Student Declaration of Authorship

HERIOT WATT UNIVERSITY

UK | DUBAI | MALAYSIA

Course code and name:	**F21MP – Master's Project and Dissertation**
Type of assessment:	**Individual**
Coursework Title:	Master's Project and Dissertation
Student Name:	Daniyal Syed
Student ID Number:	H00155307

Declaration of authorship. By signing this form:

- **I declare** that the work I have submitted for individual assessment OR the work I have contributed to a group assessment, is entirely my own. I have NOT taken the ideas, writings or inventions of another person and used these as if they were my own. My submission or my contribution to a group submission is expressed in my own words. Any uses made within this work of the ideas, writings or inventions of others, or of any existing sources of information (books, journals, websites, etc.) are properly acknowledged and listed in the references and/or acknowledgements section.

- I confirm that I have read, understood and followed the University's Regulations on plagiarism as published on the University's website, and that I am aware of the penalties that I will face should I not adhere to the University Regulations.

- I confirm that I have read, understood and avoided the different types of plagiarism explained in the University guidance on Academic Integrity and Plagiarism

Student Signature *(type your name):* *Daniyal Syed*

Date: *20/08/2023*

MACHINE LEARNING IN TRANSPORTATION: PREDICTION OF TRAIN DELAYS

Author:

Daniyal Syed

H00155307

Supervisor:

Dr. Neamat El Gayar

A thesis submitted in fulfilment of the requirements for the degree of MSc Data Science in the

School of Mathematical and Computer Sciences

August 2023

Declaration of Non-Plagiarism

Abstract

The effective management of train delays is critical for ensuring high-quality passenger service and efficient railway planning. In this Master’s thesis, we present a time series model to predict secondary train departure delays that ensue immediately after a primary delay of 30 seconds, originating from the preceding train at a given station within the Dubai Metro system. We wanted to explore the principal factors that allowed us to accurately quantify the impact and influence exerted by primary delays on the subsequent train's departure punctuality which would enable dispatchers to implement effective mitigation measures.

To attain this, we establish a benchmark utilizing the Random Forest Model and compare its performance to different Deep Learning and Machine Learning models, encompassing the LSTM, SVM, and ANN models. After undertaking hyperparameter optimization and model experimentation we found the LSTM model yielded the best RMSE score of 11.785. However, it was noted that the other models displayed similar performance, some with greater computational efficiency, which indicated that the temporal dependencies originating from previous delays at platforms are primarily short-term and have a limited impact. Consequently, employing simpler Machine Learning models can give comparative performances to Deep Learning models.

The project was done in collaboration with the Dubai Road and Transportation Authority (RTA), which provided the data for the project. The results of our research contribute to the field of train delay prediction, offering insights to dispatchers that can better inform their decision-making and help improve passenger service quality.

Acknowledgements

I would like to thank my Supervisor Dr. Neamat El Gayar for her continued support and guidance throughout this project.

I would also like to express my gratitude to the Dubai RTA and Metro Operations team for generously providing the dataset for this project along with their support and willingness to clarify any doubts that contributed to the project's success.

Lastly, I would like to express my deepest gratitude to my family and friends for their encouragement and belief in me throughout this academic pursuit.

Table of Contents

Table of Figures

Table of Tables

List of Acronyms

AI	Artificial Intelligence
ANN	Artificial Neural Network
ARIMA	Autoregressive Integrated Moving Average
ARMA	Autoregressive Moving Average
CNN	Convolutional Neural Network
CPS	Critical Point Search
DELM	Deep Extreme Learning Machine
DL	Deep Learning
DNN	Deep Neural Network
EDA	Exploratory Data Analysis
ETA	Estimated Time of Arrival
FCNN	Fully Connected Neural Network
GP	Gaussian Process
GRU	Gated Recurrent Unit
IQR	Interquartile Range
KNN	K-Nearest Neighbor
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
MASE	Mean absolute Scaled Error
MDAE	Median Absolute Error
MDAPE	Median Absolute Percentage Error
MDSE	Median Scaled Error
MICE	Multivariate Imputation by Chained Equation
ML	Machine Learning
MSE	Mean Squared Error
MLP	Multilayer Perceptron
ReLU	Rectified Linear Unit
RMDSE	Root Median Squared Error
RMSE	Root Mean Squared Error
RMSPE	Root Mean Squared Percentage Error
RMSprop	Root Mean Squared Propagation
RNN	Recurrent Neural Network
RTA	Road and Transport Authority
SARIMA	Seasonal Autoregressive Integrated Moving Average
SGD	Stochastic Gradient Descent
SMAPE	Symmetric Mean Absolute Percentage Error
SVM	Support Vector Machine
VAR	Vector Autoregression

1. Introduction

1.1 Project Background

Rail transportation is a vital part of modern infrastructure, connecting people and goods across the globe today. According to a market research report, the global rail transport market grew from $505.41 billion in 2022 to $538.55 billion in 2023 and is expected to be $677.78 billion by 2027 (The Business Research Company, 2023). Preventing delays in railway trains, whether passenger or freight, is essential to providing service quality and reducing operating costs. According to Lazo, 2019, in her article for *The Washington Post*, the national railroad passenger company of the United States, Amtrak, had over 27 percent of their trains delayed in late 2018. They found that even a 5 percent improvement in on-time performance can have the short-term benefit of savings of $12 million and an annual cost savings and revenue increase of $41.9 million for more substantial and sustained improvements.

Given the costs of delays and the savings reaped by preventing them, railway companies and operators require accurate and robust delay prediction approaches for more effective railway traffic management and planning.

Machine Learning (ML) methods, a subset of the broader Artificial Intelligence (AI) field, have erupted in recent years and are considered amongst the most accurate and efficient at making predictions across various industries. While AI and ML methods have greatly progressed in numerous areas, Tang et al., 2022 argue, they are still in their early stages for the railway sector. Many of the field's research papers have focused on the Maintenance and Inspection sub-domains, with Traffic Planning and Management coming in second, being a relatively more unexplored, but equally important sub-domain.

The application of ML methods to predict train delays has been a rapidly rising trend due to their ability to capture the importance of critical features, moving away from previously used statistical analytic approaches, which required assumptions about the distribution of the underlying data (Tiong et al., 2023). As of this project, the ML-based methods are ever-evolving and have great potential to make accurate predictions for train delays and help rail operators better plan accordingly.

1.2 Motivation

Dubai has transformed its transportation infrastructure at an unprecedented pace in recent decades, positioning itself as a global leader in cutting-edge advancements and unrivaled convenience for commuters across the UAE (Kamarudeen et al., 2020). The establishment and growth of Dubai Metro have played a significant factor in this growth and are an integral part of the city's image. Therefore, effective railway planning and management are essential for ensuring Dubai Metro's quality of service. Since its inception, the Dubai Metro has seen a rapid rise in the number of passengers, as displayed in Fig. 1. This means any train delay has the rising potential to cause a large commuter backlog. Notably, primary train delays stemming from maintenance and operational challenges tend to result in a knock-on effect, propagating to successive trains in the sequence. This phenomenon results in secondary delays, thereby further exacerbating the adverse impact on overall line operations.

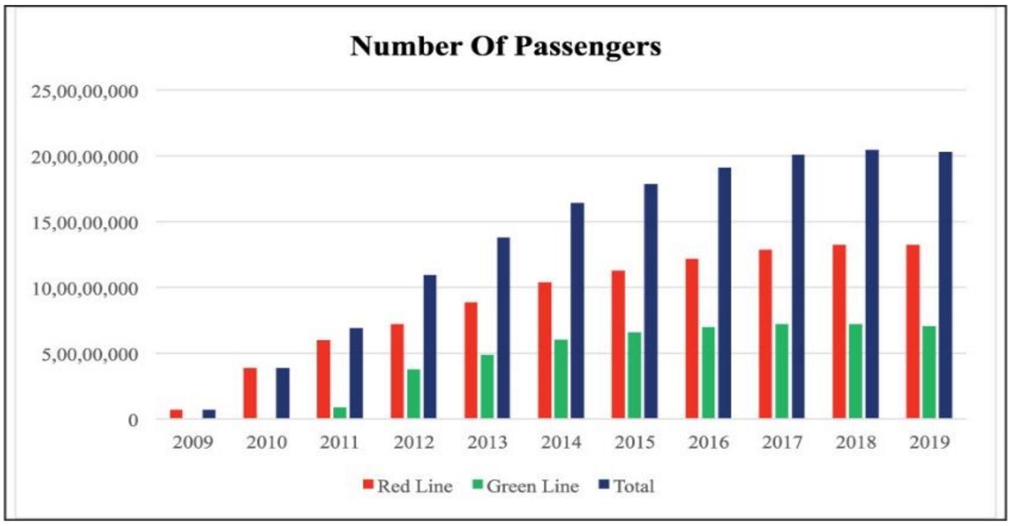


Fig. 1: Passenger growth for Dubai Metro. Adapted from (Kamarudeen et al., 2020).

While several studies have been conducted around the world for predicting train delays, no study for predicting train delays has been conducted in Dubai to the best of this author’s knowledge. Developing a model that predicts train delays can assist dispatchers in making operational decisions. This, in turn, can help mitigate the effects of delays, improve passenger service quality, and enhance network reliability.

