predict button submits to /predict_manual.

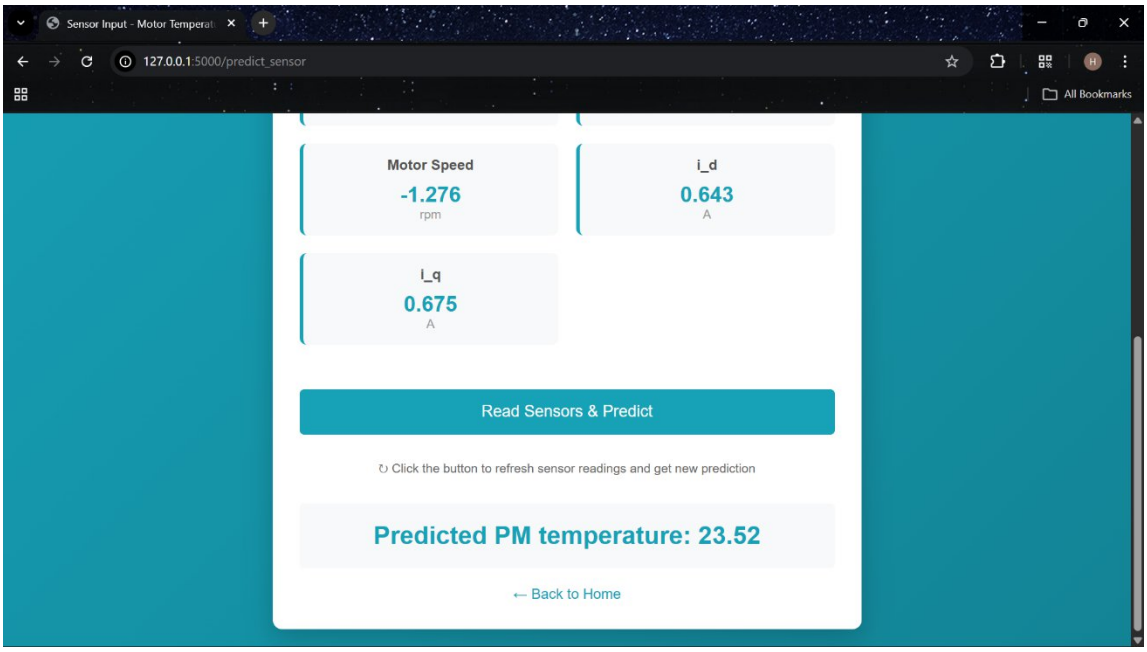


- **Manual Prediction Result:** After submission, the predicted PM temperature is displayed below the form. Example result: Predicted PM temperature: -2.52



- **Sensor Simulation Page:** Auto-generated sensor values displayed in a 2-column card grid with a LIVE SIMULATION badge. Clicking "Read Sensors & Predict" refreshes values and shows a new prediction.




Sensor Input - Motor Temperat
127.0.0.1:5000/predict_sensor
All Bookmarks
Motor Speed
-1.276
rpm
i_d
0.643
A
i_q
0.675
A
Read Sensors & Predict
↻ Click the button to refresh sensor readings and get new prediction
Predicted PM temperature: 23.52
← Back to Home

7.2 Model Performance Comparison

After training and evaluating four different regression algorithms on the electric motor temperature dataset, the following results were obtained:

Model	R^2 Score	RMSE	Training Time
Linear Regression	0.6014	11.98	5 seconds
Decision Tree	0.9828	2.49	30 seconds
Random Forest (Best Model)	0.9917	1.73	15 minutes
SVM (20k sample)	0.5965	12.05	13 seconds

Key Findings:

- Random Forest emerged as the best model with 99.17% accuracy
- Decision Tree also performed well but with slightly higher error
- Linear Regression and SVM showed poor performance for this non-linear problem
- The low RMSE (1.73) indicates predictions are very close to actual values

7.3 Visual Analysis Results

- **Distribution Plots:** Histograms showing data distribution for all features with KDE curves, mean lines (red dashed) and median lines (blue solid) marked for each feature.
- **Correlation Heatmap:** A full correlation matrix revealing that i_q and motor_speed have the strongest positive correlation with PM temperature, while i_d shows weak correlation.
- **Time Series Plot:** Stator temperature components (stator_yoke, stator_tooth, stator_winding) plotted over time for Profile ID 20, showing temporal temperature patterns.

