Low Level Design Document
Predictive maintenance

Document Control

Date issued	Version	Description	Authors
02-05-22	1	Initial LLD	Shikhar Amar, Hitesh Pathak

Contents

1 Introduction

1.1 What is Low level design document ?

The purpose of a low-level design document (LLDD) is to provide the internal logical structure of the program, which in this case is the RUL detection system for Predictive maintenance of turbofan jet engines. It describes the core components of the program identified during High-level design phase, and the logical relation between them, so that the developers can program the solution and maintain / debug with ease.

1.2 Scope

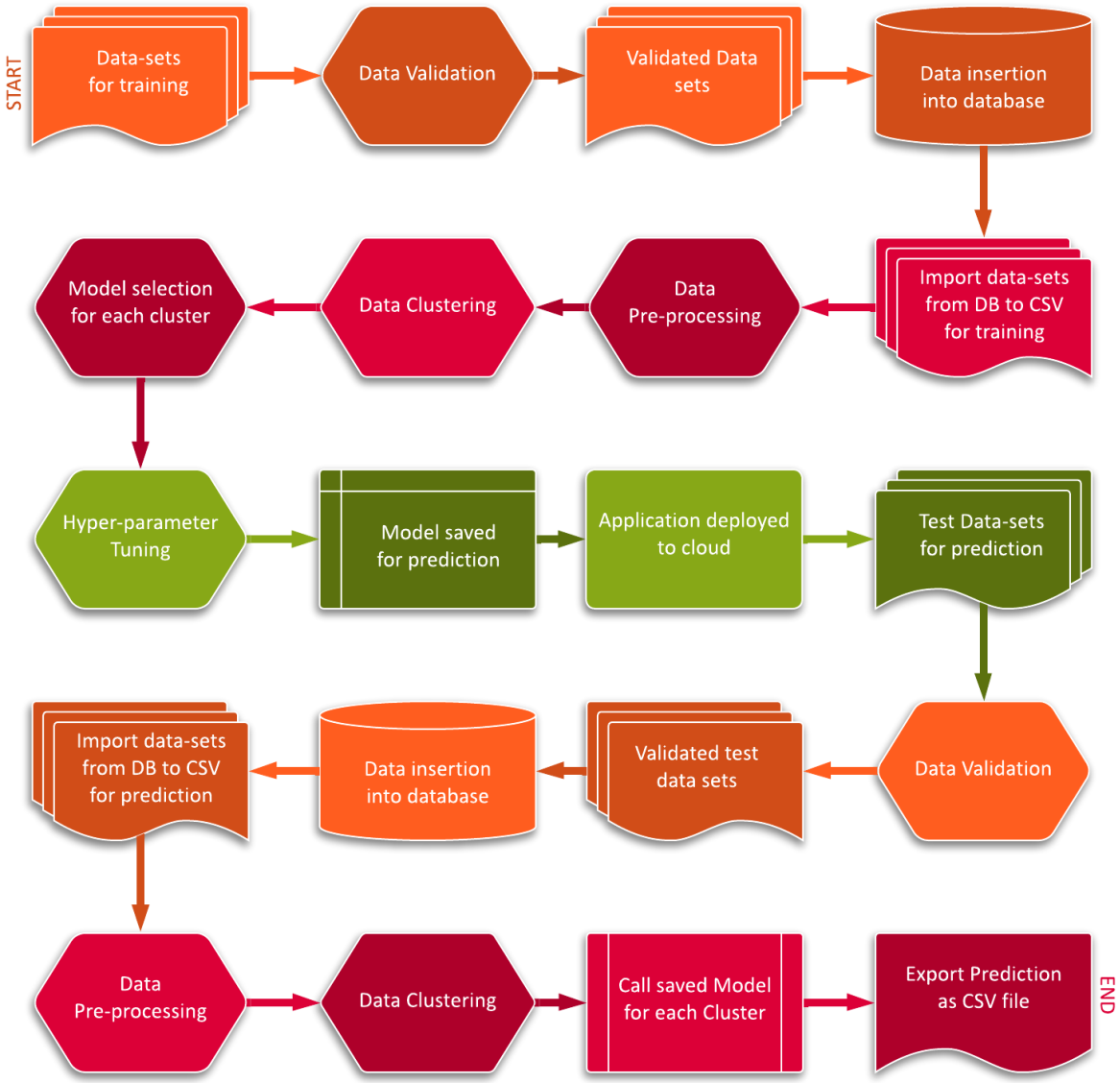
LLD is a design process for components of the system identified in the HLD phase. The process details the architecture and implementation details of all these components and continuously refines the design over the course of development. These components can include data-structures, required software architecture, source code and performance optimization code. The data organization is defined during requirement analysis of project, and refined during data-design phase. After the architecture is build each component is described in detail.