1.3 Aims and Objectives

The study aims to leverage supervised deep learning and machine learning techniques to accurately predict secondary train departure delays at a given station, following an initial primary delay of a predetermined duration.

The dataset for this project pertaining to different train operational-related features for predicting delays will be provided by the Dubai Roads and Transport Authority (RTA).

The primary research questions are:

1. What are the most relevant features for accurately predicting train platform departure delays?
2. What pre-processing steps are required to transform the data into a suitable time-series format for forecasting secondary departure delays, aligned with the project’s use case?
3. What Deep Learning Models give the best prediction accuracy, and how do they compare to a benchmark Machine Learning model such as Random Forest?

The primary objectives of the project are as follows:

- Define the pertinent research requirements and acquire the necessary data from RTA to effectively conduct the study.
- Pre-process and prepare the data for analysis and time series modeling, tailored to the project’s use case of predicting secondary delays following primary delays of a predefined duration.
- Explore the trends, features and distinct characteristics present in the data through analysis and visualizations.
- Experiment with different Deep Learning and Machine Learning models to find the optimal hyperparameters that give the highest delay prediction accuracy.
- Compare the models’ performances against each other culminating in the identification and selection of the most optimal model.

1.4 Thesis Structure

The subsequent chapters of this thesis are organized with each chapter further enhancing and expanding on the research project. Chapter 2 constitutes the literature review, wherein the relevant previous studies on predicting train delays are examined. This chapter also covers pre-processing techniques for time series forecasting and details various ML and deep learning techniques for modeling and their relevant evaluation metrics.

In Chapter 3, an overview of the dataset is provided, along with an exploration of its characteristics. A detailed account of the steps taken for data cleansing and pre-processing for modeling, aligned with the project's specific use case, is explained. Furthermore, this chapter expounds on the rationale behind the many experimental setups carried out during the modeling phase.

Subsequently, Chapter 4 presents the outcomes of the experiments accompanied by a critical analysis, which delves into the implications and significance of the results. The challenges and limitations encountered during the implementation are also detailed.

Chapter 5 serves as the conclusion of the thesis and project consolidating and summarizing the undertaken work. Finally, recommendations for further work within the research domain are also delineated.

2. Literature Review

2.1 Overview of Approaches to Predict Train Delays

Many railways worldwide today are operating at a high degree of capacity utilization due to steadily increasing demand and supply, which increases the risk that a delay of even a single train can have a knock-on impact that can propagate to other trains in the entire network (Tiong et al., 2023). The delays to a network from their scheduled time would be detrimental to providing a quality service and have a negative public perception as well as unwanted additional costs. Hence, effective planning by railway companies is a matter of utmost importance. This has led to an increasing rise in research with many different state-of-the-art innovative approaches being explored and evolving over the years.

A previous review of the available literature for the existing research for train delay prediction approaches by Spanninger et al., 2022, has been categorized based on two main types of model paradigms, which are:

1. Event-driven Predictive Models
2. Data-driven Predictive Models

Event-driven approaches explicitly capture pre-defined train-event dependencies, such as departure time, arrival time, or unexpected service disruption. These models rely on a set of rules which define how the system should respond to different events and are multi-step predictions as they can only predict the next event in a sequence rather than an nth event of the series. An example for our case would be a train delay that could only be predicted for the next train in the sequence in one step and would require a chain of predictions or iterations to predict the nth train. Event-driven approaches are mainly modeled by graph models such as Bayesian Networks or equation systems. They are helpful for delay predictions caused by specific events and are relatively simple to implement. They are relatively easy to interpret and implement since they predict based on established events and rules. However, they have limitations when accounting for far more complex trends, or factors leading to delays.

Data-driven approaches, on the other hand, are models that are not based on explicit train-event dependencies, instead they make predictions based on patterns identified in a variety of complex factors, such as scheduled and actual arrival times combined with weather data based on historical data. Most data-driven models are machine learning models. Unlike event-driven approaches, they do not have to have an iterative chain of predictions to reach the desired prediction and can directly predict the delay of the nth train of the sequence in a single step. They are generally more accurate and flexible compared to event-driven approaches as they can capture more complex trends contributing to delays. However, they are more difficult to interpret and understand how exactly the models are making the predictions.

Further review of the literature as well as further project development is limited to data-driven and, more specifically, machine learning approaches. These are more relevant and state-of-the-art to building prediction models which can handle large volumes of data as well as the wide variety and complexity of real-world factors to provide more accurate predictions and insights with the rapid amount of data being produced in the modern day and the future.

2.2 Prediction Scope and Objectives of Data-Driven Models

As previously stated, data-driven approaches do not take explicit train-event dependency structure to model their predictions, which means they are mainly Machine Learning Models. Machine Learning is a technique that allows algorithms to learn from experience through existing data by finding apparent and non-apparent patterns and trends within this data to build a model for prediction-making (Zhou, 2022). They improve the more accurate data provided to them. A review of the different literature leads to further categorization of data-driven models into two main prediction scope categories, which are Long-Term delay predictions and Short-Term delay predictions. As per Tiong et al., 2023, the models are chosen based on the research's final aim, which can be of three implementation levels of significance in railway traffic planning and scope; these include:

- Operational level
- Tactical level
- Strategic level

The three listed implementation levels are sequentially increasing in scope, cost, and complexity. Strategic levels of delay prediction models are for high-level planning that includes investment in future infrastructure planning, such as the improvement of existing rail network facilities. They aim to find the correlation between rail transportation efficiency and the need for infrastructure improvements, or modifications that will provide cost savings and quality improvement by reducing delays.

Tactical level implementations have been done considering the historical train records and delays to allow railway planners to make more accurate and robust timetabling and resource planning (Marković et al., 2015).

Operational level implementations have been done for real-time planning to allow dispatchers to manage disruptions, alert passengers, and adjust schedules according to the level of delay predicted.

Spanninger et al., 2022, have noted that Long-Term delay models are those that aim to make predictions more than 4 hours ahead of a given time. These typically do not deal with a stream of real-time spatiotemporal data from an individual train and instead rely on aggregated historical records, such as train schedules, or weather data to try and gain insight into various train related, or external factors that can be contributing to delays and sub-optimal performance. Due to this higher-level outlook, their level of implementation is mostly at the tactical and strategic levels where planning strategies can be formulated according to the insight provided.

Short-Term delay prediction models are dealing with real-time train operational data, such as real-time train positions, train schedules, and actual and scheduled operational run times as well as potentially considering external data such as the weather. This makes their implementation level at the operational level. A common theme in most short-term delay prediction is that real time-data is required with an emphasis on the data's spatiotemporal representation, which is in other words the space and time data representation for the trains. Due to the focus on spatiotemporal representation, the prediction target for these models is usually represented and limited to delays in space for the current train on the current rail line. The space could be single, or multiple stations ahead. As noted by Tiong et al., 2023, Common prediction targets are further divided and subcategorized for predicting delays in the downstream stations in the following categories:

1. One Station Ahead
2. Multiple Stations Ahead
3. Static Multiple Stations Ahead
4. Dynamic Multiple Stations Ahead

One Station Ahead models predict delays to the Estimated Time of Arrival (ETA) to the next station relative to the current. The Multiple Stations Ahead models predict time delays for a certain number of stations. In contrast, the static and dynamic multiple stations models aim to forecast delays at all stations downstream to the current one. All scheduled delays and further information from upstream stations are also considered for making the predictions. The main difference between the static and dynamic models is that static models take the data from the number of stations ahead to make a firm delay prediction for all or a specified number of downstream stations. On the other hand, dynamic models take the evolving traffic information scenarios and their impact on the current train's delay along its route is adjusted accordingly. The features for the downstream stations ahead are also dynamically adjusted based on the train's current position, such as the number of upstream stations it uses to predict further delay.

An overview of the data-driven prediction model hierarchies for train delay prediction scopes can be referred to in Fig. 2 below.

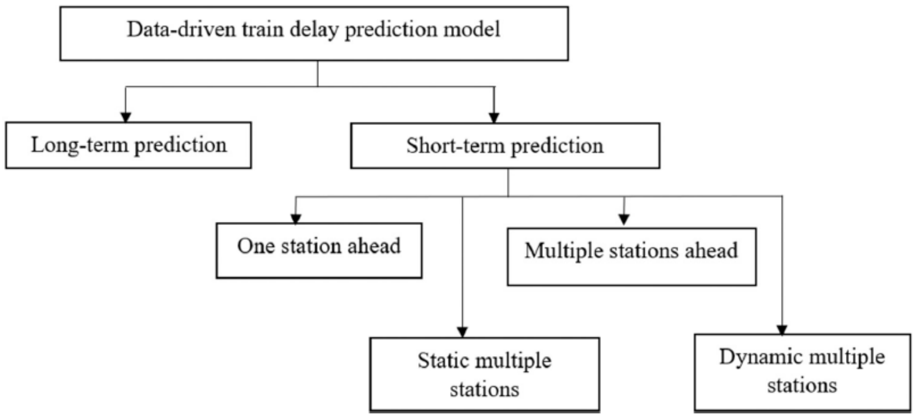


Fig. 2: Overview & Hierarchy of Data-Driven Train Delay Prediction Models in terms of prediction scope Adapted from (Tiong et al., 2023)

This research project will further explore the related work and implementation of different time-series data models for various state-of-the-art data-driven train delay scenarios. These scenarios include the propagation of delays across successive and consecutive trains at a station.

