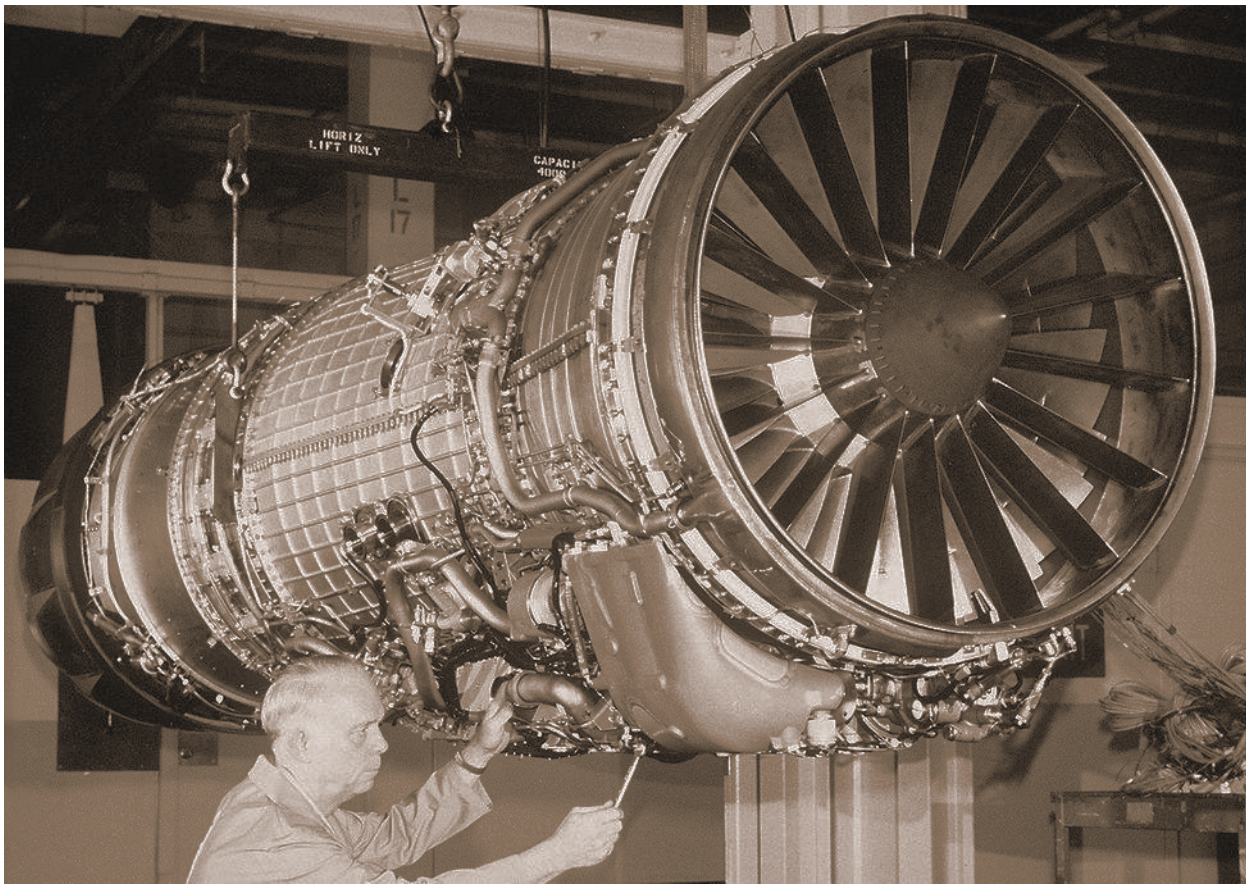
Low Level Design (LLD)

Predictive Maintenance - NASA Turbofan Jet Engine RUL Prediction



Author : Kunal Lokhande

February 2025

iNeuron Internship

End to End Machine Learning Internship Project

Document version control

Date issued - February 2025

Version - 0.0.1

Description - Low Level Design Document

Author - Kunal Lokhande

Table of contents

Abstract

Machine learning is revolutionizing industries by enabling data-driven decision-making and predictive analytics. In aviation, predictive maintenance plays a crucial role in optimizing engine performance and reducing operational costs. This project focuses on leveraging machine learning to estimate the **Remaining Useful Life (RUL)** of aircraft engines, ensuring timely maintenance and minimizing downtime.

The study involves **data exploration, feature engineering, and model development** using advanced regression and deep learning techniques. By analyzing sensor data from aircraft engines, the model predicts degradation patterns, helping airlines schedule maintenance proactively. Additionally, the aviation industry faces challenges such as fluctuating demand, safety regulations, and dynamic pricing strategies. This research also explores key factors influencing flight ticket pricing, including demand-supply dynamics, booking trends, and operational constraints.