1.3 Definitions

Terms	Meaning
LLD	Low-level design
RUL	Remaining useful life
API	Application programming interface
DB	Database
CSV file	Comma separated values file

2 Architecture

The flow-chart describing the architecture and data-flow through the architectural components is shown below :



3 Architecture description

3.1 Training data description

The training data consists of several data sets, each consisting of multiple multivariate time series, where a unit of time is a cycle of operation of the engine. Each time series in the set corresponds to a different engine, hence the data can be considered to be from a fleet of similar engines. Each engine starts with a different initial condition including manufacturing flaws, variations. As the time series progresses, the engine develops a fault which leads to its breakdown.

The data sets include three operational settings and data from 21 sensors, all columns in the data set indicate different variables as shown :

1. Unit number of engine.
2. Time in cycles of operation.
3. Operational setting 1
4. Operational setting 2
5. Operational setting 3
6. Sensor 1
7. Sensor 2

   ...

26. Sensor 21

3.2 Model Training process flow :

3.2.1 Data validation

Along with the data sets described in 3.1 a *schema* file is also required, which is used for data validation. Schema is a .json file that contains all the necessary metadata about the training data sets - file names, column names, number of columns, datatype of columns. This file is checked against the training data for its validation. The process follows the following steps :

Name validation The schema file contains the specifications for file name, a regex pattern is created according to this specification and the filename of training data sets is matched against the pattern, If a match is found, the corresponding data set is moved to *Good_data_folder* otherwise it is moved to *Bad_data_folder*.

Column name Similarly, the name of the columns in the data sets should match with the specification present in the schema files. Accordingly the file is put into *Good_data_folder* or *Bad_data_folder.*

Number of columns The schema specifies the permissible number of columns present in the data set, the files that validate against schema are moved to *Good_data_folder* and otherwise to *Bad_data_folder*.

Column datatype The datatype of each column should be the same as that specified in the schema file, following which the data is put in *Good_data_folder* and failing which it is put in *Bad_data_folder*.

3.2.2 Insertion of data sets into database

After the training data is validated, it has to be stored in a database. This is achieved following the steps listed below :

Connect to DB system Establish connect to the database system which might be a server running locally or a cloud DB system, depending upon the developers. In our implementation we will be using *DataStax Astra DB* system which uses Apache Cassandra database.

Create database and connect A database is created on Astra DB and a connection is established to the database.

Create data table In the database, multiple tables are created, the schema of the tables i.e., field names, data types are inferred from the *schema* file. If old tables are already present containing previously imported training data, they are not replaced or modified, instead new table are created. So that the models can learn from old as well as new data.

Data insertion The validated training data files are present in *Good_data_folder*, the data from these files is uploaded to the appropriate tables. Since different data sets correspond to different type of training data, multiple columns are mandatory and the data sets should *not* be combined together. If importing a file leads to an error because of the file type (column mismatch, datatype mismatch etc.) the same file is moved to *Bad_data_folder*.

3.2.3 Model training

After the validated data is inserted into the database, it can be used to train models. This process is elaborated below :

Exporting data fromDB The training data sets are present as tables in the database. These tables are exported to a .csv (comma separated values) file, that is used for training.

Data pre-processing In this step, the csv files are loaded into pandas *DataFrame* objects, and the data is cleaned and modified. This involves following sub steps :

- All sensor data columns are checked for standard deviation, if any of them has zero standard deviation, it's deleted from the data set as it does not provide any useful information.
- If any sensor data columns contain *NULL* or missing entries, these are imputed using KNN (k nearest neighbors) Imputer.