2.3 Time Series Forecasting

2.3.1 Overview

Time-series data is a set of data observations ordered sequentially in regular time steps over a given period (Géron, 2023). The most common aim of time-series data models is to predict future values, in other words: forecasting. Although they can be used for anomaly detection, classification, imputation, and more. Depending on the total number of variables varying over time, the model can be either a univariate time series, which only has one variable varying over time or a multivariate time series, if two or more variables are varying over time. Multivariate time series are usually more complex because they require analyzing relationships between multiple time series and understanding how they interact.

According to Deb et al., 2017, a typical approach when exploring time-series data is to decompose it into its constituent components, each of which can help better understand and analyze the patterns in the data during Exploratory Data Analysis (EDA). The constituents are:

1. Trend
2. Seasonality
3. Residuals

The trend constituent is the general long-term trend over time or the general direction of slope for the time series, as illustrated in Fig 3, which can be increasing, decreasing or flat.

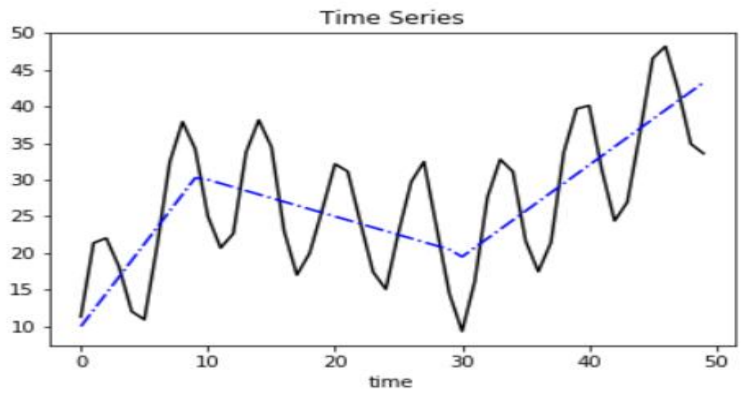


Fig. 3: Visualizing the trend for general time series data. Adapted from (*Ziganto, 2018*)

The seasonal component of the time series represents the regular and repeating patterns that occur in the data over a given fixed period of time, such as daily, weekly, or yearly. The seasonality visualization is shown in Fig. 4.

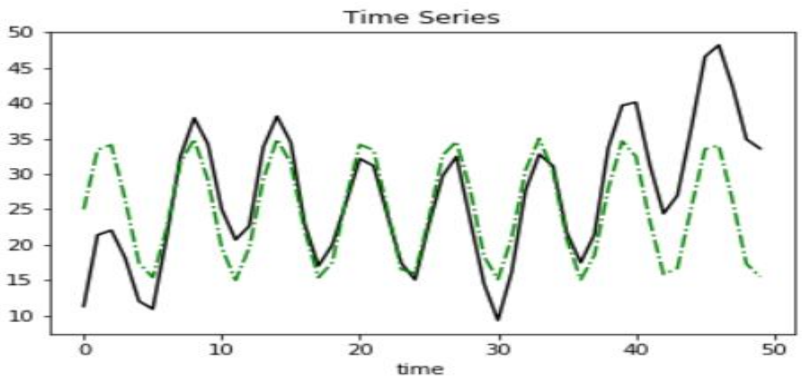


Fig. 4: Visualizing the seasonality for general time series data. Adapted from (*Ziganto, 2018*)

The residual component of the time series, unlike the trend and seasonality, is not associated with time and instead represents the unexplainable noise and random fluctuations present in the data points. This component is obtained by taking the difference between the actual data points and the sum of the trend and seasonal components, as shown in Fig. 5 below.

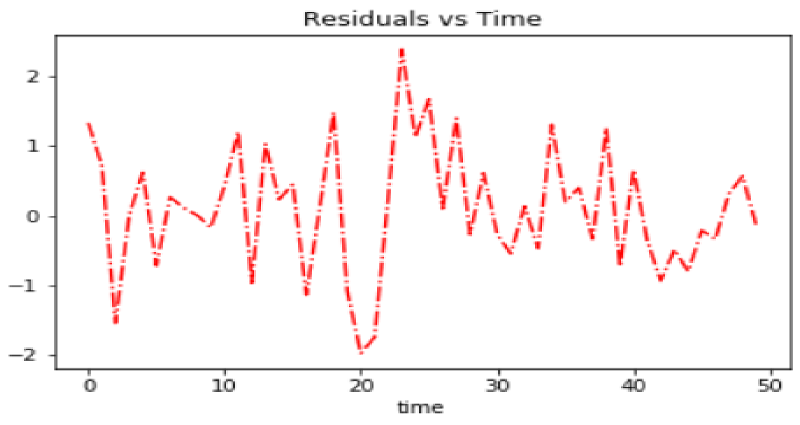


Fig. 5: Visualizing the residuals for general time series data. Adapted from (*Ziganto, 2018*)

Time series decomposition helps understand the time series. However, to make decomposition more accurate for data exploration and improving the data quality for further modeling certain preprocessing needs to be carried out beforehand.

2.3.2 Time Series Data Pre-processing Overview

Data cleaning and pre-processing are crucial steps in the Data Science pipeline, particularly for time series data. Removing trends and seasonal effects is essential to ensure the data is stationary, a property where the mean, variance, and trend correlation of the values remain constant over time (Géron, 2023). Stationary time series data is critical for accurate modeling and dependable forecasting, as will be demonstrated later.

To transform non-stationary data into stationary data and other techniques for preprocessing time-series data are given as follows:

1. Handling Missing Data
2. Handling Outliers
3. Smoothing
4. Differencing
5. Detrending & Deseasonalization
6. Feature Scaling

2.3.3 Handling Missing Data

Many datasets, due to a variety of reasons, have missing values, which has the negative effect of giving degrading model performance and serves as a basis for further misinformed analysis. In time series analysis, there are two categories of missing value imputations, univariate and multivariate methods (Park et al., 2022). The first category deals with filling in missing values in a single column only, using data values, such as traditional values like mean or median for the missing values, when models may not require higher level complexity. Additional methods include linear interpolation and rolling means. Multivariate methods deal with missing values by considering other features of the dataset for performing imputation. These techniques include more complex methods, including Machine Learning Algorithms, such as K-Nearest Neighbor (KNN), Multivariate Imputation by Chained Equation (MICE) and even Multilayer Perceptron (MLP) for imputation. They have been shown to generally provide a more accurate representation of the missing values. Imputation techniques can use the previously observed values, the following observed values, or interpolation to fill data gaps, as shown in Fig. 6 below, where linear interpolation is used.

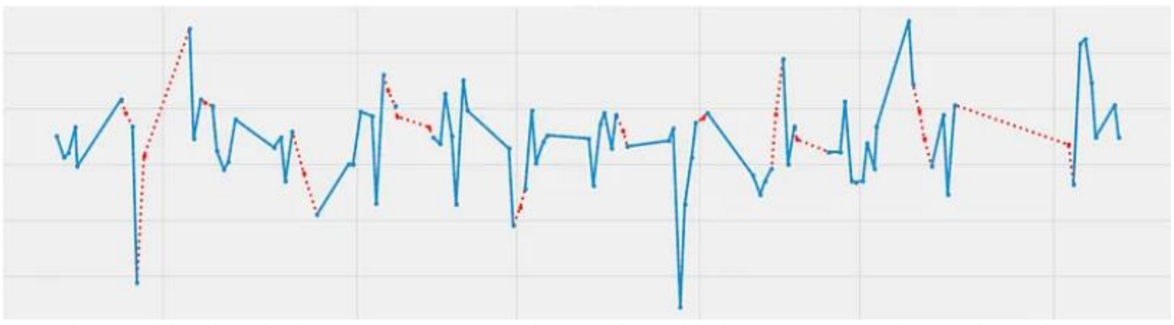


Fig. 6: Filling in Missing Data values for time series with linear interpolation. Adapted From (Ismiguzel, 2022)

2.3.4 Handling Outliers

An outlier is an observation that differs from the overall pattern of any given dataset (Antony et al., 2021). They result from inaccurate data, either through faulty sensor data or human entries. Depending on their severity, they can have a disproportionate effect on model predictions if not correctly handled. Outliers can be identified visually through scatter or box plots or mathematical functions such as Z-Scores or Interquartile Range (IQR) Score. As explained by Antony et al., 2021, any observation with a Z-Score of 3, or a value 1.5 times the IQR, either below the lower quartile or above the upper quartile, are considered outliers and can be removed, or imputed with more representative values. However, it should be noted that this may introduce bias into the dataset. Fig. 7 shows outliers present and highlighted in a scatter plot of Ozone values over time.

