EV Battery Health Prediction: 3-Day Capstone Project Guide

Intelligent Predictive Maintenance and Battery Health Forecasting System for Electric Vehicles

Executive Summary

This comprehensive guide provides a **step-by-step roadmap** to complete an EV battery health prediction capstone project within **3 days**. The project focuses on predicting **State of Health (SoH)** and **Remaining Useful Life (RUL)** of lithium-ion batteries using machine learning techniques across supervised, unsupervised, and reinforcement learning paradigms[1][2].

The guide covers data acquisition from NASA and CALCE datasets, exploratory data analysis, feature engineering, model development, evaluation, and deployment to free cloud platforms like Hugging Face Spaces[38][39].

Project Objectives

Primary Goals

- Predict battery **SoH within ±5% absolute error** on unseen data[^1]
- Estimate **RUL with >90% recall** for failure detection[^1]
- Build a **scalable, deployable ML pipeline** within 3 days
- Deploy as a **web application** on free hosting platforms[38][41]

Technical Requirements

- Implement **supervised learning** (Random Forest, XGBoost, LSTM/GRU)[2][7]
- Apply **unsupervised learning** (K-means clustering for degradation patterns)[59][62]
- Integrate **reinforcement learning** (Q-learning/DQN for charging optimization)[60][63]
- Achieve **engineering-grade data quality** (<5% missing data)[^1]

Day 1: Data Acquisition, EDA, and Baseline Models

Morning Session (3-4 hours): Data Setup and EDA

Step 1.1: Dataset Acquisition

Data Sources:

1. **NASA PCoE Battery Dataset**[2][23]
    - Location: https://www.nasa.gov/intelligent-systems-division/discovery-and-systems-health/pcoe/pcoe-data-set-repository/
    - Contains: Li-ion 18650 battery aging data
    - Batteries: B0005, B0006, B0007, B0018 (commonly used for testing)[^19]
    - Format: .mat files with voltage, current, temperature, capacity
2. **CALCE Battery Dataset**[16][20][^24]
    - Location: https://calce.umd.edu/battery-data
    - Contains: LiCoO2-graphite pouch cells under various stress conditions
    - Variables: Temperature (10°C-60°C), C-rates, charge cutoff conditions
    - Sample size: 192 cells with different aging profiles

Download Instructions:

```
import scipy.io
import pandas as pd

# Load NASA .mat file
mat_data = scipy.io.loadmat('B0005.mat')
battery_data = mat_data['B0005'][^0][^0]

# Extract cycle data
cycle = battery_data['cycle'][^0]
```

Step 1.2: Exploratory Data Analysis (EDA)

Statistical Summary[76][79]

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Create DataFrame from battery cycles
capacity_data = []
for i in range(len(cycle)):
    capacity_data.append({
        'cycle': i,
        'capacity': cycle[i]['data'][^0][^0]['Capacity'][^0][^0][^0],
        'voltage_avg': cycle[i]['data'][^0][^0]['Voltage_measured'][^0][^0].mean(),
        'current_avg': cycle[i]['data'][^0][^0]['Current_measured'][^0][^0].mean(),
        'temp_avg': cycle[i]['data'][^0][^0]['Temperature_measured'][^0][^0].mean()
    })

df = pd.DataFrame(capacity_data)
print(df.describe())
```

Key Visualizations:

1. **Capacity Degradation Curves**[2][7]
    - Plot capacity vs. cycle number for all batteries
    - Identify knee points where rapid degradation begins[^25]
2. **Correlation Analysis**[52][57]
    - Heatmap of features (voltage, current, temperature, capacity)
    - Identify highly correlated features for dimensionality reduction
3. **Outlier Detection**[^79]
    - Z-score method: |z| > 3 indicates outliers
    - Box plots for voltage/current/temperature distributions

Expected Outputs:

- Data shape and statistics summary
- Missing value report (<5% target)[^1]
- Capacity fade visualization with knee points identified[^25]
- Feature correlation matrix

Afternoon Session (4-5 hours): Feature Engineering

Step 1.3: Feature Engineering for SoH/RUL[71][76]

Electrochemical Features:

1. **Charge Throughput (Ah)**

   ```
   df['charge_throughput'] = df.groupby('battery')['current'].cumsum() * (1/3600)
   ```

   Importance: Direct indicator of battery aging[^1]

2. **C-Rate**

   ```
   rated_capacity = 2.0  # Ah
   df['c_rate'] = df['current'] / rated_capacity
   ```

   Importance: High C-rates cause lithium plating and faster fade[^24]

3. **Depth of Discharge (DoD)**

   ```
   df['dod'] = (df['discharge_capacity'] / rated_capacity) * 100
   ```

   Importance: Higher DoD leads to more stress[16][20]

4. **Cumulative Energy Throughput (Wh)**

   ```
   df['energy_throughput'] = (df['voltage'] * df['current'] * df['time']).cumsum()
   ```

   Importance: Total energy processed correlates with SoH decline[^71]

5. **Internal Resistance (Ohms)**

   ```
   df['internal_resistance'] = df['voltage_drop'] / df['current']
   ```

   Importance: Rising resistance indicates capacity loss[^23]

Health Indicators:

1. **Incremental Capacity (IC):** $\frac{dQ}{dV}$ [52][57]
2. **Differential Voltage (DV):** $\frac{dV}{dQ}$ [^9]
3. **Capacity Retention Ratio:** $\frac{\text{Current Capacity}}{\text{Initial Capacity}} \times 100$[^75]

Temporal Features:[^80]

- Cycle number (primary feature)
- Capacity fade rate between consecutive cycles
- Time between charges (rest periods)
- Charge/discharge duration

Data Preprocessing:

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Normalize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train-test split (80-20)
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42
)
```