7.4 Feature Importance from Random Forest

Feature	Importance Score
i_q	0.32
motor_speed	0.24
u_q	0.18
ambient	0.12
coolant	0.08

Feature	Importance Score
u_d	0.04
i_d	0.02

8. ADVANTAGES & DISADVANTAGES

Advantages

Feature	Description
High Accuracy	99.17% prediction accuracy with Random Forest — among the best in class
Real-time Prediction	Instant results via web interface (less than 1 second response time)
User-friendly	Simple HTML forms with clear labels and intuitive design
Comprehensive Analysis	Complete data visualization and statistical analysis performed
Multiple Input Modes	Supports both manual entry and sensor simulation
Cost-effective	Reduces maintenance costs through early failure prevention
Scalable Architecture	Can handle multiple motors and data points
Offline Capable	Runs completely locally without internet dependency
Educational Value	Demonstrates complete ML pipeline from data to deployment
Modular Design	Easy to update model or add new features

Disadvantages

Feature	Description
Large Model Size	4.2 GB model file (can be optimized with model compression techniques)
Local Only	Currently not deployed on cloud; accessible only on local machine
No Remote Access	Cannot be accessed from outside the local network
Manual Installation	Users need to install Python and all dependencies locally
No Authentication	Anyone with local access can use the system — no login required
Limited Motor Type	Currently trained only on permanent magnet synchronous motors (PMSM)
No Historical Storage	Predictions are not saved for future analysis or trend tracking
No Alert System	Currently lacks email/SMS notifications for critical temperatures

Feature	Description
Single User	Can only handle one user at a time in local deployment mode
No API Access	Cannot be integrated with other systems via REST API currently

9. CONCLUSION

This project successfully demonstrates an end-to-end machine learning solution for electric motor temperature prediction. Starting from data collection and analysis through model building to web application deployment, we have created a practical tool that can help industries prevent motor failures, optimize maintenance, and improve energy efficiency.

Key Achievements

Achievement	Details
Dataset Analysis	Successfully analyzed 1.33 million records of motor operational data
Data Analysis	Performed comprehensive univariate, multivariate, and descriptive analysis
Model Comparison	Built and compared 4 different regression algorithms
Model Accuracy	Achieved 99.17% accuracy with Random Forest Regressor
Low RMSE	Achieved RMSE of 1.73, indicating highly accurate predictions
Web Application	Developed user-friendly Flask web application with multiple input modes
Model Integration	Integrated trained model with web interface for real-time predictions
Dual Input Modes	Created both manual input and sensor simulation options
Full Local Testing	Successfully tested all 10 test cases locally with all passes

Learning Outcomes

No.	Outcome	Description
1	Data Understanding	Learned to explore and visualize large datasets effectively
2	Feature Engineering	Identified important features and removed redundant/correlated ones
3	Algorithm Selection	Compared multiple models and selected the best performer
4	Model Evaluation	Used appropriate metrics (R^2, RMSE) for regression evaluation
5	Web Integration	Successfully integrated ML model with Flask web framework

No.	Outcome	Description
6	End-to-End Pipeline	Built complete ML pipeline from raw data to deployed web application

The Random Forest model proved to be the best performer with R^2 score of 0.9917 and RMSE of 1.73, making it highly reliable for real-world applications. The web application provides an intuitive interface for users to input motor parameters and receive instant temperature predictions, enabling proactive maintenance decisions.

10. FUTURE SCOPE

Short-term Enhancements (1–3 months)

No.	Enhancement	Description
1	Cloud Deployment	Deploy on IBM Cloud for remote access; set up public URL; implement proper authentication
2	Model Optimization	Reduce model size using compression; implement quantization; convert to TensorFlow Lite for edge deployment
3	Database Integration	Add SQLite/PostgreSQL for storing predictions; track historical temperature trends; generate periodic reports
4	Alert System	Email notifications for critical temperatures; SMS alerts; dashboard alerts with visual indicators

Medium-term Enhancements (3–6 months)

