Architecture Design Document
Predictive Maintenance

Version: 1.0

Last Date of Revision: 05/07/2024

Contents

Document Version Control

VERSION	DATE ISSUED	DESCRIPTION	AUTHOR
1.0	05/07/24	Initial Architecture Design	Vardan S Kamra

1. Introduction

1.1 Why this Architecture Design Document

This document aims to provide a comprehensive overview of the system architecture for the Predictive Maintenance Project, focusing on the neural network model for predicting engine RUL. This architecture design document serves as a comprehensive guide for developers, contributors, and researchers, providing a clear blueprint for implementing the predictive maintenance system. It ensures consistency, facilitates communication, and helps in identifying potential risks and planning resources effectively.

1.2 Scope

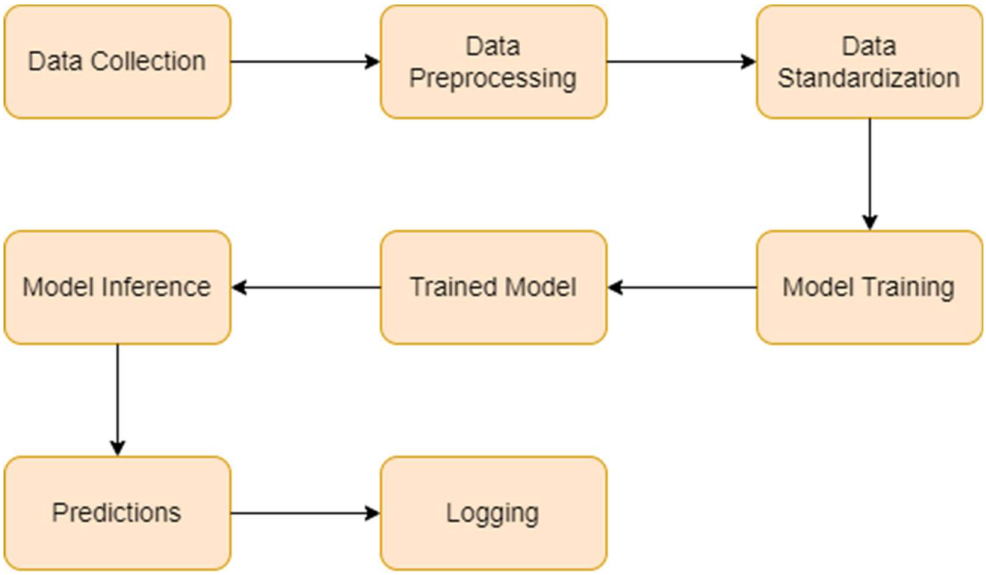
This document covers high-level and detailed architectural views, component interactions, data flow, and the technology stack.

The scope of this architecture design document encompasses the following components and functionalities:

- Data Ingestion and Preprocessing: Reading raw data files and standardizing them.
- Feature Engineering: Extracting and creating features from the raw data for model training.
- Model Training and Evaluation: Training machine learning models and evaluating their performance.
- Model Deployment: Saving trained models and using them for real-time predictions.
- User Interface: Generating visualizations to help understand model performance.
- Testing and Validation: Running automated tests to validate functionality.
- Error Handling and Logging: Managing errors and logging system events.
- Documentation and Maintenance: Providing comprehensive documentation and ongoing maintenance of the system.