Evening Session (2-3 hours): Supervised Learning Baseline

Step 1.4: Baseline Supervised Models

Model 1: Random Forest Regression[51][56]

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

rf_model = RandomForestRegressor(
    n_estimators=100,
    max_depth=10,
    random_state=42
)

rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)

# Evaluate
rmse_rf = mean_squared_error(y_test, y_pred_rf, squared=False)
mae_rf = mean_absolute_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

print(f"Random Forest - RMSE: {rmse_rf:.4f}, MAE: {mae_rf:.4f}, R²: {r2_rf:.4f}")
```

Expected Performance:[^56]

- RMSE: 1.5-2.5% of capacity
- R^2: >0.90
- Training time: 5-10 minutes

Model 2: XGBoost Regression[2][7]

```
import xgboost as xgb

xgb_model = xgb.XGBRegressor(
    n_estimators=100,
    learning_rate=0.1,
    max_depth=5,
    random_state=42
)

xgb_model.fit(X_train, y_train)
y_pred_xgb = xgb_model.predict(X_test)

# Evaluate
rmse_xgb = mean_squared_error(y_test, y_pred_xgb, squared=False)
print(f"XGBoost - RMSE: {rmse_xgb:.4f}")
```

Feature Importance Analysis:

```
import matplotlib.pyplot as plt

importances = rf_model.feature_importances_
feature_names = X.columns

plt.figure(figsize=(10, 6))
plt.barh(feature_names, importances)
plt.xlabel('Feature Importance')
plt.title('Random Forest Feature Importance for SoH Prediction')
plt.tight_layout()
plt.savefig('feature_importance.png')
```

Day 1 Deliverables:

- ✓ Downloaded NASA/CALCE datasets

- ✓ EDA notebook with 5+ visualizations
- ✓ Cleaned dataset with engineered features (CSV)
- ✓ Baseline Random Forest and XGBoost models
- ✓ Initial metrics: RMSE, MAE, R^2 scores

Day 2: Deep Learning, Unsupervised, and Reinforcement Learning

Morning Session (3-4 hours): Deep Learning Models

Step 2.1: LSTM for Time-Series SoH Prediction

Architecture:[2][4][^7]

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

# Reshape data for LSTM (samples, timesteps, features)
X_train_lstm = X_train.reshape((X_train.shape[^0], 1, X_train.shape[^1]))
X_test_lstm = X_test.reshape((X_test.shape[^0], 1, X_test.shape[^1]))

# Build LSTM model
lstm_model = Sequential([
    LSTM(64, activation='relu', return_sequences=True, input_shape=(1, X_train.shape[^1])),
    Dropout(0.2),
    LSTM(32, activation='relu'),
    Dropout(0.2),
    Dense(16, activation='relu'),
    Dense(1)  # SoH prediction
])

lstm_model.compile(optimizer='adam', loss='mse', metrics=['mae'])

# Train
history = lstm_model.fit(
    X_train_lstm, y_train,
    validation_split=0.2,
    epochs=50,
    batch_size=32,
    verbose=1
)

# Predict
y_pred_lstm = lstm_model.predict(X_test_lstm)
rmse_lstm = mean_squared_error(y_test, y_pred_lstm, squared=False)
print(f"LSTM - RMSE: {rmse_lstm:.4f}")
```

Expected Performance:[2][4]

- RMSE: 1.0-1.5% (better than traditional ML)[^77]
- Training time: 15-20 minutes on CPU
- Accuracy: 96-97%[^17]

Model 2: GRU (Gated Recurrent Unit)[77][80]

```
from tensorflow.keras.layers import GRU

gru_model = Sequential([
    GRU(64, activation='relu', return_sequences=True, input_shape=(1, X_train.shape[^1])),
    Dropout(0.2),
    GRU(32, activation='relu'),
    Dense(16, activation='relu'),
    Dense(1)
])
```

```
gru_model.compile(optimizer='adam', loss='mse', metrics=['mae'])
gru_model.fit(X_train_lstm, y_train, epochs=50, batch_size=32, validation_split=0.2)
```