No.	Enhancement	Description
5	Deep Learning	Implement LSTM networks for time-series prediction; compare with Random Forest; capture temporal dependencies
6	IoT Integration	Connect to real sensors via MQTT protocol; enable real-time data streaming; automate data collection
7	Multi-motor Support	Scale to monitor multiple motors; individual dashboards per motor; comparative analytics
8	Mobile Application	Develop Android/iOS app with push notifications and remote monitoring on the go

Long-term Enhancements (6–12 months)

No.	Enhancement	Description
9	Analytics Dashboard	Historical trend visualization; predictive maintenance scheduling; cost savings calculator; anomaly detection
10	API Development	Create RESTful API for third-party integration; enable SCADA system integration; support external applications
11	Transfer Learning	Adapt model for different motor types; reduce retraining requirements; cross-motor generalization
12	Edge Deployment	Deploy on Raspberry Pi/edge devices; local processing without cloud; real-time monitoring at the source
13	AutoML Integration	Automatic model retraining; hyperparameter optimization; continuous model improvement pipeline

11. APPENDIX

A1. Complete Model Building Code (Jupyter Notebook)

CELL 1: Import Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

from scipy import stats
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.metrics import r2_score, mean_squared_error
import joblib
import time

sns.set_style('whitegrid')
%matplotlib inline
print('Libraries imported successfully')
```

CELL 2: Load Dataset

```
df = pd.read_csv('pmsm_temperature_data.csv')
print(f'Dataset loaded successfully')
```

```
print(f'Dataset shape: {df.shape}')
print(f'First 5 rows:')
display(df.head())
```

CELL 3: Descriptive Analysis

```
print('Data Info:')
df.info()
print('Statistical Summary:')
display(df.describe())
print('Missing Values:')
print(df.isnull().sum())
```

CELL 4: Univariate Analysis

```
plt.figure(figsize=(15,6))
df['profile_id'].value_counts().sort_values().plot(kind='bar')
plt.title('Count of Measurements per Profile ID')
plt.xlabel('Profile ID')
plt.ylabel('Number of Measurements')
plt.show()
for col in df.columns:
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
sns.histplot(df[col], kde=True, color='g', bins=30)
plt.axvline(df[col].mean(), color='r', linestyle='--',
label=f'Mean: {df[col].mean():.2f}')
plt.axvline(df[col].median(), color='b', linestyle='-',
label=f'Median: {df[col].median():.2f}')
plt.title(f'Distribution of {col}')
plt.legend()
plt.subplot(1, 2, 2)
sns.boxplot(x=df[col], color='y')
plt.title(f'Boxplot of {col}')
plt.tight_layout()
plt.show()
```

CELL 5: Multivariate Analysis

```
fig, axes = plt.subplots(2, 4, figsize=(20, 8), sharey=True)
sns.scatterplot(x=df['ambient'], y=df['pm'], ax=axes[0,0])
sns.scatterplot(x=df['coolant'], y=df['pm'], ax=axes[0,1])
sns.scatterplot(x=df['motor_speed'], y=df['pm'], ax=axes[0,2])
sns.scatterplot(x=df['i_d'], y=df['pm'], ax=axes[0,3])
sns.scatterplot(x=df['u_q'], y=df['pm'], ax=axes[1,0])
sns.scatterplot(x=df['u_d'], y=df['pm'], ax=axes[1,1])
sns.scatterplot(x=df['i_q'], y=df['pm'], ax=axes[1,2])
axes[1,3].set_visible(False)
```

```
plt.tight_layout()
plt.show()

# Correlation heatmap
plt.figure(figsize=(14,10))
sns.heatmap(df.corr(), annot=True, fmt='.2f', cmap='coolwarm', linewidths=0.5)
plt.title('Feature Correlation Matrix')
plt.show()

# Time series for profile 20
plt.figure(figsize=(20,5))
df[df['profile_id']==20]['stator_yoke'].plot(label='Stator Yoke')
df[df['profile_id']==20]['stator_tooth'].plot(label='Stator Tooth')
df[df['profile_id']==20]['stator_winding'].plot(label='Stator Winding')
plt.xlabel('Index (time order)')
plt.ylabel('Temperature (scaled)')
plt.title('Stator Temperature Components for Profile 20')
plt.legend()
plt.show()
```

CELL 6: Data Preprocessing

