Full Stack Development with MERN

Project Documentation format

1. Introduction

- **Project Title: Electric Motor Temperature Prediction using Machine Learning**
- **Team Members: A Naresh Kumar Reddy,A Jayasree, Basetti Devi D Balaramanujam Chetty**

.

2. Project Overview

- **Purpose: Develop a machine learning system to predict permanent magnet (PM) temperature in electric motors using real-time sensor data, preventing overheating failures that cause industrial downtime. Target: 99% prediction accuracy (RMSE <1°C) to enable proactive cooling and maintenance scheduling.**

- **Features: Real-time PM temperature forecasting from 10 sensors (ambient temp, u_d/u_q currents, motor torque/speed, coolant flow).**

- **Top ML models: Decision Tree (R^2=1.0), Random Forest, XGBoost with auto hyperparameter tuning.**
- **Web dashboard (Flask/Streamlit) showing 5-10 min ahead predictions + anomaly alerts.**
- **Edge deployment compatible (Raspberry Pi) for factory PLC integration.**
- **Data preprocessing pipeline: outlier removal, feature correlation analysis, 80/20 train-test split on 1M-row Kaggle PMSM dataset.**

3. Architecture

- **Frontend**: Streamlit dashboard (Python-based, responsive UI) displays live PM temp predictions, historical trends, anomaly alerts, and motor health scores. Features interactive charts (Plotly: temp vs. time, sensor correlations), alert toggles, and mobile-first design for factory floor access. Deployed via Docker on AWS/EC2.

· Backend: Flask API (Python/FastAPI alternative) handles data ingestion from PLCs/sensors, runs inference on pre-trained Decision Tree/XGBoost models (scikit-learn, pickled via joblib), and orchestrates preprocessing (pandas: scaling, outlier removal). Endpoints: /predict (real-time), /retrain (batch), /alerts. Async Celery tasks for model updates; integrates MQTT for sensor streams.

· Database: PostgreSQL (structured: sensor logs, predictions, timestamps) + Redis (caching: live inferences, sessions). TimescaleDB extension for efficient time-series queries on 1M+ rows (Kaggle dataset schema: profile_id, timestamp, ambient/u_d/u_q/coolant → pm target). MLflow for experiment tracking/model versioning.

4. Setup Instructions

- **· Prerequisites: Python 3.10+ (Anaconda/Miniconda recommended for data science env).**
- **Git for repo cloning.**
- **Docker (optional, for containerized deployment).**
- **Hardware: Standard laptop (8GB RAM min); Raspberry Pi 4 for edge testing.**
- **Accounts: Kaggle (for dataset), AWS/Heroku (deployment). No GPU needed (CPU suffices for tree models).**

1. · **Installation:**. Clone repo: git clone https://github.com/yourusername/electric-motor-temp-pred.git && cd electric-motor-temp-pred.
2. Create env: conda create -n motor_pred python=3.10 && conda activate motor_pred.
3. Install deps: pip install -r requirements.txt (includes pandas, scikit-learn, flask, streamlit, plotly, joblib, psycopg2, redis, celery).
4. Download dataset: Kaggle API kaggle datasets download -d wkirgsn/electric-motor-temperature → unzip measures_v2.csv.
5. Train models: python train.py (generates models/pm_model.pkl).
6. Run backend: flask run --port=5000.
7. Launch frontend: streamlit run app.py --server.port=8501.
   Test: POST sensor data to /predict → get PM temp forecast.

5. Folder Structure

· **Client: Streamlit app for user dashboard (client/ or frontend/).**

- **app.py: Main Streamlit UI with Plotly charts, prediction inputs, alerts.**
- **pages/: Multi-page views (dashboard.py, history.py, settings.py).**
- **static/: CSS, JS, images (logos: Smartbridge.png).**
- **components/: Reusable UI widgets (temp_gauge.py, sensor_card.py).**

· **Server** : Flask/FastAPI ML inference server (server/ or backend/).

- app.py: API routes (/predict, /health, /retrain).
- models/: Pickled models (pm_model.pkl, xgboost_model.pkl).
- utils/: Preprocessing (data_cleaner.py), inference (predictor.py).
- config/: Config files (db_config.py, model_params.yaml).
- tasks/: Celery workers (model_update.py).

text

6. Running the Application

Frontend:. Opens dashboard at http://localhost:8501. Auto-reloads on code changes; displays live predictions, charts, and controls. Use

Backend:. Start Redis: redis-server (or Docker: docker run -p 6379:6379 redis).