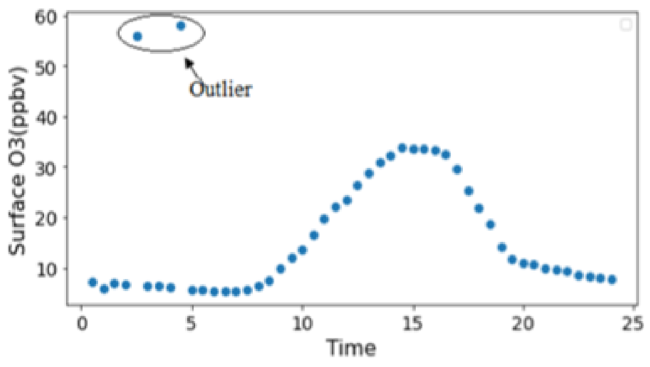


Fig. 7: Outliers in Scatter Plot for Ozone Values over time. Adapted from (Antony et al., 2021)

2.3.5 Smoothing

Smoothing is a technique used in time series analysis to remove random fluctuations and noise in the data. By removing the undesirable noise, we can better understand the underlying short-term trends and behavior of the raw data (Hyndman & Athanasopoulos, 2015). As discussed by Deb et al., 2017, two commonly applied smoothing methods include moving averages and exponential smoothing. Moving averages take the average of a fixed number of past observations to smooth out fluctuations in the data, but they may not capture sudden trends in data as well as exponential smoothing. An Exponential smoothing model assigns weight to previous observations that decay exponentially over time. In other words, the more recent observations have higher weight, which generally produces more accurate forecasts in time series analysis (Hyndman & Athanasopoulos, 2015). However, they may not perform as well if the data has a high level of noise. Fig. 8 shows a graph of COVID-19 deaths in the US state of Florida with 7-days moving average smoothing applied to reveal the underlying trend.

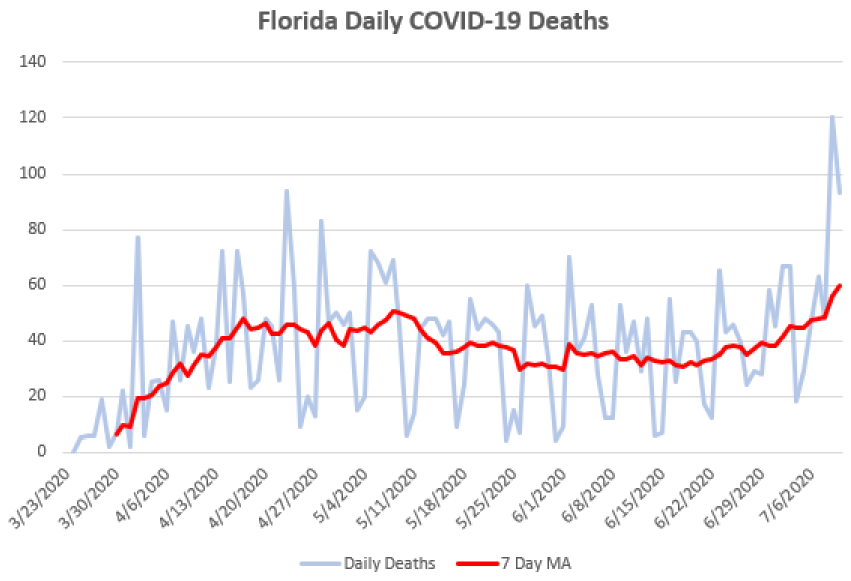


Fig. 8: Moving Average Smoothing applied to COVID-19 daily deaths in Florida, USA. Adapted from (Frost, 2021)

2.3.6 Differencing

Differencing is a method that takes the difference between an observation and the same observation in the previous time step to provide the differenced value $y'\mathrm{t}$, as shown in Equation (1) which is adapted from (Hyndman & Athanasopoulos, 2015):

$$y'_t = y_t - y_{t-1} \tag{1}$$

It is one way to transform non-stationary data into stationary data by reducing the trend and seasonality effects, which can result from autocorrelation, where time series is correlated with its lagged value (Géron, 2023). Autocorrelation can be problematic because it violates the assumption of independence between observations, which is required for many statistical methods to be valid and can lead to inaccurate model results. Occasionally, a second-order differencing with the t-2 timestep being subtracted from the first-order may be necessary if the first-order difference does not make the data completely stationary.

2.3.7 Detrending and Deseasonalization

Removing the trend from time series data is crucial for transforming non-stationary data into stationary data. Some techniques have already been mentioned, such as differencing and smoothing for short-term trends. However, detrending also includes other methods to capture more long-term trends, such as linear regression. This involves fitting the data to a linear line and then subtracting the data from the line; however, this may not be able to best capture non-linear trends which can be present in the data (Hippke et al., 2019). Other detrending methods that have been discussed by Hippke et al., 2019 include Polynomial regression and Lowess Regression. Polynomial regression involves fitting a curve instead of a linear line to the data, then subtracting the data from it. In Lowess Regression the data is split into subset windows and each window has its own regression line fit, which the data will remove for detrending. It should be noted that Lowess Regression has the disadvantage of being more computationally expensive.

Deseasonalization can be considered a subset of detrending, as the seasonal element of a time series is essentially a repeating trend over a fixed interval of time. Many of the previously mentioned techniques can be used with different parameters. For example, moving average smoothing can be used with the window size being the season’s fixed duration. Similarly, differencing can be used, except the previous observation would not be the previous time step, but the previous season timestep instead. Deseasonalization would remove the seasonality component that was not necessarily removed in the last method, which focused on the trend component of the time series.

2.3.8 Feature Scaling

Feature scaling is the next critical step in data preprocessing to ensure that all features are on comparable numerical scales and to avoid model bias towards a specific feature. Different scaling techniques can be used, depending on the data and the problem at hand. Two commonly used scaling techniques are normalization and standardization (Géron, 2023). Normalization scales all numerical values to a range between 0 and 1, while standardization calculates the Z-Score of each value in the feature. While standardization is less affected by outliers, normalization can also work well in the presence of outliers.

The data preprocessing steps and techniques described are all key components of preparing data for further modeling and analysis, as many statistical and machine learning models, including ARIMA and Neural Networks, require data to be stationary to be able to make accurate forecasts (Hyndman & Athanasopoulos, 2015). This research paper will further discuss the different ML and DL models.

2.4 Machine Learning for Time Series

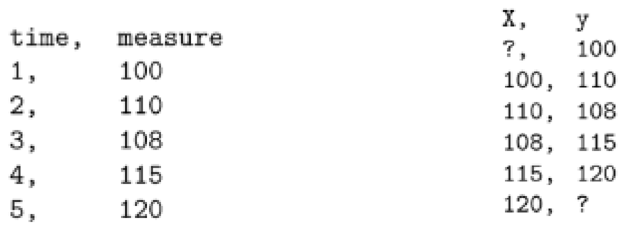
2.4.1 Overview

Time Series models can be considered as a supervised learning problem; however, the temporal aspect must be considered, where each data point would be measured and ordered by a point in time, with there being a dependency structure. Each data point would depend on the previous timestep and the current timestep's input features (**X**) (Brownlee, 2018). This factor makes time series problems particularly challenging, and specific Machine Learning & Deep Learning Models need to be deployed to capture temporal patterns and make more accurate predictions. These include Autoregressive Integrated Moving Average (ARIMA) models and deep learning models like Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks.

2.4.2 Sliding Window Technique

The sliding window technique is a popular method to restructure a time series dataset into a format suitable form for classical supervised learning models such as KNN, decision trees, and linear regression. It maintains the temporal dependency by including the output variable from the previous time step as input variables to the current time step alongside other relevant input variables. This allows the model to predict the output variable for the current time step based on the previous output and the current inputs (Brownlee, 2018).

The key concept here is to define a fixed-length window for a fixed number of time steps and slide it through the time series generating input/output pairs for supervised learning each time (Norwawi, 2021). Fig. 9 shows a simple time series dataset transformed into a supervised learning dataset with the sliding window approach, with the output from each time step serving as the input to the next, barring the first- and last-time steps.



time,	measure
1,	100
2,	110
3,	108
4,	115
5,	120

X,	y
?,	100
100,	110
110,	108
108,	115
115,	120
120,	?

Fig. 9: (Left) Original Time Series Data (Right) Transformed Time Series Data with Sliding Window. Adapted from (Brownlee, 2018)

The width of the window is a crucial factor to consider, as it determines the number of preceding time steps used to predict the current output (Norwawi, 2021). A wider window may better capture more complex temporal-related patterns but may overfit as well. A smaller window may not capture enough information for precise predictions. Ultimately, it must be tweaked to achieve a trade-off between these factors. Equation (2) shows the forecast of output yt with a window width of 2. The previous two-time steps were utilized as input variables to the function.

$$y_t = f(y_{t-1}, y_{t-2}) \tag{2}$$

2.4.3 Classical Time Series Forecasting Models

One of the oldest time series forecasting models is the Autoregressive Moving Average (ARMA) model, developed in the 1930s by Herman Wold (Géron, 2023). It consists of both the Autoregressive (AR) and Moving Average (MA) components. The AR component models the linear relationship between an observation and a number of lagged versions of that observation. The MA component models the linear relationship between an observation and the residual error from a moving average model applied to lagged observations (Brownlee, 2018). ARMA models assume all data provided is stationary and can not deal with non-stationary data unless it is transformed into a stationary form.