Data Clustering Pre-processed data is fed into K-means algorithm to create clusters, the reason behind clustering is to implement different algorithms for the clusters. K-means model is trained over pre-processed data and the model is saved for later predictions. The *Elbow plot* of the data is used to find the optimal number of clusters, this is achieved dynamically using *Knee-locator* function.

Model selection In this step, we find the best model for each cluster of data. Two algorithms are being used in this implementation - *Random Forest* and *XGBoost*. For each algorithm the parameters are calculated using GridSearch, thereafter the AUC scores are evaluated for each of the models and the one with the best score is selected. All models selected using this procedure are saved for predictions.

3.3 Test data description

The test data sets are similar to training data sets. They also include multiple data sets with time series data of a fleet of engines that undergo degradation. The column names and the operational settings are exactly the same, the only difference for test data sets is that the time series of each engine stops sometime before breakdown, unlike the training data sets wherein the data is Run-to-failure.

The end goal of RUL prediction system is the predict the remaining useful life for the engines in test data set, this is equivalent to the number of remaining cycles that the engine can undergo after the last operational cycle in its test series and before it breaks down.

3.4 Prediction process flow

3.4.1 Data validation

Along with the test data, a schema file is required. Schema lists the relevant metadata about data-sets - file names, column names, number of columns, datatype of columns. The validation process is exactly similar to that in the training case 3.2.1.

Name validation The schema file contains the specifications for file name, a regex pattern is created according to this specification and the filename of training data sets is matched against the pattern, If a match is found,

the corresponding data set is moved to *Good_data_folder* otherwise it is moved to *Bad_data_folder*.

Column name Similarly, the name of the columns in the data sets should match with the specification present in the schema files. Accordingly the file is put into *Good_data_folder* or *Bad_data_folder*.

Number of columns The schema specifies the permissible number of columns present in the data set, the files that validate against schema are moved to *Good_data_folder* and otherwise to *Bad_data_folder*.

Column datatype The datatype of each column should be the same as that specified in the schema file, following which the data is put in *Good_data_folder* and failing which it is put in *Bad_data_folder*.

3.4.2 Insertion of data into database

This step is also exactly similar to 3.2.2.

Connect to DB system Establish connect to the database system which might be a server running locally or a cloud DB system, depending upon the developers. In our implementation we will be using *DataStax Astra DB* system which uses Apache Cassandra database.

Create database and connect A database is created on Astra DB and a connection is established to the database.

Create data table In the database, multiple tables are created, the schema of the tables i.e., field names, data types are inferred from the *schema* file. If old tables are already present containing previously imported training data, they are not replaced or modified, instead new table are created. So that the models can learn from old as well as new data.

Data insertion The validated training data files are present in *Good_data_folder*, the data from these files is uploaded to the appropriate tables. Since different data sets correspond to different type of training data, multiple columns are mandatory and the data sets should *not* be combined together. If importing a file leads to an error because of the file type (column mismatch, datatype mismatch etc.) the same file is moved to *Bad_data_folder*.

3.4.3 Prediction of RUL

After the validated data is inserted into the database, the trained ML algorithms are used to model it and predict RUL for the engines. The steps involved are explained below :

Exporting data fromDB The training data sets are present as tables in the database. These tables are exported to a .csv (comma separated values) file.

Data pre-processing In this step, the csv files are loaded into pandas *DataFrame* objects, and the data is cleaned and modified. This involves following sub steps :

- All sensor data columns are checked for standard deviation, if any of them has zero standard deviation, it's deleted from the data set as it does not provide any useful information.
- If any sensor data columns contain *NULL* or missing entries, these are imputed using KNN (k nearest neighbors) Imputer.

Data Clustering The K-means model trained during the clustering of training data is loaded. This model is used to predict clusters of the pre-processed test data sets.

RUL prediction After data is clustered, each cluster is loaded and based on its cluster number, a corresponding ML model is loaded, this model was saved during the training phase. The model is used to predict remaining useful life RUL for all data points in the cluster.
After the predictions are done for all clusters, RUL along with engine number are written to a csv file. This file is then exported to Astra DB.

4 Deployment