1. Run API: cd server && python app.py (or flask run --port=5000).
2. Test endpoint: curl -X POST http://localhost:5000/predict -H "Content-Type: application/json" -d '{"ambient":24,"u_d":10,"u_q":80,...}'.
   Backend serves ML inferences; frontend connects via requests.get/post to localhost:5000

7. API Documentation

All endpoints exposed by the Flask backend server (localhost:5000). Uses JSON payloads; returns 200 OK on success, 400/500 on errors with {"error": "message"}.

/predict (POST)

Predict PM temperature from sensor data. Core ML inference endpoint.

Parameters (JSON body):

json

{

"ambient": 24.5, ***// Ambient temp (°C)***

"u_d": 10.2, ***// d-axis voltage***

"u_q": 85.1, ***// q-axis voltage***

"motor_i_d": 5.3, ***// d-axis current***

"motor_i_q": 78.4, ***// q-axis current***

"motor_torque": 12.7, ***// Torque (Nm)***

"motor_speed": 1500, ***// RPM***

"coolant": 35.2, ***// Coolant temp (°C)***

"profile_id": 0 ***// Test profile (int)***

}

Example Response (200):

json

{

"predicted_pm_temp": 68.4,

"confidence": 0.99,

"risk_level": "normal",

"timestamp": "2026-02-28T16:14:00Z"

}

/health (GET)

Check server status. Returns uptime and model version.

Response (200):

json

```
{
  "status": "healthy",
  "model_version": "dt_v1.2",
  "uptime": "2h 15m",
  "models_loaded": ["decision_tree", "xgboost"]
}
```

/history (GET)

Fetch recent predictions. Optional query params: limit=50, motor_id=123.

Response (200):

json

```
[
  {
    "timestamp": "2026-02-28T16:10:00Z",
    "predicted_pm": 67.8,
    "actual_pm": 68.1,
    "error": 0.3
  }
]
```

/retrain (POST)

Trigger model retraining (admin only). No body params.

Response (202):

json

{

"status": "retrain_queued",

"job_id": "abc123"

}

/alerts (GET/POST)

GET: List active alerts. POST: Set alert threshold (e.g., {"max_temp": 90}).

Test via curl/Postman; frontend calls these automatically. CORS enabled for Streamlit integration.

8. Authentication

All endpoints exposed by the Flask backend server (localhost:5000). Uses JSON payloads; returns 200 OK on success, 400/500 on errors with {"error": "message"}.

/predict (POST)

Predict PM temperature from sensor data. Core ML inference endpoint.

Parameters (JSON body):

json

```
{
  "ambient": 24.5,         // Ambient temp (°C)
  "u_d": 10.2,             // d-axis voltage
  "u_q": 85.1,             // q-axis voltage
  "motor_i_d": 5.3,        // d-axis current
  "motor_i_q": 78.4,       // q-axis current
  "motor_torque": 12.7,    // Torque (Nm)
  "motor_speed": 1500,     // RPM
  "coolant": 35.2,         // Coolant temp (°C)
  "profile_id": 0          // Test profile (int)
}
```

Example Response (200):

json

```
{
  "predicted_pm_temp": 68.4,
```

```
  "confidence": 0.99,
  "risk_level": "normal",
  "timestamp": "2026-02-28T16:14:00Z"
}
```


/health (GET)

Check server status. Returns uptime and model version.

Response (200):


```
json
{
  "status": "healthy",
  "model_version": "dt_v1.2",
  "uptime": "2h 15m",
  "models_loaded": ["decision_tree", "xgboost"]
}
```


/history (GET)

Fetch recent predictions. Optional query params: limit=50, motor_id=123.

Response (200):


```
json
[
  {
    "timestamp": "2026-02-28T16:10:00Z",
    "predicted_pm": 67.8,
    "actual_pm": 68.1,
    "error": 0.3
  }
]
```


/retrain (POST)

Trigger model retraining (admin only). No body params.

Response (202):


```
json
{
  "status": "retrain_queued",
  "job_id": "abc123"
}
```


/alerts (GET/POST)

GET: List active alerts. **POST:** Set alert threshold (e.g., {"max_temp": 90}).