Another model that incorporated the autoregressive and moving average components from ARMA to handle non-stationary data better was the Autoregressive Integrated Moving Average (ARIMA model), which incorporated the notion of integration. The integration component adds the differencing method to remove trend components and make the data stationary. The Seasonal ARIMA (SARIMA) was an even further evolution, which includes a seasonality parameter that is not present in the original (Brownlee, 2018).

These models' limitations are that they assume that data is generated by a linear process, which may not be the case in real-world scenarios. They are also limited to Univariate time series forecasting. On the other hand, Vector Autoregression (VAR) and Vector ARIMA models are multivariate models that can handle multiple time series together. Still, they also have their unique strengths and limitations, such as the assumption that there is an intercorrelation between the different time series being modeled (Brownlee, 2018).

Ahmed et al., 2010, in their widely cited paper evaluated eight Machine Learning models for time series forecasting, including KNN Regression, Support Vector Machine Regression (SVM), and various Neural Network configurations. They found that the different models gave significantly differing results, with the Gaussian Process (GP) Regression and Multilayer Perceptron (MLP) models giving the highest performance. Both models can model nonlinear relationships between the variables, making them more applicable in real-world scenarios. The limitation of this study is that it has been applied only to business-type time series and did not explore time series data associated with other phenomena such as in Physics that may exhibit different behavior; hence, it may not necessarily be extended across domains.

2.5 Deep Learning for Time Series

2.5.1 Overview and Evolution of Artificial Neural Networks

Deep Learning is a subfield of Machine Learning characterized by multiple layers of processing units connected in a network. It has shown remarkable success in a variety of applications, including image and speech recognition, and natural language processing, amongst others. These processing units, also known as neurons or nodes, work together using linear regression and activation functions to capture non-linear relationships and build hierarchical representations of data (Dong et al., 2021). Among the different deep learning models, Artificial Neural Networks (ANNs) are the most popular, inspired by the human brain, which consists of billions of neurons connected in layers by neural pathways.

The most straightforward ANN architecture is a Perceptron, which consists of a single feedforward neuron connected to a vector of inputs (x = [xi, xi+2… xn]) through weights (w = [wi, wi+2… wn]), similar to the neural pathways in the brain. The neuron multiplies the input values with corresponding weight values to produce a weighted sum, to which a bias value (b) is added as a constant value to adjust the linear function's intercept and produce an output (**o**) (Géron, 2023). The final step in the feedforward network is applying an activation function to the output, which gives a bounded result (**y**) either in binary form or a continuous value. The function output is commonly between 0 and 1 or -1 and 1 (Banoula, 2023). The corresponding functions for the perceptron are shown below in Equation (3), which shows the weighted sum output, and in Equation (4), the final output after applying an activation function.

$$o = \sum w_i x_i + b \tag{3}$$

$$y = f(o) = f(\sum w_i x_i + b) \tag{4}$$

Fig. 10 shows the graphical representation of the perceptron's architecture and the arrangement of the forward stage in steps from taking in the input vector to producing the final output value. The main limitations of the perceptron arise from the fact that it is only capable of learning linearly separable patterns with binary outcomes and cannot handle non-linearly separable problems such as those XOR or XNOR logic gates, where two lines are needed to separate the classes (Banoula, 2023). This problem was overcome by adding one or more hidden layers between the input and output to form a Multi-Layered perceptron (MLP) capable of processing linear and non-linear patterns.

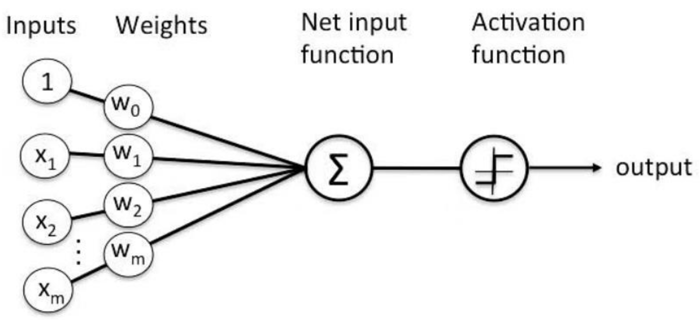


Fig. 10: Perceptron Architecture. Adapted from (Banoula, 2023).

The hidden layers in the MLP receive the previous layer's outputs as their inputs. All current layer neurons are connected to all the previous layer neurons through weights. They transmit their outputs as inputs to the next layer after taking the weighted sum and applying an activation function. The role of an activation function in neural networks is critical; they bind the limits of the output and, most importantly, introduce non-linearity, allowing us to handle the non-linear patterns present in the data (Banoula, 2023). The simplest activation function is the step function producing either a 0 or 1 output. However, it is non-differentiable, which means we cannot train the network with gradient descent. Other popular choices of activation functions are the sigmoid function, hyperbolic tangent, and Rectified Linear Unit (ReLU), which are all differentiable and popularly used in various Neural Networks and Deep Learning architectures (Banoula, 2023).

Like all other Machine Learning Models, Neural Networks must be trained, and they learn through experience. Learning begins from the backward pass that starts after the forward pass, explained above, is completed. The first step is to calculate the error by applying a loss function between the predicted output (Oi) and the actual outputs (Ii) (Dong et al., 2021). Once the error metric has been evaluated, various methods of training the neural network exist. The most popular method employed is Backpropagation, a gradient descent algorithm.

Backpropagation is a widely used gradient descent algorithm, which starts by taking the gradient of the loss function with respect to neuron weights and biases and then propagating it backward through a Neural Network's layers, using the chain rule until it reaches the input layer (Géron, 2023). It efficiently computes the error gradient for all weights and biases and adjusts their values to reduce the error based on the computed gradients. One forward and backward pass combined creates one iteration or an epoch. The number of epochs is repeated, gradually reducing the error margin and outputting values closer to the actual values. The entire process is stopped when the iteration limit is reached, or the error margin is below a defined threshold. Equation (5) represents the update formula for a single weight in the network adapted from (Géron, 2023).

$$w_{i,j}^{(nextstep)} = w_{i,j} + \eta \nabla L(w_{i,j}; x_i, y_j) \tag{5}$$

$w_{i,j}$ represents the weight between the i^{th} input and the j^{th} neuron, η is the learning rate, a hyperparameter, which controls the step size or rate at which the neural network is optimized. $\nabla L(w_{i,j}; x_i, y_j)$ is the gradient of the loss with respect to the $w_{i,j}$ for given input x_i and output y_j.

Tuning the learning rate hyperparameter η is a necessary process, as too low a value can lead to slow convergence to the loss minima. At the same time, too high a value can lead to overshooting the minima and oscillating loss values (Géron, 2023). The gradient descent parameter update process can be further divided into the following types:

1. Batch Gradient Descent
2. Mini-Batch Gradient Descent
3. Stochastic Gradient Descent

Batch gradient descent is when the entire dataset calculates the gradient of the loss function, and the weights and biases are only updated after the gradients are computed. The process can be computationally expensive but can converge to the global minima point of the loss function in case the function's shape is convex, which is not always the case (Patrikar, 2019).

On the other hand, Mini-Batch gradient descent calculates the gradient of the loss function in batches, where the weights and biases are updated for each batch. It is faster than Batch gradient descent but adds the batch size as another hyperparameter that would need to be appropriately tuned for optimal results (Patrikar, 2019).

Lastly, stochastic Gradient Descent (SGD) updates the model's weights and biases after calculating the gradient for each record. It is faster than batch and mini-batch gradient descent, but convergence can be slow due to noise in the dataset. It may never reach the global minimum of the loss function (Patrikar, 2019).

When an ANN contains a deep stack of hidden layers, it is generalized as a Deep Neural Network (DNN). DNNs can contain hundreds of hidden layers or more and have been shown capable of effectively modeling non-linear relationships between inputs and outputs (Géron, 2023). In terms of time series forecasting, DNN architectures such as Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs) and its variant Long Short-Term Memory Networks (LSTM) have shown great promise in providing more accurate and robust models for handling complex and high-dimensional temporal data for both univariate and multivariate time series. They have shown better performance than the classical time series forecasting methods previously mentioned (Brownlee, 2018). The different architectures will be explored further in the coming section.

Fig. 11 shows the evolution of ANNs from the simple perceptron to a shallow neural network (MLP), to DNNs, along with the different activation functions used.

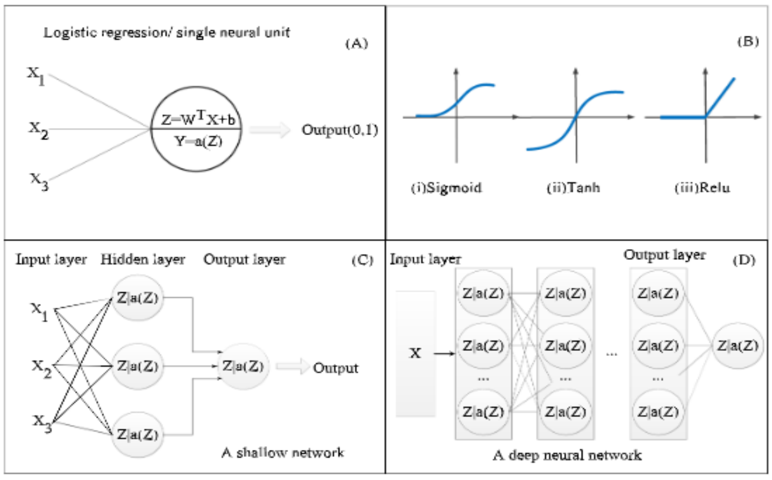


Fig. 11: (A) Perceptron used for classic logistic regression (B) Different Activation function plots (C) Shallow Neural Network with One Hidden Layer (D) Feed Forward Deep Neural Network. Adapted from (Dong et al., 2021).