GRU Advantages:[^80]

- Faster training than LSTM (fewer parameters)
- Average RMSE: 0.724%[^77]
- Comparable accuracy to LSTM

Step 2.2: Unsupervised Learning - K-Means Clustering

Objective: Cluster batteries by degradation patterns[59][62]

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

# Determine optimal K using elbow method
inertias = []
silhouette_scores = []
K_range = range(2, 10)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertias.append(kmeans.inertia_)
    silhouette_scores.append(silhouette_score(X_scaled, kmeans.labels_))

# Plot elbow curve
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(K_range, inertias, 'bx-')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal K')

plt.subplot(1, 2, 2)
plt.plot(K_range, silhouette_scores, 'rx-')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Analysis')
plt.tight_layout()
plt.savefig('clustering_analysis.png')

# Apply K-means with optimal K
optimal_k = 4  # Based on elbow/silhouette analysis
kmeans_final = KMeans(n_clusters=optimal_k, random_state=42)
clusters = kmeans_final.fit_predict(X_scaled)

# Analyze cluster characteristics
df['cluster'] = clusters
cluster_summary = df.groupby('cluster').agg({
    'capacity': 'mean',
    'charge_throughput': 'mean',
    'c_rate': 'mean',
    'cycle': 'count'
}).round(2)

print("Cluster Characteristics:")
print(cluster_summary)
```

Applications:[^59]

- Identify batteries with similar degradation patterns
- Proactive maintenance strategies by cluster
- Improved RUL predictions for clustered batteries

Afternoon Session (4-5 hours): Reinforcement Learning

Step 2.3: Q-Learning for EV Charging Optimization

Environment Setup:[60][63][^66]

State Space:

- Current SoC (State of Charge): 0-100%
- Battery temperature: °C
- Grid load: Low/Medium/High
- Electricity price: $/kWh

Action Space:

- Charge rate: 0-1C (continuous or discretized)
- Charge duration: minutes
- V2G discharge option: Yes/No

Reward Function:[^63]

$$R = -\alpha \cdot \mathrm{Cost} + \beta \cdot \mathrm{SoH}\backslash\_\mathrm{improvement} - \gamma \cdot \mathrm{Grid}\backslash\_\mathrm{stress}$$

Where:

- α, β, γ are weight parameters
- Cost: Electricity cost
- SoH_improvement: Reduced degradation
- Grid_stress: Peak load contribution

Q-Learning Implementation:

```
import numpy as np
import gym

# Simplified Q-learning for discrete state-action space
class BatteryChargingEnv:
    def __init__(self):
        self.state_space_size = 100  # SoC levels
        self.action_space_size = 5   # Charge rates: 0, 0.25C, 0.5C, 0.75C, 1C
        self.current_soc = 50
        self.battery_health = 100

    def reset(self):
        self.current_soc = np.random.randint(0, 50)
        return self.current_soc

    def step(self, action):
        # Simulate charging
        charge_rates = [0, 0.25, 0.5, 0.75, 1.0]
        rate = charge_rates[action]

        # Update SoC
        self.current_soc = min(100, self.current_soc + rate * 10)

        # Calculate reward (simplified)
        cost = rate * 0.1  # Electricity cost
        degradation = rate * 0.05  # Higher rates = more degradation
        reward = (self.current_soc / 100) * 10 - cost - degradation

        done = self.current_soc &gt;= 95
        return self.current_soc, reward, done

# Q-learning algorithm
env = BatteryChargingEnv()
Q = np.zeros((env.state_space_size, env.action_space_size))
```

```
# Hyperparameters
alpha = 0.1  # Learning rate
gamma = 0.95  # Discount factor
epsilon = 0.1  # Exploration rate
episodes = 1000

for episode in range(episodes):
    state = env.reset()
    done = False

    while not done:
        # Epsilon-greedy action selection
        if np.random.random() &lt; epsilon:
            action = np.random.randint(0, env.action_space_size)
        else:
            action = np.argmax(Q[state, :])

        next_state, reward, done = env.step(action)

        # Q-table update
        Q[state, action] = Q[state, action] + alpha * (
            reward + gamma * np.max(Q[next_state, :]) - Q[state, action]
        )

        state = next_state

print("Q-learning training complete!")
print(f"Optimal policy learned for {episodes} episodes")
```

Deep Q-Network (DQN) Alternative:[63][66]
For continuous state spaces, implement DQN using TensorFlow:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# DQN network
dqn_model = Sequential([
    Dense(24, activation='relu', input_shape=(4,)),  # 4 state features
    Dense(24, activation='relu'),
    Dense(5)  # 5 actions (charge rates)
])

dqn_model.compile(optimizer='adam', loss='mse')
```

Expected Outcomes:[^63]

- Learned optimal charging policy
- Reduced battery degradation by 10-15%
- Minimized charging costs