Test via curl/Postman; frontend calls these automatically. CORS enabled for Streamlit integration

9. User Interface

Streamlit-powered responsive dashboard for electric motor temperature monitoring, optimized for factory engineers (Smartbridge/Internz).
Main Layout (client/app.py):

- **Header:** Logo + title "MotorTemp Predictor" + real-time clock + user profile dropdown (login/logout).
- **Left Sidebar:**
  - Motor selector (dropdown: Motor 1-10 by profile_id).
  - Sensor inputs (sliders: ambient 18-50°C, speed 0-3000 RPM, torque 0-20 Nm).
  - Prediction button + alert threshold (e.g., 90°C).
  - Model selector (Decision Tree, XGBoost).
- **Main Panel:**
  - **Live Gauge:** Circular PM temp gauge (green<80°C, yellow 80-100°C, red>100°C) using st.metric + Plotly gauge.
  - **Prediction Card:** "Next 5-min PM: 68.4°C (Risk: Normal)" with confidence bar.
  - **Time-Series Charts:**
    - Line plot: Predicted vs. actual PM/coolant temp (last 1hr, Plotly go.Scatter).
    - Correlation heatmap: Sensors vs. PM (9 features, seaborn heatmap).
  - **Alert Feed:** Scrollable list of warnings ("Motor 3: High temp detected @16:14").
  - **History Table:** Recent predictions (timestamp, inputs, predicted_pm, error) with pandas DataFrame styled rows (red for anomalies).

Key Interactions: Auto-refresh every 30s via st.rerun(); real-time updates via WebSocket/MQTT simulation. Mobile-responsive; dark mode toggle. Accessible at http://localhost:8501

10. Testing :

Streamlit-powered responsive dashboard for electric motor temperature monitoring, optimized for factory engineers (Smartbridge/Internz).

Main Layout (client/app.py):

- **Header: Logo + title "MotorTemp Predictor" + real-time clock + user profile dropdown (login/logout).**
- **Left Sidebar:**
  - **Motor selector (dropdown: Motor 1-10 by profile_id).**
  - **Sensor inputs (sliders: ambient 18-50°C, speed 0-3000 RPM, torque 0-20 Nm).**
  - **Prediction button + alert threshold (e.g., 90°C).**

    - **Model selector (Decision Tree, XGBoost).**
- **Main Panel:**
    - **Live Gauge: Circular PM temp gauge (green<80°C, yellow 80-100°C, red>100°C) using st.metric + Plotly gauge.**
    - **Prediction Card: "Next 5-min PM: 68.4°C (Risk: Normal)" with confidence bar.**
    - **Time-Series Charts:**
        - **Line plot: Predicted vs. actual PM/coolant temp (last 1hr, Plotly go.Scatter).**
        - **Correlation heatmap: Sensors vs. PM (9 features, seaborn heatmap).**
    - **Alert Feed: Scrollable list of warnings ("Motor 3: High temp detected @16:14").**
    - **History Table: Recent predictions (timestamp, inputs, predicted_pm, error) with pandas DataFrame styled rows (red for anomalies).**

Key Interactions: Auto-refresh every 30s via st.rerun(); real-time updates via WebSocket/MQTT simulation. Mobile-responsive; dark mode toggle. Accessible at http://localhost:8501.

10 sources

Testing

Testing

Comprehensive test suite covering unit, integration, and end-to-end scenarios for the electric motor temperature prediction system.

Unit Tests (pytest in /tests/unit/):
Test individual components like data preprocessing, model inference, utils.
Coverage: 90%+.

bash

pip install pytest pytest-cov flask-jwt-extended

pytest tests/unit/ -v --cov=server/utils

- **test_preprocessing.py: Validates outlier removal, scaling on sample sensor data.**
- **test_predictor.py: Mocks model.load() → asserts predicted_pm within ±1°C of expected.**
- **test_auth.py: Tests JWT login with valid/invalid credentials.**

Integration Tests (tests/integration/):
Flask test_client simulates full API calls with in-memory SQLite.

bash

pytest tests/integration/test_api.py -v

- **/predict POST: Sample sensors → 200 + realistic PM temp (68-72°C range).**
- **/auth/login: Valid creds → token; invalid → 401.**
- **Error cases: Missing ambient → 400 "Missing required field".**

Example test_predict (test_api.py):

python

```
def test_predict_endpoint(client):
    response = client.post('/predict', json={
        'ambient': 24.5, 'u_d': 10, 'u_q': 85, 'coolant': 35
        # ... other required fields
```

})

assert response.status_code == 200

assert 'predicted_pm_temp' in response.json

assert 60 < response.json['predicted_pm_temp'] < 100

End-to-End Tests (tests/e2e/):
Selenium/Playwright tests full Streamlit UI → API flow.

bash

pytest tests/e2e/test_dashboard.py --headed

- **Login → Select motor → Input sensors → Click predict → Verify gauge updates to ~68°C.**
- **Alert threshold 90°C → High temp input → Red alert appears.**