2.5.2 Deep Neural Networks in Forecasting

Deep Neural Networks for time series forecasting, as previously mentioned, offer the ability to capture complex-nonlinear dependencies and multi-step forecasting for univariate and multivariate predictions that are limited in traditional forecasting methods (Brownlee, 2018). Some of the distinct advantages of DNNs, as discussed by Brownlee, 2018, over most traditional forecasting methods are described in the following paragraphs.

Deep Neural Network models can capture higher-level abstractions that may not be apparent in the raw data and extract the relevant features instead of the traditional time series forecasting that requires domain expertise and manual feature engineering (Brownlee, 2018).

Similarly, DNNs can handle missing and noisy data within their hidden layers, capable of interpolating the missing values and automatically smoothing out the noise. These reduce the need for imputation and smoothing preprocessing techniques beforehand, as are critical in traditional modeling methods, which can also have the unwanted effect of introducing bias (Brownlee, 2018).

DNNs effectively map complex and non-linear relationships between input and output variables and handle high-dimensional data. On the other hand, traditional statistical methods, such as ARIMA, have limitations in handling non-linear data and cannot easily model their relationship, as they assume linear distributions for the underlying data (Brownlee, 2018).

DNNs can also handle many varying input features that can influence forecasts as opposed to traditional time series models, where usually only a single variable is used to predict the future. This has allowed for a more comprehensive real-world analysis and more accurate forecasts for the target variable (Brownlee, 2018).

DNNs can predict multiple time steps into the future, not only one step ahead, as is the norm for traditional time series forecasting models. They can capture long-term temporal dependencies that allow them to perform multi-steps forecasts. Additionally, they can predict multiple time series simultaneously, allowing for multivariate forecasting with multiple target variables, which, combined with taking in numerous input variables, makes it capable of building sequence-to-sequence models (Brownlee, 2018).

The previously mentioned advantages allow DNNs to handle real-world data far more effectively and efficiently than the traditional statistical models, making them more useful in general (Dong et al., 2021). Fig. 12 compares the prediction performance of conventional algorithms and Neural Networks. It is evident that larger Neural Networks can make better predictions with a large amount of data. However, it must be noted that DNN modeling is more computationally intensive and generally less interpretable than more simple time series forecasting models (Brownlee, 2018). Furthermore, it is important to consider that DNNs struggle to model temporal dependencies, especially in time steps that are further apart, while RNNs and the variant LSTMs explicitly model sequential data including time series data.

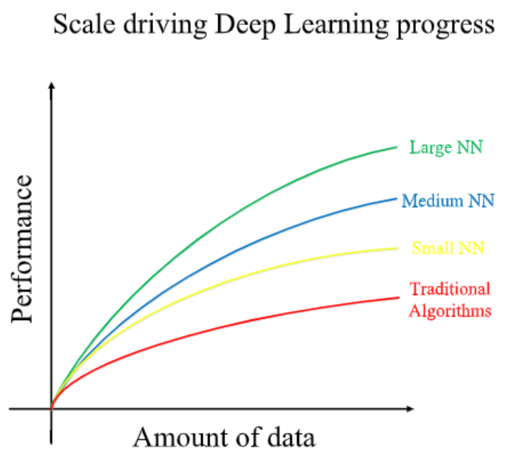


Fig. 12: Prediction Performance of Models based on Data Size. Adapted from (Dong et al., 2021).

2.5.3 Convolutional Neural Networks for Time Series

Convolutional Neural Networks (CNNs) are a type of DNN widely used in computer vision applications such as image classification and video analysis. They have also been successfully used in other tasks, such as Natural Language Processing and even time series forecasting, as long as the data is represented in a grid-like structure (Géron, 2023). CNNs consist of three types of layers, in order they are:

1. Convolutional Layers
2. Pooling Layers
3. Fully Connected Layers

The convolutional layers extract local features from the input data with filters that convolute over the input grid and map the distinct features extracted to feature maps. Compared to feed-forward neural networks, the weights for the filter are shared across the input features, reducing computational cost, with fewer parameters to learn for the final feature map. It should be noted that the filters are also a learned parameter of the model and only the filter size, which depends on how broad or fine the pattern requirements are. The number of filters defines the number of distinct features to be detected. Both of them are hyperparameters for the convolutional layer (Géron, 2023).

The pooling layer is where the pooling operation is carried out by reducing the dimensionality of the feature map to achieve translational invariance, which means that slight variations from the input do not affect the output (Saha, 2022). Different types of pooling include max pooling and average pooling. Max pooling is where the maximum value for a specific subregion represents the entire subregion, with the other values discarded. Average pooling is where the average of a particular subregion is calculated to represent that subregion (Géron, 2023). Overall max pooling has been found to perform better than average pooling, due to being noise suppressant (Saha, 2022).

Both the convolution and pooling layers capture the relevant features and reduce dimensionality to finally feed to the Fully Connected layer after flattening the grid into a vector for the input layer. The fully connected network is the same as a regular feed-forward DNN with neurons arranged in layers and individual weights for each neuron connection across the layers. The final regression or classification output is made from this section (Géron, 2023).

For time series a distinct advantage of using CNNs would be automatic feature engineering where the input and output variables' relationship may not be apparent (Brownlee, 2018). Through the convolution and pooling operations, the CNN model carries out feature compression and extracts the main features (Jin et al., 2019). Fig. 13 displays the feature extraction process for a time series with a CNN model.

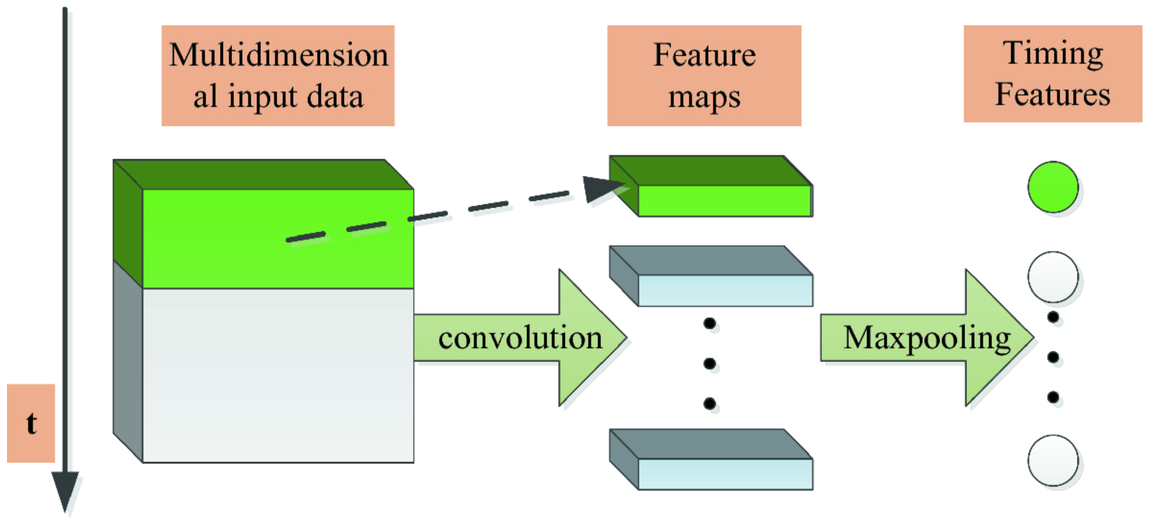


Fig. 13: Timing Features Extraction from a 1D-CNN model. Adapted from (Jin et al., 2019).

Nonetheless, it is important to consider that CNNs struggle to model temporal dependencies, especially in time steps that are further apart, while RNNs and the variant LSTMs explicitly model sequential data including time series data.

2.5.4 Recurrent Neural Networks in Forecasting

Recurrent Neural Networks (RNNs) are a Neural Network architecture capable of handling sequential data. Traditional feedforward neural networks process input data in only one direction. RNNs differ from these as they also have a connection pointing backward from the output, which is fed into the hidden layer neuron at the next time step (frame) (Géron, 2023). The network consists of recurrent layers receiving inputs (***X***) as a scalar value or sequence of vector variables along with the output, sequence or scalar value, from the previous time step (Y_{t-1}). This is often referred to as the hidden state or memory of the network and is updated at every time step. The network can be represented similarly to a sequence of a feedforward network by unrolling it against a time axis of time steps (Géron, 2023). Fig. 14 shows a simple RNN with a single recurrent neuron unrolled over time. The recurrent neuron consists of weight sets for the hidden state (***W***), input sequence (***U***) and output sequence (***V***), along with the bias (Nabi, 2019).