Evening Session (2-3 hours): Model Evaluation and Explainability

Step 2.4: Comprehensive Model Comparison

Metrics:[1][6][^7]

Model	RMSE (%)	MAE (%)	R^2	Training Time	Inference Time
Random Forest	1.8	1.4	0.92	8 min	10 ms
XGBoost	1.5	1.2	0.94	6 min	8 ms
LSTM	1.2	0.9	0.96	18 min	15 ms

Model	RMSE (%)	MAE (%)	R^2	Training Time	Inference Time
GRU	1.1	0.8	0.97	14 min	12 ms

SHAP Explainability:[1][7]

```
import shap

# Use SHAP for feature importance and model interpretation
explainer = shap.TreeExplainer(rf_model)
shap_values = explainer.shap_values(X_test)

# Visualize
shap.summary_plot(shap_values, X_test, feature_names=X.columns)
plt.savefig('shap_summary.png')
```

Day 2 Deliverables:

- ✓ Trained LSTM/GRU models with <1.5% RMSE
- ✓ K-means clustering analysis with 4 degradation clusters
- ✓ Q-learning/DQN agent for charging optimization
- ✓ Model comparison table with all metrics
- ✓ SHAP explainability plots

Day 3: Deployment and Documentation

Morning Session (3-4 hours): Application Development

Step 3.1: Build Streamlit Web Application

app.py:

```
import streamlit as st
import pandas as pd
import numpy as np
import joblib
import matplotlib.pyplot as plt

# Load trained model
@st.cache_resource
def load_model():
    model = joblib.load('best_model.pkl')
    scaler = joblib.load('scaler.pkl')
    return model, scaler

model, scaler = load_model()

# App title
st.title("□ EV Battery Health Prediction System")
st.markdown("Predict State of Health (SoH) and Remaining Useful Life (RUL)")

# Sidebar inputs
st.sidebar.header("Battery Parameters")
cycle_number = st.sidebar.number_input("Cycle Number", min_value=1, max_value=2000, value=100)
voltage = st.sidebar.slider("Average Voltage (V)", 2.5, 4.5, 3.7)
current = st.sidebar.slider("Average Current (A)", 0.0, 5.0, 2.0)
temperature = st.sidebar.slider("Temperature (°C)", 15.0, 45.0, 25.0)
capacity = st.sidebar.number_input("Discharge Capacity (Ah)", 1.0, 3.0, 2.0)

# Feature engineering
charge_throughput = cycle_number * current * 0.5
c_rate = current / 2.0
dod = (capacity / 2.0) * 100
```

```
# Prepare input
input_data = np.array([[
    cycle_number, voltage, current, temperature, capacity,
    charge_throughput, c_rate, dod
]])

input_scaled = scaler.transform(input_data)

# Prediction
if st.sidebar.button("Predict SoH &amp; RUL"):
    soh_pred = model.predict(input_scaled)[^0]
    rul_pred = max(0, int((0.8 - soh_pred/100) / 0.0002))  # Simplified RUL calculation

    st.subheader("Prediction Results")
    col1, col2 = st.columns(2)

    with col1:
        st.metric("State of Health (SoH)", f"{soh_pred:.2f}%")

    with col2:
        st.metric("Remaining Useful Life (RUL)", f"{rul_pred} cycles")

    # Health status
    if soh_pred &gt; 85:
        st.success("✅ Battery Health: Excellent")
    elif soh_pred &gt; 70:
        st.warning("⚠ Battery Health: Good")
    else:
        st.error("❌ Battery Health: Replace Soon")

    # Visualization
    fig, ax = plt.subplots(figsize=(8, 4))
    categories = ['Current SoH', 'Healthy Threshold', 'End of Life']
    values = [soh_pred, 80, 0]
    colors = ['green', 'orange', 'red']
    ax.barh(categories, values, color=colors, alpha=0.7)
    ax.set_xlabel('State of Health (%)')
    ax.set_xlim(0, 100)
    st.pyplot(fig)

# Model info
st.sidebar.markdown("---")
st.sidebar.info(f"Model: GRU Neural Network\nAccuracy: 97%\nRMSE: 1.1%")
```

requirements.txt:

```
streamlit==1.30.0
pandas==2.1.0
numpy==1.24.0
scikit-learn==1.3.0
tensorflow==2.14.0
matplotlib==3.7.0
joblib==1.3.0
```

Afternoon Session (3-4 hours): Deployment to Hugging Face Spaces

Step 3.2: Deploy to Hugging Face Spaces[38][39][^41]

Steps:

1. **Create Hugging Face Account**
    - Visit: https://huggingface.co/join
    - Verify email and login
2. **Create New Space**

    - Click "+ New Space"
    - Space name: `ev-battery-health-predictor`
    - SDK: Select **Streamlit**
    - Hardware: CPU basic (FREE)
    - Visibility: Public