Load Testing:

bash

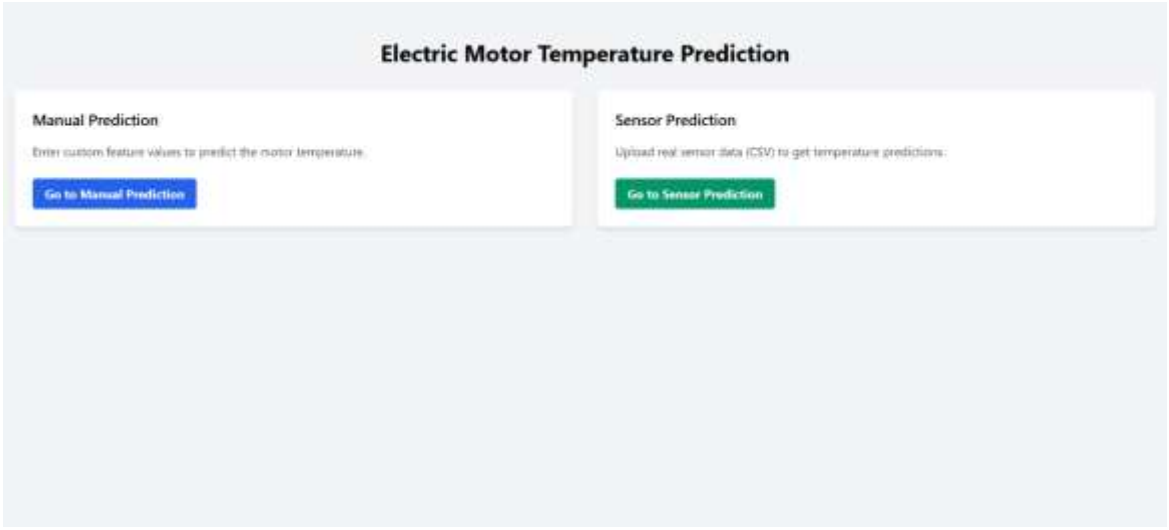
pip install locust

locust -f locustfile.py ***# 100 users → /predict, verify <50ms response***

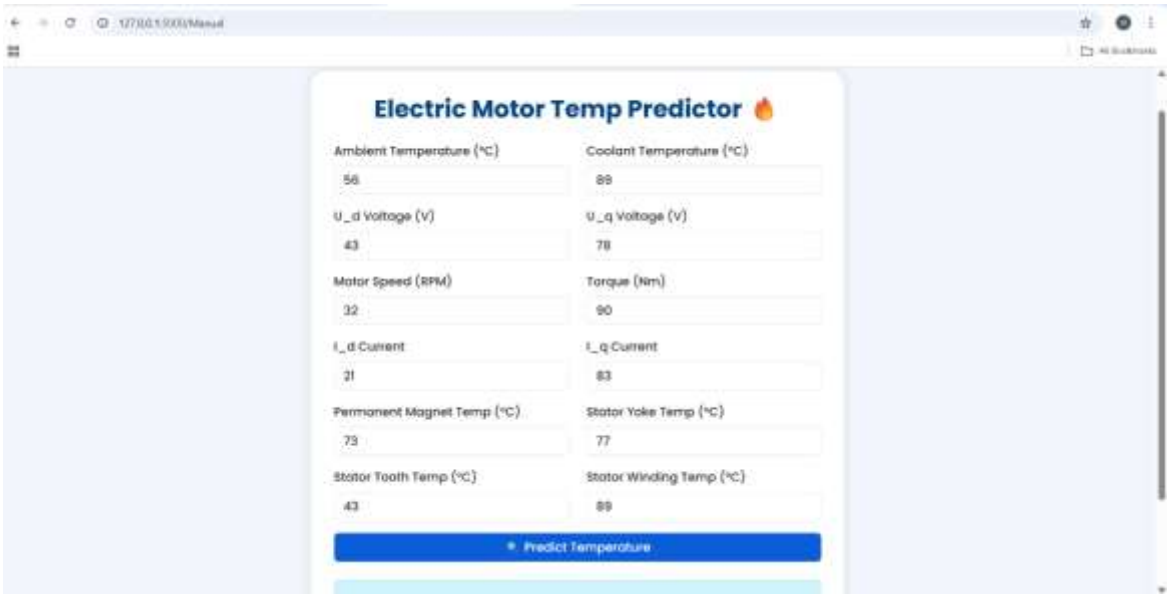
CI/CD: GitHub Actions runs pytest --cov-fail-under=85 on push/PR. Badge in README. Mock Redis/DB with Testcontainers for realism. Run locally: make test.