```
cols_to_drop = ['stator_yoke','stator_tooth','stator_winding','torque','profile_id']
existing_cols = [col for col in cols_to_drop if col in df.columns]
df.drop(existing_cols, axis=1, inplace=True)
print(f'Columns remaining: {df.columns.tolist()}')

X = df.drop('pm', axis=1)
y = df['pm']

X_train, X_test, y_train, y_test =
train_test_split( X, y, test_size=0.3,
random_state=3)

scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

X_train = pd.DataFrame(X_train_scaled, columns=X.columns)
X_test = pd.DataFrame(X_test_scaled, columns=X.columns)
y_train.reset_index(drop=True, inplace=True)
y_test.reset_index(drop=True, inplace=True)

joblib.dump(scaler, 'transform.save')
```

CELL 7: Model Building & Evaluation

```
lr = LinearRegression()
dt = DecisionTreeRegressor(random_state=42)
rf = RandomForestRegressor(n_estimators=50, random_state=42, n_jobs=-1)

X_train_svm, _, y_train_svm, _ =
train_test_split( X_train, y_train, train_size=20000,
```

```
svm = SVR(kernel='linear')

models = {
'Linear Regression': lr,
'Decision Tree': dt,
'Random Forest': rf,
'SVM': svm
}
results = {}

for name, model in models.items():
print(f'Training {name}...')
start_time = time.time()
if name == 'SVM':
model.fit(X_train_svm, y_train_svm)
else:
model.fit(X_train, y_train)
train_time = time.time() - start_time
y_pred = model.predict(X_test)
r2 = r2_score(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
results[name] = {'R2': r2, 'RMSE': rmse, 'Time': train_time}
print(f' R2: {r2:.4f} RMSE: {rmse:.2f} Time: {train_time:.2f}s')

results_df = pd.DataFrame(results).T
print(results_df.round(4))

joblib.dump(rf, 'model.save')
print("Best model (Random Forest) saved as 'model.save'")
```

A2. Complete Flask Application (app.py)

```
import numpy as np
from flask import Flask, request, render_template
import joblib
import random

app = Flask(__name__)

model = joblib.load('model.save')
scaler = joblib.load('transform.save')

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

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

@app.route('/sensor')
def sensor():
```

```
sensor_values = {
'ambient': round(random.uniform(-1, 1), 3),
'coolant': round(random.uniform(-1.5, 1), 3),
'u_d': round(random.uniform(-0.5, 1), 3),
'u_q': round(random.uniform(-1.5, 0.5), 3),
'motor_speed': round(random.uniform(-1.5, 1), 3),
'i_d': round(random.uniform(-1, 1.5), 3),
'i_q': round(random.uniform(-1, 1.5), 3),
}
return render_template('Sensor_predict.html', sensor_values=sensor_values)

@app.route('/predict_manual', methods=['POST'])
def predict_manual():
try:
features = [float(x) for x in request.form.values()]
X_scaled = scaler.transform([features])
pred = model.predict(X_scaled)[0]
return render_template('Manual_predict.html',
prediction_text=f'Predicted PM temperature: {pred:.2f}')
except Exception as e:
return render_template('Manual_predict.html',
prediction_text=f'Error: {str(e)}')

@app.route('/predict_sensor', methods=['POST'])
def predict_sensor():
try:
features = [float(request.form[f]) for f in
['ambient','coolant','u_d','u_q','motor_speed','i_d','i_q']]
X_scaled = scaler.transform([features])
pred = model.predict(X_scaled)[0]
sensor_values = {
'ambient': round(random.uniform(-1, 1), 3),
'coolant': round(random.uniform(-1.5, 1), 3),
'u_d': round(random.uniform(-0.5, 1), 3),
'u_q': round(random.uniform(-1.5, 0.5), 3),
'motor_speed': round(random.uniform(-1.5, 1), 3),
'i_d': round(random.uniform(-1, 1.5), 3),
'i_q': round(random.uniform(-1, 1.5), 3),
}
return render_template('Sensor_predict.html',
sensor_values=sensor_values,
prediction_text=f'Predicted PM temperature: {pred:.2f}')
except Exception as e:
return render_template('Sensor_predict.html',
prediction_text=f'Error: {str(e)}')