The findings aim to enhance **predictive maintenance efficiency** while providing insights into **cost optimization** for airline operations. The integration of machine learning in aviation not only improves safety but also drives economic sustainability in a highly competitive sector.

1. Introduction

1.1 Why this Low-Level Design Document?

The Low-Level Design (LLD) document provides the detailed technical blueprint for implementing the Remaining Useful Life (RUL) Prediction System for NASA turbofan engines. It translates high-level architectural decisions into executable specifications for developers, ensuring accurate and efficient coding.

1. Code-Level Clarity

- Defines class diagrams, methods, and relationships (e.g., DataPreprocessor, RUL Model).

- Specifies input/output contracts for each module (e.g., CSV → cleaned data → features).

2. Direct Implementation Guidance

- Enables programmers to write code directly from the document.

- Includes database schemas, API endpoints, and algorithm hyperparameters.

3. System Integrity

- Describes inter-module interactions (e.g., how FeatureEngineering feeds into ModelTraining).

- Covers error handling, logging formats, and performance thresholds.

4. Experimentation & Deployment

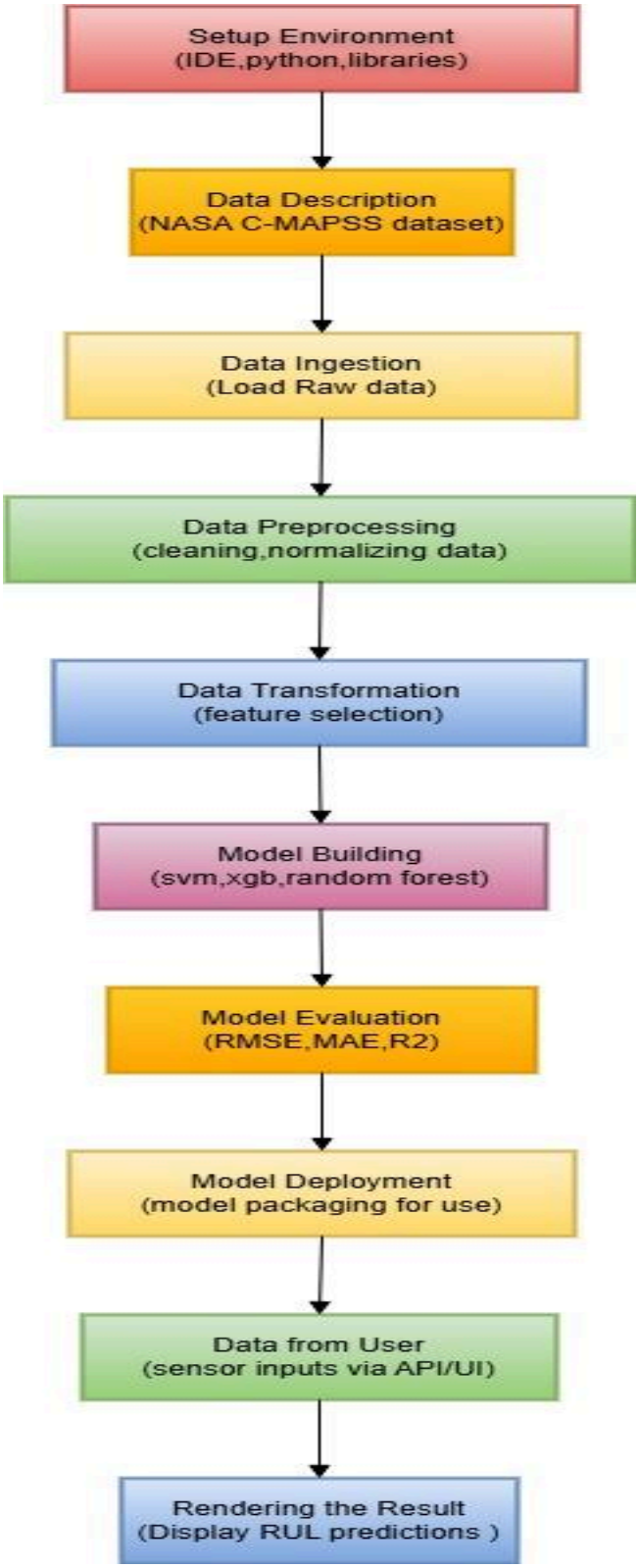
- Outlines model training workflows (e.g., SVM vs Random Forest).

- Specifies cloud deployment and CI/CD pipelines.