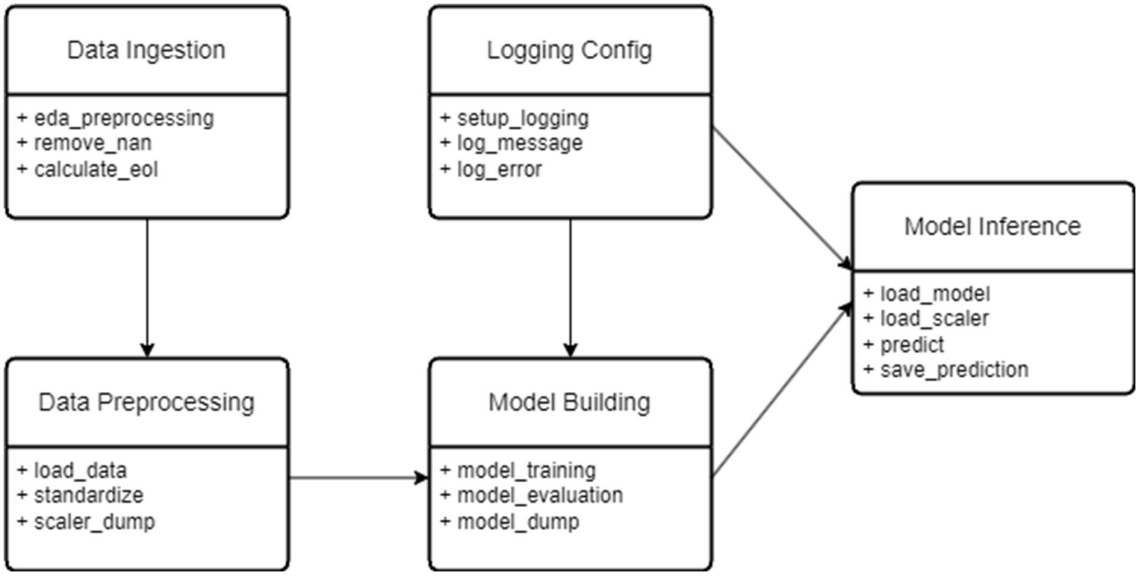
2. Architecture Overview

The Predictive Maintenance system is designed to predict the Remaining Useful Life (RUL) of engines based on historical sensor data. The system involves data preprocessing, feature engineering, machine learning model training, and deployment for real-time predictions.



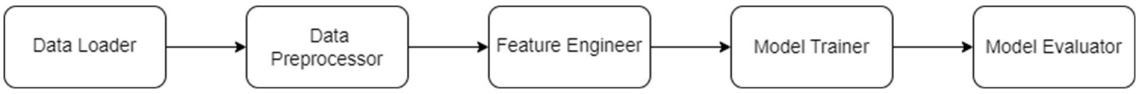
3. Logical View

The logical view describes the system's functional requirements and how they are implemented in the design.



4. Process View

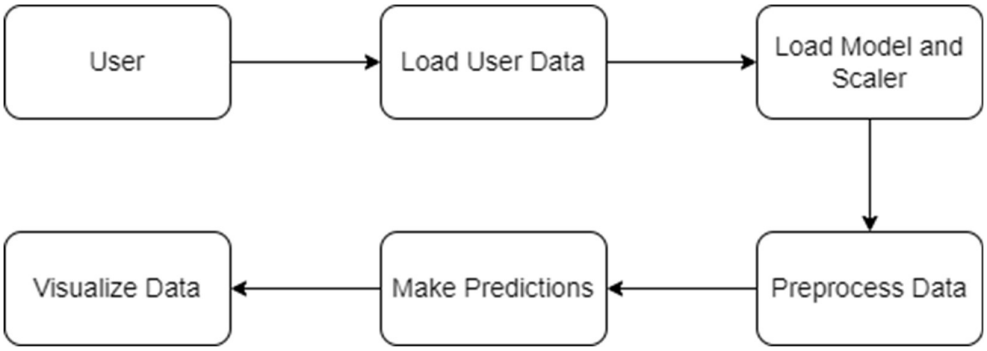
The process view focuses on the dynamic aspects of the system, explaining how the system's components interact during runtime.



Workflow:

- Data Ingestion and Preprocessing: DataLoader reads raw data files, and Data Preprocessor standardizes the data.
- Feature Engineering: Extracts and creates relevant features.
- Model Training and Evaluation: Model Trainer trains the model, and Model Evaluator assesses its performance.
- Model Deployment: Model Serializer saves the trained model, and Inference Engine loads the model for making predictions.
- User Interface: DataVisualizer generates plots and visualizations.
- Testing and Validation: Test cases run tests to validate system components.
- Error Handling and Logging: Logging Config manages errors, and Logger records system events.

5. Use-Case View



Use-Cases:

- Load Data: User loads engine data files.
- Preprocess Data: System preprocesses the loaded data.
- Deploy Model: User deploys the trained model.
- Make Predictions: System makes RUL predictions using the deployed model.
- Visualize Data: User generates data visualizations.

6. Technology Stack

Technologies Used for Each Component:

- Data Collection: CSV files, pandas.
- Data Preprocessing: pandas, scikit-learn.
- Model Training: TensorFlow, Keras.
- Model Inference: TensorFlow, Keras.
- Logging: Python logging module.

Rationale for Technology Choices:

- pandas: Efficient data manipulation.
- scikit-learn: Robust preprocessing utilities.
- TensorFlow/Keras: Advanced neural network capabilities.
- Python logging: Comprehensive logging capabilities.

7. Conclusion

This architecture design document provides a comprehensive view of the Predictive Maintenance Project's structure and components. It outlines the logical and dynamic aspects of the system, ensuring a clear understanding for all stakeholders. The detailed design will guide the development, testing, and deployment phases, ensuring a robust and scalable system.