3. **Upload Files**[^39]
    - **Method 1: Git Push (Recommended)**

      ```
      git clone https://huggingface.co/spaces/YOUR_USERNAME/ev-battery-health-predictor
      cd ev-battery-health-predictor

      # Add files
      cp app.py requirements.txt best_model.pkl scaler.pkl ./

      # Commit and push
      git add .
      git commit -m "Initial deployment"
      git push
      ```

    - **Method 2: Drag & Drop**[^39]
        - Go to "Files and versions" tab
        - Click "+ Contribute" → "Upload files"
        - Drag and drop: app.py, requirements.txt, model files

4. **Wait for Build**
    - HF Spaces automatically installs dependencies
    - Build takes 2-5 minutes
    - Monitor logs in "Logs" tab

5. **Test Deployment**
    - Access your app at: `https://huggingface.co/spaces/YOUR_USERNAME/ev-battery-health-predictor`
    - Test predictions with sample inputs
    - Share public URL

Alternative: Streamlit Community Cloud[38][41]

1. Push code to GitHub repository
2. Visit: https://streamlit.io/cloud
3. Connect GitHub account
4. Select repository and branch
5. Click "Deploy" - Done!

Alternative: Render[^38]

- Free tier: 750 CPU-hours/month
- Supports Flask/FastAPI backends
- Docker support available

Deployment Checklist:

- ✓ Model files (<50MB for free tier)
- ✓ requirements.txt with all dependencies
- ✓ app.py with Streamlit code
- ✓ README.md with usage instructions
- ✓ Public URL accessible

Evening Session (2-3 hours): Documentation and Finalization

Step 3.3: GitHub Repository Setup

Repository Structure:

```
ev-battery-health-prediction/
├── README.md
├── requirements.txt
├── app.py
├── notebooks/
│   ├── 01_EDA.ipynb
│   ├── 02_Feature_Engineering.ipynb
│   ├── 03_Model_Training.ipynb
│   └── 04_Model_Evaluation.ipynb
├── models/
│   ├── best_model.pkl
│   ├── scaler.pkl
│   └── model_comparison.csv
├── data/
│   ├── processed/
│   └── raw/
├── images/
│   ├── capacity_degradation.png
│   ├── feature_importance.png
│   └── shap_summary.png
└── docs/
    └── project_report.pdf
```

README.md **Template:**

````
# EV Battery Health Prediction System

Predict State of Health (SoH) and Remaining Useful Life (RUL) of lithium-ion batteries using machine learn

## □ Project Objectives
- SoH prediction with ±5% accuracy
- RUL estimation with &gt;90% recall
- Deployed web application

## □ Datasets
- NASA PCoE Battery Dataset
- CALCE Battery Degradation Dataset

## □ Models Implemented
- **Supervised:** Random Forest, XGBoost, LSTM, GRU
- **Unsupervised:** K-Means Clustering
- **Reinforcement:** Q-Learning for charging optimization

## □ Performance
| Model | RMSE | R² | Accuracy |
|-------|------|-----|----------|
| GRU | 1.1% | 0.97 | 97% |
| LSTM | 1.2% | 0.96 | 96% |

## □ Deployment
Live demo: [Hugging Face Spaces](https://huggingface.co/spaces/YOUR_USERNAME/ev-battery-health-predictor)

## □ Installation
```bash
pip install -r requirements.txt
streamlit run app.py
````
````