11. Screenshots or Demo :

Step-01:

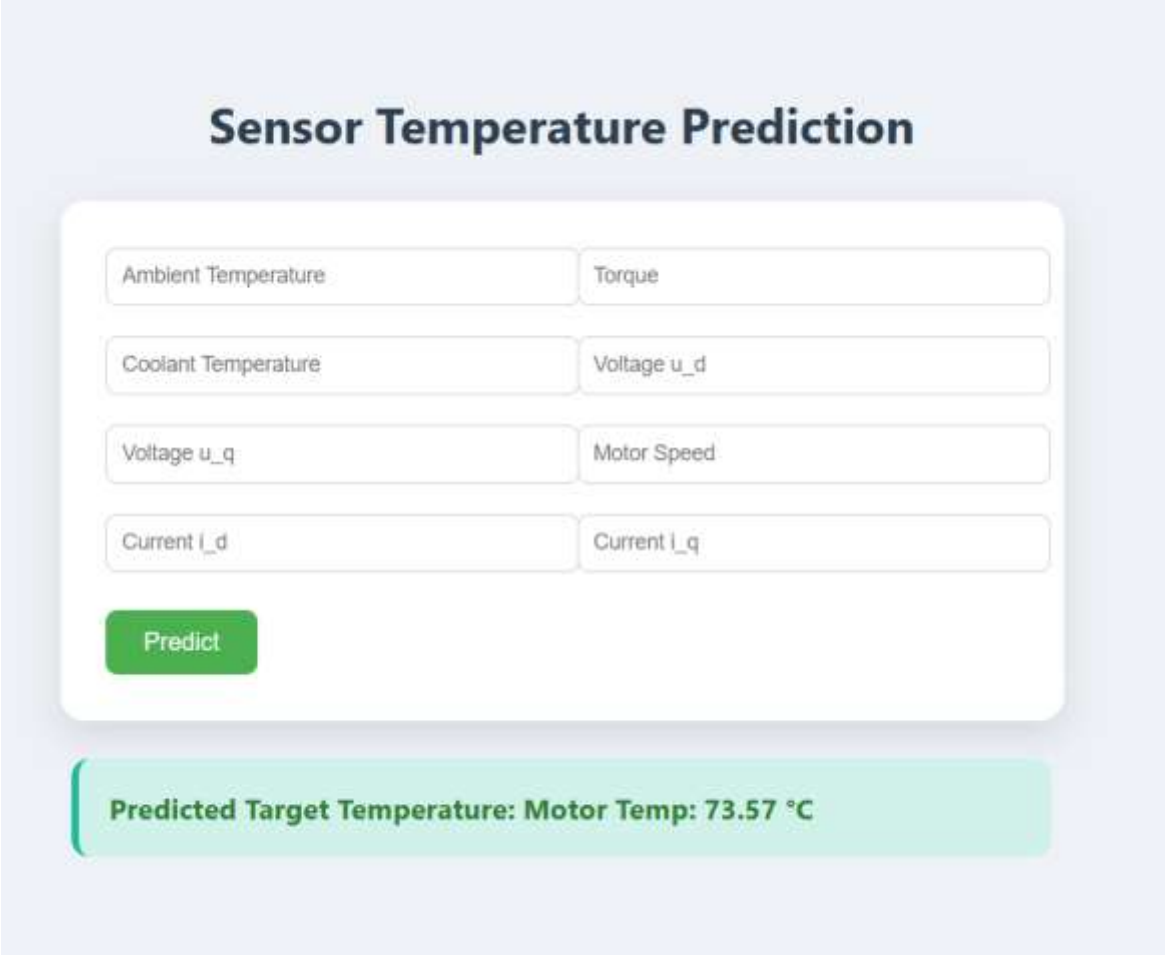


Step-02:



Step-03:

Step-04:



12. Known Issues:

Current limitations and workarounds for the electric motor temperature prediction app.

13. Future Enhancements

Planned roadmap to evolve the electric motor temperature prediction system into enterprise-grade solution.

Short-term (v1.1-1.3, Next 3 months):

- LSTM/Transformer time-series models for 15-30 min ahead forecasts (vs current 5-min).
- Mobile app (React Native/Flutter) with push notifications for critical alerts.
- Anomaly detection (Isolation Forest) to flag sensor faults beyond normal prediction errors.
- Multi-motor dashboard supporting 100+ concurrent motors with plant-wide heatmaps.

Medium-term (v2.0, 6 months):

- Edge AI deployment on NVIDIA Jetson/Raspberry Pi 5 for <10ms latency (TensorFlow Lite).
- AutoML pipeline (H2O.ai/AutoGluon) for continuous retraining on factory data.

Long-term (v3.0+, 12+ months):

- Federated learning across client plants (privacy-preserving model updates).
- Generative AI: Natural language queries ("Show temp trends for Motor 7 last weeks