We have successfully developed a Remaining Useful Life (RUL) prediction model for bearing health monitoring using the NASA IMS bearing dataset. The model achieves 98.82% accuracy and is ready for deployment on Red Pitaya with accelerometer and vibration sensors.

Model Performance

- Model Type: Random Forest Regressor
- Accuracy: $R^2 = 0.9882$ (98.82%)
- Mean Absolute Error: 35.36 time units
- Dataset: NASA IMS bearing dataset (2,155 samples, 96 features)
- Training Data: Real bearing degradation data from laboratory tests

Sensor Requirements

Hardware Setup

- Primary Sensor: Accelerometer (measures acceleration in g-force)
- Secondary Sensor: Vibration sensor (measures displacement/velocity)
- Mounting: Both sensors on the same bearing
- Sampling Rate: 34.1 Hz (same as NASA dataset)
- Update Rate: Every 10-60 seconds (configurable)

Sensor Specifications

- Accelerometer: 2-axis or 3-axis accelerometer
- Vibration Sensor: Compatible with bearing monitoring
- Data Interface: Analog or digital output to Red Pitaya
- Calibration: Standard industrial calibration

Feature Extraction Requirements

Required Features per Sensor (8 features each)

Time-Domain Features:

1. RMS (Root Mean Square) - $\text{sqrt}(\text{mean}(\text{data}^2))$
1. **Peak - max(abs(data))**

1. Mean - mean(data)
1. Standard Deviation - std(data)
1. **Kurtosis - kurtosis(data) (measure of peakiness)**
1. **Skewness - skew(data) (measure of asymmetry)**
1. Crest Factor - peak / rms
1. **Entropy - Shannon entropy of data distribution**

Total Feature Count

- Accelerometer: 8 features
- Vibration Sensor: 8 features
- Total: 16 features per prediction

Feature Calculation Details

python

```python
# Example feature calculation for accelerometer data

def calculate_features(sensor_data):

    features = {}

    # RMS

    features['rms'] = np.sqrt(np.mean(sensor_data**2))

    # Peak

    features['peak'] = np.max(np.abs(sensor_data))

    # Mean

    features['mean'] = np.mean(sensor_data)

    # Standard Deviation

    features['std'] = np.std(sensor_data)

    # Kurtosis

    features['kurtosis'] = stats.kurtosis(sensor_data)

    # Skewness
```

features['skew'] = stats.skew(sensor_data)

 # Crest Factor

features['crest'] = features['peak'] / features['rms']

 # Entropy

features['entropy'] = entropy(pd.cut(sensor_data, 500).value_counts())

return features

Data Processing Pipeline

1. Data Collection

Raw Sensor Data → Feature Extraction → Feature Scaling → RUL Prediction → Time Conversion

2. Feature Extraction Process

1. Collect raw data from both sensors (minimum 1000 samples per update)
1. Calculate 8 features for each sensor
1. Combine features into single feature vector (16 total)
1. Scale features using pre-trained scaler
1. Predict RUL using trained model
1. Convert to readable time (hours, days, weeks, months)

3. Real-Time Implementation

Red Pitaya Implementation Example

def predict_rul_realtime(*accel_data*, *vib_data*):

 # 1. Extract features

accel_features = extract_features(accel_data)

```python
    vib_features = extract_features(vib_data)

    # 2. Combine features

    combined_features = accel_features + vib_feature

    # 3. Load model and predict

    predictor = joblib.load('smart_rul_predictor.pkl')

    result = predictor.predict_rul_smart(combined_features

    # 4. Return formatted result

    return result['formatted']  # e.g., "2 weeks 3 days"
```

Model File

File: smart_rul_predictor.pkl

- Size: ~15MB
- Contents: Trained model + scaler + time conversion
- Format: Python pickle file
- Compatibility: Python 3.7+

Model Loading

```python
import joblib

predictor = joblib.load('smart_rul_predictor.pkl')
```

Output Format

Smart Time Conversion

The model automatically chooses the most appropriate time unit:

- < 24 hours: "45 hours 30 minutes"
- 1-7 days: "3 days 12 hours"
- 1-4 weeks: "2 weeks 3 days"
- > 1 month: "1 month 15 days"