License


```
**Day 3 Deliverables:**
- ✓ Working Streamlit application
- ✓ Deployed app with public URL
- ✓ Complete GitHub repository with documentation
- ✓ Project report (PDF/Markdown)

---

## Key Performance Indicators (KPIs)

### Model Performance KPIs[^1][^7]

| KPI | Target Value | Measurement Method |
|-----|--------------|-------------------|
| **SoH Prediction Accuracy** | ±5% or better | \|Predicted - Actual\| / Actual |
| **RUL Prediction RMSE** | &lt;5% of max cycles | \(\sqrt{\text{mean}((\text{pred} - \text{actual})^2)}\)
| **Mean Absolute Error** | &lt;2% of capacity | \(\text{mean}(\|\text{pred} - \text{actual}\|)\) |
| **R² Score** | &gt;0.95 | \(1 - \frac{SS_{res}}{SS_{tot}}\) |
| **Failure Detection Recall** | &gt;90% | TP / (TP + FN) |

### Data Quality KPIs[^1]

| KPI | Target Value | Measurement |
|-----|--------------|-------------|
| **Missing Data Rate** | &lt;5% | (Missing / Total) × 100 |
| **Data Consistency** | &gt;95% | Validation checks |
| **Feature Coverage** | &gt;90% | (Available / Required) × 100 |

### Operational KPIs[^75][^78]

| KPI | Target Value | Purpose |
|-----|--------------|---------|
| **Inference Latency** | &lt;100ms | Real-time predictions |
| **Model Size** | &lt;50MB | Deployment efficiency |
| **API Response Time** | &lt;500ms | User experience |
| **System Uptime** | &gt;99% | Application reliability |

---

## Algorithm Summary

### Supervised Learning[^2][^7][^51]

1. **Random Forest Regression**
   - Ensemble of decision trees
   - Feature importance analysis
   - RMSE: 1.5-2.0%

2. **XGBoost Regression**
   - Gradient boosting
   - Handles missing data
   - RMSE: 1.2-1.5%

3. **LSTM (Long Short-Term Memory)**[^2][^4][^77]
   - Recurrent neural network
   - Time-series prediction
   - RMSE: 1.0-1.3%
   - Best for sequential data

4. **GRU (Gated Recurrent Unit)**[^77][^80]
   - Simplified LSTM
   - Faster training
   - RMSE: 0.7-1.1%
   - Average: 0.724%[^77]

### Unsupervised Learning[^59][^62]
```

```
1. **K-Means Clustering**
   - Groups batteries by degradation patterns
   - Optimal K: 3-5 clusters
   - Applications: Proactive maintenance strategies

2. **DBSCAN Clustering**
   - Density-based outlier detection
   - Identifies abnormal batteries

3. **PCA (Principal Component Analysis)**
   - Dimensionality reduction
   - 2D/3D visualization of degradation space

### Reinforcement Learning[^60][^63][^66]

1. **Q-Learning**
   - Tabular RL for discrete states
   - Optimal charging policy
   - Reduces degradation by 10-15%

2. **Deep Q-Network (DQN)**
   - Deep RL for continuous states
   - Experience replay
   - Better scalability

3. **Proximal Policy Optimization (PPO)**[^63]
   - State-of-the-art policy gradient
   - Stable training
   - Real-time charging optimization

---

## Free Deployment Platforms Comparison[^38][^41]

| Platform | Free Tier | Best For | Limitations |
|----------|-----------|----------|-------------|
| **Hugging Face Spaces** | 2 vCPU, 16GB RAM | ML models, large community | Apps sleep after inactivity |
| **Streamlit Community Cloud** | Unlimited public apps | Quick demos, GitHub integration | ~1hr idle slee
| **Render** | 750 CPU-hours/month | Flask/FastAPI backends | 15min idle sleep |
| **Railway** | $5 credit | Easy deployment | Credit exhausts quickly |
| **Gradio on HF** | Same as HF Spaces | Interactive demos | Public only |

**Recommendation:** **Hugging Face Spaces** or **Streamlit Community Cloud**[^38][^39][^41]

---

## Critical Success Factors

### Technical Excellence
- Achieve **±5% SoH prediction accuracy**[^1]
- Maintain **&lt;5% missing data** rate[^1]
- Ensure **&gt;90% failure detection recall**[^1]
- Keep **inference latency &lt;100ms**[^75]

### Time Management
- **Day 1:** Data + EDA + Baseline (9-12 hours)
- **Day 2:** Deep learning + Clustering + RL (9-12 hours)
- **Day 3:** Deployment + Documentation (8-10 hours)
- **Total:** ~30 hours over 3 days

### Quality Assurance
- Validate models on unseen test batteries[^2][^19]
- Use cross-validation (5-fold recommended)
- Test app before public deployment
- Document all assumptions and limitations

---

## Common Pitfalls and Solutions

### Data Issues
```

```
**Problem:** Missing temperature/voltage data
**Solution:** Forward fill interpolation or use median imputation[^79]

**Problem:** Different battery chemistries in dataset
**Solution:** Train separate models per chemistry or use transfer learning

### Model Issues
**Problem:** Overfitting on training data
**Solution:** Early stopping, dropout layers, cross-validation[^80]

**Problem:** Poor RUL prediction at end-of-life
**Solution:** Use knee point detection and piecewise models[^25]

### Deployment Issues
**Problem:** Model file too large (&gt;50MB)
**Solution:** Quantization, pruning, or use model APIs[^38]

**Problem:** App crashes on HF Spaces
**Solution:** Check requirements.txt versions, use requirements freeze

---

## Additional Resources

### Research Papers
1. NASA Battery Dataset Analysis[^2][^4][^7]
2. LSTM/GRU for SoH Prediction[^77][^80]
3. Reinforcement Learning for EV Charging[^60][^63]

### Code Repositories
- NASA Battery Analysis[^19]
- LSTM SOH Prediction[^77]
- Battery Health Monitoring[^76]

### Deployment Guides
- Hugging Face Spaces Tutorial[^39][^42]
- Streamlit Deployment[^38][^41]

---

## Conclusion

This 3-day capstone project provides a **comprehensive, production-ready** battery health prediction syster

✅ Build **state-of-the-art ML models** (LSTM/GRU) with &lt;1.5% RMSE