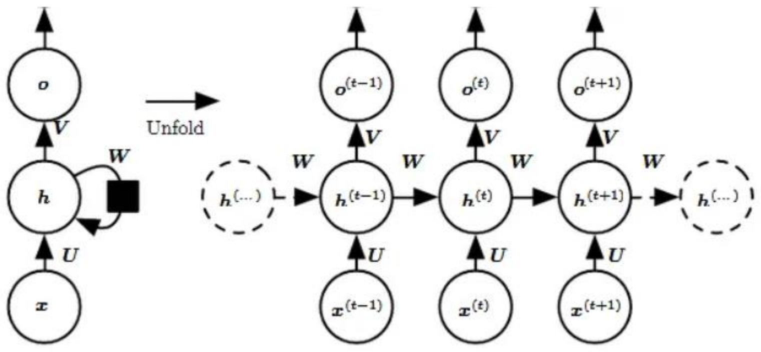


Fig. 14: RNN with a single Recurrent Neuron Unrolled against time axis. Adapted from (Nabi, 2019)

Hidden states in RNNs make them particularly useful for time series forecasting. They can capture temporal dependencies in the data, allowing future predictions based on past values from previous time steps; this sets them apart from CNNs and Feed-forward networks that lack this ability (Brownlee, 2018).

Despite the advantage of capturing temporal dependencies, RNNs are hard to train properly due to having what is called the vanishing and exploding gradient problems. These problems occur while training through backpropagation through time from the last time-step neuron layer to the first recurrent neuron layer. The gradient's product either approaches zero or increases exponentially, making it challenging for the network to capture long-term dependencies from faraway time steps (Nabi, 2019. To handle this problem, a new variant of the traditional RNN called the Long-Short Term Memory Network (LSTM) was introduced, which can better handle these long-term dependencies.

2.5.5 Long-Short Term Memory Networks in Forecasting

The Long Short-Term Memory (LSTM) Network is a variant of the RNN architecture that was proposed in 1997 to overcome the vanishing and exploding gradient problems and improve the ability to capture long-term dependencies (Géron, 2023). The LSTM introduces a new memory cell and three gates responsible for controlling the flow of information into and out of the cell. The long-term memory captured over the previous time steps is stored in the memory cell. The gates that control the flow of information in and out of the cell are:

1. The Forget Gate
2. The Input Gate
3. The Output Gate

In addition to the hidden state short-term memory ($\boldsymbol{h_{t-1}}$) from the previous timestep, it also contains a long-term cell state memory ($\boldsymbol{c_{t-1}}$). These and the input at the current time step ($\boldsymbol{X_t}$) form the cell inputs. The first section of the cell is the forget gate which takes $\boldsymbol{h_{t-1}}$ and $\boldsymbol{X_t}$ as its input and then applies a sigmoid activation function and element-wise multiplication operation. The output is used to scale $\boldsymbol{c_t}$ determining how much of the cell state memory should be forgotten as it is no longer relevant based on the new inputs (Géron, 2023).

The second section of the cell consists of a main layer, which takes $\boldsymbol{h_{t-1}}$ and $\boldsymbol{X_t}$ as its input and then applies a hyperbolic tangent activation function to normalize the value and produce an output $\boldsymbol{g_t}$. The input gate acts similarly to the forget gate, but instead it determines how much of $\boldsymbol{g_t}$. is added to $\boldsymbol{c_t}$ as part of the long-term state instead of forgotten (Géron, 2023).

The last section of the cell consists of the output gate, which determines how much of $\boldsymbol{c_t}$ should be output from the cell in terms of the output $\boldsymbol{Y_t}$ and the current hidden state $\boldsymbol{h_t}$ after passing it through a hyperbolic tangent function first. The output gate function acts in the same fashion as the input and forget gates (Géron, 2023).

The input and forget gates update long-term memory cell state ($\boldsymbol{c_t}$), while the output gate calculates the output $\boldsymbol{Y_t}$ and hidden state value for the next time step $\boldsymbol{h_{t.}}$ LSTMs, like all other ANNs, also have weights and biases which are similarly trained through Backpropagation. Fig. 15 shows the LSTM cell and its constituent components and operations.

In 2014 Kyunghyun Cho et al. introduced a new version of LSTMs with a simpler architecture called Gated Recurrent Units (GRUs) (Géron, 2023). GRUs similar to traditional RNNs only take in one hidden state ($\boldsymbol{h_{t-1}}$) as an input along with the current time step input $\boldsymbol{X_t}$, however, they maintain LSTMs advantages of capturing long-term dependencies (Géron, 2023). The GRU cell consists of two gates, as opposed to three in the LSTM cell; these include the update and reset gates. The update gate determines how much of $\boldsymbol{h_{t-1}}$ should be retained and how much of the $\boldsymbol{X_t}$ should be added to the $\boldsymbol{h_{t.}}$ The reset gate determines how much of the previous state $\boldsymbol{h_{t-1}}$ should be forgotten (Géron, 2023). GRUs have a simpler architecture with fewer parameters and can be trained more efficiently than LSTMs.

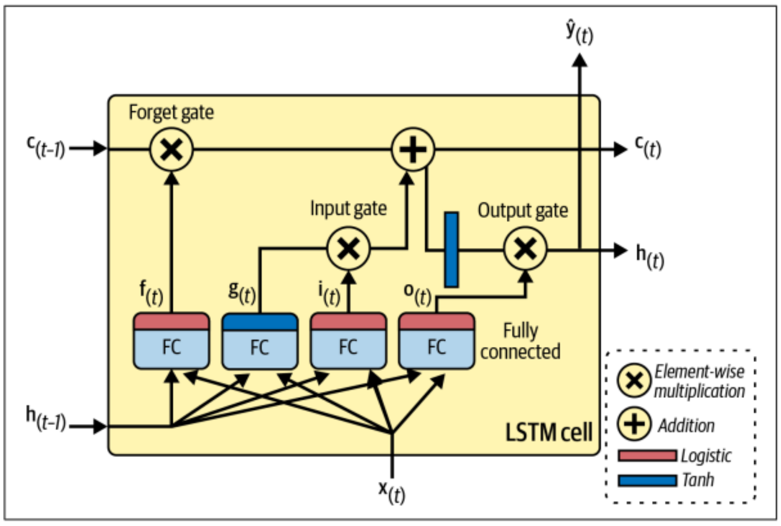


Fig. 15: LSTM Cell with constituent components and operations. Adapted from (Géron, 2023).

2.6 Evaluation Methods

Evaluating any model is critical to conclude its effectiveness at capturing the real underlying relationship between the inputs and the outputs by comparing the difference between the actual and predicted values, also known as the error. For time series models, which are mainly regression problems, several evaluation metrics exist for error measurement; they are further described as follows (Bergmeir & Benítez, 2012):

1. Scale-Dependent measures: These consist of absolute errors and square errors averaged by the arithmetic mean or median of the data. The following equations (6) to (9) fall into this category for a single time step *t*.

$$Mean\ Absolute\ Error(MAE) = \frac{1}{n}\sum_{i=1}^{n} |y_t - \widehat{y_t}| \tag{6}$$

$$Mean\ Square\ Error(MSE) = \frac{1}{n}\sum_{i=1}^{n} (y_t - \widehat{y_t})^2 \tag{7}$$

$$Root\ Mean\ Square\ Error(RMSE) = \sqrt{MSE} \tag{8}$$

$$Median\ Absolute\ Error(MDAE) = Median(|y_t - \widehat{y_t}|) \tag{9}$$

Scale-dependent measures cannot be used to compare and average errors across heterogeneous time series (Bergmeir & Benítez, 2012). From these measures, the square error measures have a higher sensitivity to outliers.

2. Percentage error measures: These overcome scale dependency by dividing it by the reference value, which describes the percentage error (PE_t). The percentage error is given in equation (10) and the metrics that fall into this category are given from equations (11) to (14).

$$PE_t = 100\ \frac{y_t - \widehat{y_t}}{y_t} \tag{10}$$

$$Mean\ Absolute\ Percentage\ Error(MAPE) = Mean(|PE_t|) \tag{11}$$

$$Median\ Absolute\ Percetange\ Error(MDAPE) = Median(|PE_t|) \tag{12}$$

$$Root\ Mean\ Square\ Percentage\ Error\ (RMSPE) = \sqrt{Mean(PE_t)^2} \tag{13}$$

$$Root\ Median\ Square\ Percentage\ Error\ (RMDSE) = \sqrt{Median(PE_t)^2} \tag{14}$$

The main issue with this metric is that if y_t is zero then they have to deal with infinite values. They also give very high percentage errors at small values, as the denominator gets small, so the measures show a skewed distribution (Bergmeir & Benítez, 2012). To attempt to tackle these issues, a symmetric measure such as the Symmetric Mean Absolute Percentage Error (SMAPE) is used as shown in equation (15).

$$Symmetric\ Mean\ Absolute\ Percentage\ Error\ (SMAPE) = \frac{1}{n}\sum_{i=1}^{n} 100\frac{|y_t-\hat{y}_t|}{m_t} \tag{15}$$

$$with\ m_t = \frac{|y_t| - |\widehat{y_t}|}{2}$$

3. Relative Errors: Another approach to not scale with a reference value is with the help of a benchmark method *B*. Equation (16) shows the relative error formula.

$$Relative\ Error\ (RE_t) = \frac{y_t-\hat{y}_t}{y_t-\widehat{y_{tB}}} \tag{16}$$

With this measure, however, the general problems of the percentage error measures persist as the denominator may still evaluate to zero (Bergmeir & Benítez, 2012).