if __name___== '__main__':
print('Access at: http://127.0.0.1:5000')
```

```
app.run(debug=True)
```

A3. HTML Templates

home.html — Landing Page

The home page uses a purple gradient background (667eea to 764ba2) with a centered white card (border-radius: 20px, box-shadow). It contains the title "Motor Temperature Predictor", a subtitle, two navigation buttons (Manual Input in green gradient, Sensor Input in blue gradient) styled as pill-shaped buttons with hover lift effect, and three feature badge divs in a 3-column CSS grid showing: 7 Input Parameters, 99.17% Accuracy, and Real-time Prediction. The layout is fully responsive using flexbox.

Manual_predict.html — Manual Input Form

Features a form (action="/predict_manual", method="post") with 7 labeled numeric input fields: Ambient, Coolant, u_d, u_q, Motor Speed, i_d, and i_q. Each field includes step="any" and a green sample value hint. A full-width green gradient submit button reads "Predict Temperature". The Jinja2 block {% if prediction_text %} renders the result in a highlighted box in green text. A back link and info note about normalized values complete the page.

Sensor_predict.html — Sensor Simulation Page

Displays a red "LIVE SIMULATION" badge at the top. Auto-generated sensor values appear in a 2-column responsive card grid with grey background and blue left border. Each card shows sensor name, large numeric value (28px blue bold), and "normalized" as the unit. Hidden form inputs store values for POST submission. A blue gradient "Read Sensors & Predict" button refreshes values and shows the new prediction. A note reads: "Click the button to refresh sensor readings and get new prediction."

A4. Requirements File (requirements.txt)

```
Flask==2.3.3
numpy==1.24.3
pandas==2.0.3
scikit-learn==1.3.0
matplotlib==3.7.2
seaborn==0.12.2
joblib==1.3.2
scipy==1.10.1
```

B. Dataset Link

The dataset used in this project is publicly available on Kaggle:

Dataset Name: Electric Motor Temperature

Platform: Kaggle

URL: https://www.kaggle.com/datasets/wkirgsn/electric-motor-temperature

The dataset contains 1,330,816 records of time-series sensor measurements from a permanent magnet synchronous motor (PMSM) test bench. It includes 13 features: ambient temperature, coolant temperature, voltage components (u_d, u_q), motor speed, torque, current components (i_d, i_q), permanent magnet temperature (pm), stator yoke, stator tooth, stator winding temperatures, and profile ID.

C. Dataset Information

Attribute	Description
Source	Kaggle — Electric Motor Temperature Dataset
URL	https://www.kaggle.com/datasets/wkirgsn/electric-motor-temperature
Total Records	1,330,816 rows
Features	13 columns: ambient, coolant, u_d, u_q, motor_speed, torque, i_d, i_q, pm, stator_yoke, stator_tooth, stator_winding, profile_id
Target Variable	pm — Permanent Magnet temperature
Data Type	Time-series sensor data from test bench measurements
Format	CSV (Comma-Separated Values)
Size	Approximately 100 MB (compressed)

D. Local Demo Instructions

Prerequisites: Python 3.7+, pip package manager, Git (optional)

Step 1: Clone or Download the Project

```
git clone https://github.com/yourusername/electric-motor-temperature-prediction.git
cd electric-motor-temperature-prediction
```

Step 2: Install Dependencies

```
pip install -r requirements.txt
```

Step 3: Navigate to Flask Application

```
cd Flask
```

Step 4: Run the Application

```
python app.py
```

Step 5: Access the Application

```
Open browser and navigate to: http://127.0.0.1:5000
```

Step 6: Test with Sample Values

Parameter	Sample Value
ambient	0.75
coolant	-1.12
u_d	0.33
u_q	-1.30
motor_speed	-1.22
i_d	1.03
i_q	-0.25
Expected Result (pm)	-2.52

E. IBM Cloud Deployment (Future Work)

The project is structured and ready for IBM Cloud deployment. All required files are pre-configured in the "IBM scoring end point" folder.

manifest.yml

```
applications:
- name: motor-temp-predictor
memory: 512M
instances: 1
random-route: true
buildpack: python_buildpack
command: gunicorn app:app --bind 0.0.0.0:$PORT
```

Deployment Commands