✅ Implement **all three learning paradigms** (supervised, unsupervised, RL)
✅ Deploy a **live web application** accessible worldwide
✅ Create a **portfolio-worthy project** for job applications

**Key Takeaway:** Focus on **SoH and RUL prediction** as core objectives, use **NASA/CALCE datasets** for c

Good luck with your capstone project! □□

---

## References

[^1] Capstone Project Document - Success Metrics and KPIs
[^2] MDPI - LSTM Method for SOH Prediction (NASA/CALCE datasets)
[^4] MDPI - Early-Stage SOH Prediction using LSTM
[^7] MDPI - Multi-Feature Analysis with LSTM
[^16][^20][^24] CALCE Battery Dataset - Degradation Testing
[^17] Wipro - Battery Health Forecasting (96-97% accuracy)
[^19] GitHub - SOH Prediction using NASA Dataset
[^23] MathWorks - Battery SOH Estimation
[^25] PMC - New Energy EV Battery Prediction (K-means clustering)
[^38] KDNuggets - 7 Free ML Hosting Platforms
[^39] Shafiqul AI - Deploy Streamlit to HF Spaces Guide
[^41] LinkedIn - 5 Free Deployment Platforms
[^42] KDNuggets - Deploy LLM to HF Spaces
[^51][^56] PMC - Random Forest for Battery Health
[^52][^57] PMC - Health Index Informed Attention Model
```

```
[^59] Espublisher - K-means Clustering Framework
[^60] ScienceDirect - Quantum RL for EV Charging
[^62] Bioengineer - Deep Learning Battery Clustering
[^63] arXiv - Physics-Informed RL for EV Charging
[^66] IEEE - DQN for EV Charging Optimization
[^71] arXiv - Domain Knowledge-Guided ML Framework
[^75] TLS Containers - BESS KPIs Guide
[^76] GitHub - Oxford Battery SOC Prediction
[^77] GitHub - LSTM/GRU SOH Prediction (0.724% RMSE)
[^78] Peaxy - Battery Health KPIs (Anomaly Detection)
[^79] IEEE - Data-Driven EV Battery SOC Estimation
[^80] ScienceDirect - LSTM-GRU Compressed Model

<span>[^10][^11][^12][^13][^14][^15][^18][^21][^22][^26][^27][^28][^29][^3][^30][^31][^32][^33][^34][^35][

<div>⁂</div>

[^1]: Intelligent-Predictive-Maintenance-and-Battery-Health-Forecasting-System-for-Electric-Vehicles.docx
[^2]: https://www.mdpi.com/2313-0105/9/3/177/pdf?version=1679138840
[^3]: https://onlinelibrary.wiley.com/doi/10.1002/eng2.70073
[^4]: https://www.mdpi.com/1424-8220/25/7/2275
[^5]: http://arxiv.org/pdf/2403.05430.pdf
[^6]: https://pmc.ncbi.nlm.nih.gov/articles/PMC9208921/
[^7]: https://www.mdpi.com/2313-0105/9/2/120/pdf?version=1675858007
[^8]: https://www.mdpi.com/1996-1073/12/4/660/pdf?version=1550801852
[^9]: https://www.tandfonline.com/doi/pdf/10.1080/15435075.2023.2299402?needAccess=true
[^10]: https://www.scribd.com/document/912678925/Capstone-Project-Guide-1
[^11]: https://www.kct.ac.in/wp-content/uploads/2024/03/R2018A-2022-Batch-onwards.pdf
[^12]: https://www.bennett.edu.in/wp-content/uploads/2025/03/New-Syllabus-Booklet_BTech-CSE-2024-28.pdf
[^13]: https://presidencyuniversity.in/uploads/images/68622227dead31751261735.pdf
[^14]: https://www.scribd.com/document/742773132/Capstone-Project-Guidelines-data-science
[^15]: https://www.sciencedirect.com/science/article/pii/S2950264025000930
[^16]: https://calce.umd.edu/battery-data
[^17]: https://www.wipro.com/analytics/electric-vehicle-battery-health-forecasting/
[^18]: https://indiraicem.ac.in/assets/pdf/syllabus/MCA-(2024-26-Course)-Pattern-Syllabus.pdf
[^19]: https://github.com/anirudhkhatry/SOH-prediction-using-NASA-Dataset
[^20]: https://calce.umd.edu/data
[^21]: https://aws.amazon.com/solutions/guidance/electric-vehicle-battery-health-prediction-on-aws/
[^22]: https://www.upgrad.com/blog/data-science-roadmap/
[^23]: https://www.mathworks.com/help/predmaint/ug/battery-second-life-app-soh-estimation.html
[^24]: https://calce.umd.edu/battery-accelerated-cycle-life-testing-data
[^25]: https://pmc.ncbi.nlm.nih.gov/articles/PMC10770444/
[^26]: https://act.edu.in/wp-content/uploads/2025/08/List-of-NM-COursese-Annexure-List-of-Courses-05.09.202
[^27]: https://ieeexplore.ieee.org/document/10604252/
[^28]: https://ioft-data.engin.umich.edu/calce-battery-datasets/
[^29]: https://www.geotab.com/blog/ev-battery-health/
[^30]: https://arxiv.org/pdf/2212.03332.pdf
[^31]: https://arxiv.org/html/2309.15719
[^32]: https://arxiv.org/pdf/2501.06226.pdf
[^33]: http://arxiv.org/pdf/2208.09751.pdf
[^34]: https://arxiv.org/pdf/2308.04328.pdf
[^35]: https://arxiv.org/pdf/2403.18203.pdf
[^36]: https://arxiv.org/pdf/2211.14417.pdf
[^37]: https://arxiv.org/pdf/2403.00787.pdf
[^38]: https://www.kdnuggets.com/7-best-free-platforms-to-host-machine-learning-models
[^39]: https://shafiqulai.github.io/blogs/blog_4.html
[^40]: https://www.gradio.app/guides/deploying-gradio-with-disco
[^41]: https://www.linkedin.com/pulse/5-free-platforms-deploy-your-machine-learning-models-2025-barnwal-mvp
[^42]: https://www.kdnuggets.com/how-to-deploy-your-llm-to-hugging-face-spaces
[^43]: https://www.gradio.app/guides/running-gradio-on-your-web-server-with-nginx
[^44]: https://www.domo.com/learn/article/ai-model-deployment-platforms
[^45]: https://huggingface.co/docs/hub/en/spaces-sdks-streamlit
[^46]: https://www.gradio.app/guides/quickstart
[^47]: https://www.truefoundry.com/blog/model-deployment-tools
[^48]: https://discuss.streamlit.io/t/using-huggingface-with-streamlit/75662
[^49]: https://serve.scilifelab.se/docs/application-hosting/gradio/
[^50]: https://arxiv.org/pdf/2504.05728.pdf
[^51]: https://downloads.hindawi.com/journals/ijer/2023/9922475.pdf
[^52]: https://pmc.ncbi.nlm.nih.gov/articles/PMC10007287/
[^53]: https://pmc.ncbi.nlm.nih.gov/articles/PMC8987387/
[^54]: https://pmc.ncbi.nlm.nih.gov/articles/PMC11620055/
[^55]: https://linkinghub.elsevier.com/retrieve/pii/S0378775322001331
```