1.2 Scope

The LLD provides a component-level technical blueprint, detailing data structures, software architecture, and source code specifications for the RUL prediction system. It refines the HLD into executable modules—including database schemas, class diagrams, and API contracts—enabling direct implementation by developers. The document covers application flow, error handling, and performance algorithms while maintaining traceability to high-level requirements. Finally, it defines deployment protocols (AWS/CI-CD) and validation metrics to ensure system robustness.

2. Architecture



3. Architecture Description

3.1 Data Description

The dataset consists of **run-to-failure simulation data** from NASA's C-MAPSS (Commercial Modular Aero-Propulsion System Simulation) program, which models turbofan engine degradation under varied operational conditions and fault modes. It includes **20,631 cycles** of sensor recordings across **26 columns**, tracking parameters such as operational settings (3 features), sensor measurements (21 features), unit IDs, and cycle counts. Collected by NASA Ames' Prognostics Center of Excellence, this data enables the prediction of **Remaining Useful Life (RUL)**—the number of operational cycles before engine failure—critical for proactive maintenance. The dataset is publicly available on Kaggle and serves as a benchmark for predictive maintenance algorithms in aviation.

The columns correspond to: ● unit number ● time, in cycles ● operational setting 1 ● operational setting 2 ● operational setting 3 ● sensor measurement 1 ● sensor measurement 2 ● ……. ● sensor measurement 26

3.2 Data Ingestion

This module loads the NASA C-MAPSS dataset (mostly CSV format) from Kaggle or local storage, validating file integrity and structure before downstream processing. It converts raw sensor readings into a Pandas DataFrame, ensuring unit consistency and timestamp alignment for all 26 features across 20,631 operational cycles.

3.3 Data Preprocessing

This module applies Robust Scaling (IQR) to sensor data, effectively minimizing outlier impact while preserving degradation trends. It imputes missing values using median-based interpolation and extracts key cyclical features for RUL modeling.

3.4 Data Transformation

This module converts preprocessed sensor data into model-ready features by applying PCA for dimensionality reduction and generating lagged variables to capture temporal degradation patterns. It outputs structured sequences (sliding windows) optimized for LSTM/Random Forest training while maintaining engine cycle alignment.

3.5 Model Building

This stage develops machine learning models including Support Vector Machine, Random Forest, Gradient Boosting and XGBoost, trained on preprocessed sensor data to predict RUL. Hyperparameters are tuned via randomised search to optimize prediction accuracy while preventing overfitting.

3.6 Model Evaluation

The trained models are rigorously evaluated using metrics like MAE, RMSE, and R^2 on a held-out test set to assess RUL prediction accuracy. Performance is validated through cross-validation and compared against baseline approaches to ensure robustness.

3.7 Model Deployment

The trained model is deployed as a local Flask web application, accessible through a browser interface at http://localhost:5000 or http://localhost:5001 . The frontend displays real-time RUL predictions with interactive visualizations using Plotly Dash, while Flask handles all prediction logic.

3.8 Data From User

Users submit engine sensor readings via CSV upload or manual entry through a web form, which validates data format against expected 26-feature schema. The system automatically aligns submissions with preprocessing standards (Robust Scaling, cycle counting) before RUL prediction.

3.9 Rendering the Results

The system displays RUL predictions through an interactive dashboard, visualizing remaining cycles with color-coded alerts (green/yellow/red) and trend graphs of key sensor degradations. Users can export results as PDF reports containing prediction details and maintenance recommendations.

4. Unit Test Cases

Test Case Description	Pre-Requisite	Expected Result
Verify whether the Application URL is accessible to the user	1. Application URL should be defined	Application URL should be accessible to the user
Verify whether the Application loads completely for the user when the URL is accessed	1. Application URL is accessible 2. Application is deployed	The Application should load completely for the user when the URL is accessed
Verify whether the User is able to sign up in the application	1. Application is accessible	The User should be able to sign up in the application
Verify whether user is able to successfully login to the application	1. Application is accessible 2. User is signed up to the application	User should be able to successfully login to the application
Verify whether user is able to see input fields on logging in	1. Application is accessible 2. User is signed up to the application 3. User is logged in to the application	User should be able to see input fields on logging in
Verify whether user is able to edit all input fields	1. Application is accessible 2. User is signed up to the application 3. User is logged in to the application	User should be able to edit all input fields
Verify whether user gets Submit button to submit the inputs	1. Application is accessible 2. User is signed up to the application 3. User is logged in	User should get Submit button to submit the inputs

	to the application	
Verify whether user is presented with recommended results on clicking submit	1. Application is accessible 2. User is signed up to the application 3. User is logged in to the application	The recommended results should be in accordance to the selections user made
Verify whether the recommended results are in accordance to the selections user made	1. Application is accessible 2. User is signed up to the application 3. User is logged in to the application	User should have options to filter the recommended results as well