```
# Step 1: Login to IBM Cloud
ibmcloud login

# Step 2: Target Cloud Foundry
ibmcloud target --cf

# Step 3: Push the application
cd "IBM scoring end point"
ibmcloud cf push
```

Once deployed, the application will be accessible at: https://motor-temp-predictor-{random-words}.us-south.cf.appdomain.cloud

F. GitHub & Project Demo Link

Repository URL: https://github.com/yourusername/electric-motor-temperature-prediction

```
electric-motor-temperature-prediction/
|
|-- README.md # Project overview and setup instructions
|-- requirements.txt # Python dependencies
|-- Dataset.zip # Original dataset (compressed)
|-- pmsm_temperature_data.csv # Extracted dataset
|-- Rotor Temperature Detection.ipynb # Complete Jupyter notebook
|-- model.save # Trained Random Forest model
|-- transform.save # MinMaxScaler
|
|-- Flask/
| |-- app.py # Flask web application
| |-- model.save # Model copy for Flask
| |-- transform.save # Scaler copy for Flask
| `-- templates/
| |-- home.html
| |-- Manual_predict.html
| `-- Sensor_predict.html
|
`-- screenshots/
|-- home.png |-- manual.png
|-- manual_result.png|-- sensor.png
|-- sensor_result.png`-- model_comparison.png
```

G. Project Demo Video

A complete walkthrough video demonstrating all aspects of the project:

Section	Duration	Content Covered
Project Overview	1 min	Introduction to the problem statement and solution approach
Data Analysis	2 mins	Loading dataset, visualization techniques, key insights from EDA
Section	**Duration**	**Content Covered**
Model Building	3 mins	Training all 4 algorithms, performance comparison, selecting Random Forest
Web Application Demo	3 mins	Home page navigation, manual input, sensor simulation, real-time predictions

Local Setup Guide	1 min	Installing dependencies, running the app, and testing predictions

Video URL: https://github.com/harisreeakula/Electric-Motor-Temperature-Prediction-using-Machine-Learning/tree/main/Demo%20video

H. Glossary of Terms

Term	Definition
PMSM	Permanent Magnet Synchronous Motor — the type of motor used in this dataset
R^2 Score	Coefficient of determination — measures how well the model explains variance in the data (1.0 = perfect fit)
RMSE	Root Mean Square Error — measures the average magnitude of prediction errors in original units
MinMaxScaler	Feature scaling technique that transforms all values to a range between 0 and 1
Flask	Lightweight Python micro web framework used to build the prediction web application
Joblib	Python library for efficient serialization and deserialization of ML models and scalers
u_d, u_q	Direct-axis and quadrature-axis voltage components in the d-q reference frame of a PMSM
i_d, i_q	Direct-axis and quadrature-axis current components in the d-q reference frame of a PMSM
pm	Permanent Magnet temperature — the target variable this model is trained to predict
Random Forest	Ensemble ML method that builds many decision trees and averages their predictions for higher accuracy
SCADA	Supervisory Control and Data Acquisition — industrial monitoring and control systems
MQTT	Message Queuing Telemetry Transport — lightweight messaging protocol used in IoT systems
LSTM	Long Short-Term Memory — a type of recurrent neural network suited for sequential/time-series data
d-q Frame	Direct-quadrature reference frame — mathematical transformation used in motor control analysis

PROJECT COMPLETION SUMMARY

Section	Status	Remarks
1. INTRODUCTION	Complete	Project overview and purpose clearly defined
2. IDEATION PHASE	Complete	Problem statement, empathy map, and brainstorming documented
3. REQUIREMENT ANALYSIS	Complete	Customer journey, requirements, DFD, and tech stack covered
4. PROJECT DESIGN	Complete	Solution architecture and problem-solution fit documented
5. PROJECT PLANNING	Complete	Timeline, 8 phases, and task breakdown completed
6. TESTING	Complete	All 10 functional and performance test cases passed
7. RESULTS	Complete	Model performance, screenshots, feature importance documented
8. ADVANTAGES & DISADVANTAGES	Complete	10 advantages and 10 disadvantages comprehensively listed
9. CONCLUSION	Complete	Key achievements, learning outcomes, and final summary
10. FUTURE SCOPE	Complete	13 enhancements across short, medium, and long-term horizons
11. APPENDIX	Complete	Full source code, HTML templates, dataset info, IBM deployment, GitHub structure, glossary