[^56]: https://pmc.ncbi.nlm.nih.gov/articles/PMC11842581/
[^57]: https://www.mdpi.com/1424-8220/23/5/2587/pdf?version=1677397526
[^58]: https://www.riverpublishers.com/downloadchapter.php?file=RP_9788770040723C156.pdf
[^59]: https://www.espublisher.com/uploads/article_html/engineered-science/10.30919-es1317.htm
[^60]: https://www.sciencedirect.com/science/article/pii/S0306261925000091
[^61]: https://www.jetir.org/papers/JETIR2304796.pdf
[^62]: https://bioengineer.org/deep-learning-advances-lithium-ion-battery-estimation-and-clustering/
[^63]: https://arxiv.org/html/2510.12335v1
[^64]: https://www.nature.com/articles/s44296-024-00011-1
[^65]: https://www.sciencedirect.com/science/article/pii/S2666546824000752
[^66]: https://ieeexplore.ieee.org/document/10775525/
[^67]: https://www.sciencedirect.com/science/article/abs/pii/S0360544224013136
[^68]: https://www.nature.com/articles/s44172-025-00488-1
[^69]: https://www.sciencedirect.com/science/article/pii/S2352467723001704
[^70]: https://www.mdpi.com/2313-0105/9/7/351/pdf?version=1688208721
[^71]: https://arxiv.org/html/2409.14575v1
[^72]: https://arxiv.org/pdf/2410.19886.pdf
[^73]: https://pmc.ncbi.nlm.nih.gov/articles/PMC8294100/
[^74]: https://pmc.ncbi.nlm.nih.gov/articles/PMC11700096/
[^75]: https://www.tls-containers.com/tls-blog/comprehensive-guide-to-key-performance-indicators-of-energy-
[^76]: https://github.com/nitish20899/Oxford-Lithium-Ion-Battery-SOC-prediction-AI
[^77]: https://github.com/sileneer/NRP_2022_EEE12
[^78]: https://peaxy.net/battery-health-checks/
[^79]: https://ieeexplore.ieee.org/iel8/6287639/6514899/10876145.pdf
[^80]: https://www.sciencedirect.com/science/article/abs/pii/S2352152X25013544
[^81]: https://www.diva-portal.org/smash/get/diva2:1371050/FULLTEXT01.pdf
[^82]: https://onlinelibrary.wiley.com/doi/10.1002/cae.70099?af=R
[^83]: https://dl.acm.org/doi/fullHtml/10.1145/3489088.3489092
[^84]: https://www.twaice.com/battery-encyclopedia/key-performance-indicator-kpi
[^85]: https://www.kaggle.com/datasets/patrickfleith/nasa-battery-dataset/code
[^86]: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5616552