All commonly used error metrics have shortcomings and no widely accepted metric that is robust, scale-independent, and interpretable (Bergmeir & Benítez, 2012). One proposed method by Hyndman and Koehler to deal with the zeros in the input values is the Scaled Error (SE_t). It proposes to scale errors using the in-sample error of the benchmark time series naïve method *i* (Bergmeir & Benítez, 2012). as shown in equation (17).

$$SE_t = \frac{y_t-\hat{y}_t}{\frac{1}{n-1}\sum_{i=2}^{n}|y_i-y_{i-1}|} \tag{17}$$

From this scaled error, its Mean Absolute Scaled Error (MASE) and Median Scaled Error (MDSE) can be computed, which is robust against zeros in the input data. However, its shortcomings are that it relies on a benchmark method and can be hard to interpret.

A widely used technique for evaluating model performance on unseen data is k-fold cross-validation. The technique splits the dataset into *k* non-overlapping folds, with k-1 folds used for model training and one-fold used for testing. The process is repeated *k* times with the test set changing each evaluation to cover each fold (Géron, 2023). However, standard cross-validation is inappropriate for time series data as it does not preserve the temporal order of the data as a model can be trained on future record features to predict past value outputs, which leads to inaccurate performance estimates.

For time series cross-validation, one common form is the rolling-origin-update, where the testing window is fixed in size, and the training window expands with each iteration. It is also called n-step-ahead evaluation, as it predicts n steps ahead in the time series. By using a fixed testing window and expanding the training window, the technique preserves the temporal order of the data and provides more realistic estimates of the model's performance. (Bergmeir & Benítez, 2012).

2.7 Related Work

Several recent research investigations and journal papers have been published for data-driven train delay predictions. The most recent research papers have focused on deep learning techniques due to their ability to capture complex non-linear relationships, automatic feature engineering and long-term dependencies.

Wu et al., 2021 proposed a tactical-level hybrid LSTM and Critical Point Search (CPS) hybrid model for predicting train delays one day ahead for a specific daily scheduled trip across a vector of all the stations. The LSTM component predicted train dwell time and running time as a multivariate time series forecasting problem. The CPS component is a rule-based classification algorithm that classifies the predicted delays as primary, secondary, or on-time running for no delay. Primary delays are delays caused by external factors for a specific train while secondary delays result from a knock-on effect delay from the primary delayed train for the other trains.

Their results showed that the standard LSTM model had the best MAE and MAPE at 14.973 and 11.779 compared to the benchmark Random Forest model and different LSTM variants for running time predictions. The GRU-LSTM had the best MAE at 9.844 and MAPE at 14.480 for predicting dwell times.

Wen et al., 2019 used an LSTM operational-level model to predict arrival delays at a station based on the feature space of the train at the current sequence and the feature space of the previous train in the sequence for a specific station, as shown in Fig 16. In addition to current train operational data, the feature space also considered the delay at the previous station as a spatial consideration and the effect of the peak usage hours. They compared the LSTM model with ANN and Random Forest models across seven stations and concluded LSTM outperformed both. They verified the generalization and robustness of their model by testing on different railway sections that included stations not included in the original seven. They found that the LSTM model outperformed both benchmark models again, demonstrating the model's ability to generalize to new data beyond the initial training set.

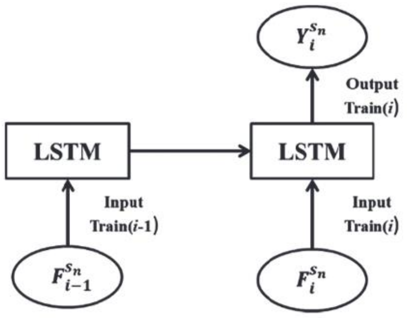


Fig. 16: Predicting Train Arrival Delay at a Station using current time step and previous time step features as proposed by (Wen et al., 2019)

Tatsui et al. 2021 utilized an LSTM network to predict train departure delays for a particular train up to a user-defined number of stations ahead. They explored various models with different input features that included departure delays of the current train and preceding trains, arrival delays of the current train, and congestion of the current train. To predict future station delays, they considered the data from the previous five stations for each feature. The study found that the model with the arrival and departure delays of the stations as input outperformed traditional ANN models and those with other input features in predicting train delays.

Huang et al., 2020 in their study developed a hybrid FCLL-Net model that combined a single Fully Connected Neural Network (FCNN) with two LSTM networks to predict the delay for a given train at the next station or the next *n* stations. The FCNN component processed non-operational static infrastructure and weather-related features. The first LSTM considered a sequence of trains at a given station and the second LSTM considered a sequence of stations for a given train. A final FCNN merged the outputs of the constituent models to produce the output delay at a downstream station. The model was benchmarked against other models such as Random Forest, MLP, and Deep Extreme Learning Machine (DELM) and demonstrated an average improvement of 17.7% for one railway line section. Fig 17 shows the framework of the FCLL-Net model.

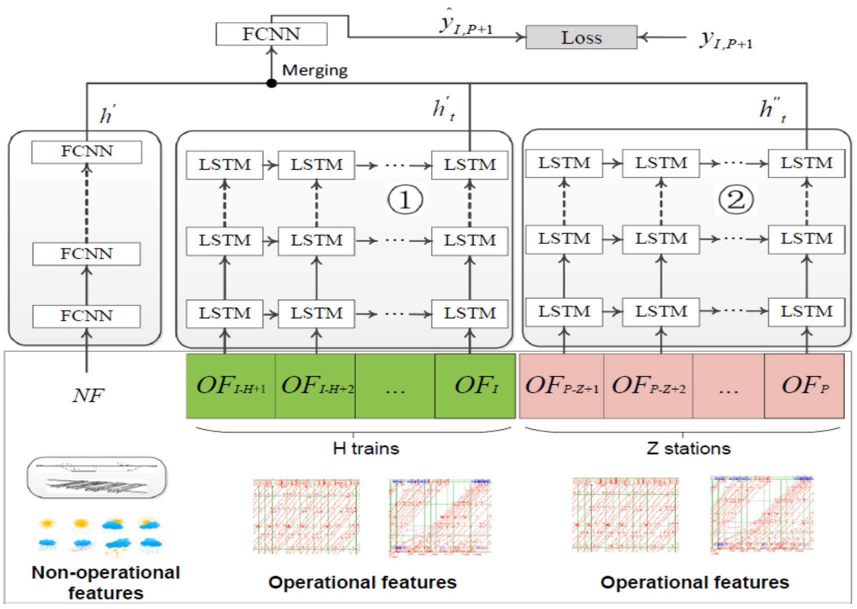
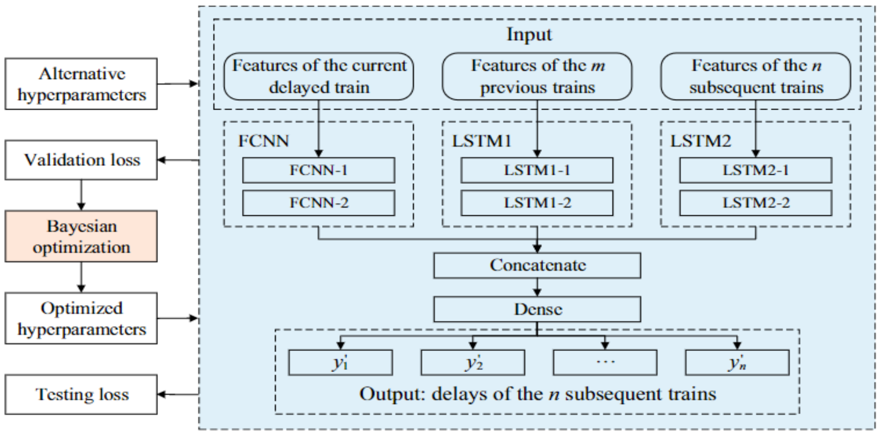


Fig. 17: FCLL-Net Model Framework. Proposed by (Huang et al., 2020)

Luo et al., 2022 proposed a multi-output hybrid model to predict a multi-step vector of train arrival delays at a specific station. The model included two LSTM components and an FCNN component. It was designed to predict delays for an *n* number of time steps ahead based on current train features and the last *m* number of time step train features. The FCNN component took features for the current train, the first LSTM took features of the *m* previous trains, and the last LSTM took features of the *n* subsequent trains. The final output vector was predicted after concatenating and combining the outputs from the models. The different hyperparameters of the hybrid model were optimized with the help of a Bayesian Optimization algorithm. Fig. 18 shows the hybrid model's framework.

The hybrid model was compared to another combinatorial benchmark model consisting of a shallow LSTM-FCNN that produced a multi-output vector. The model outperformed the benchmark in 60% of the stations for RMSE and MAE evaluations.


Input
Features of the current delayed train
Features of the m previous trains
Features of the n subsequent trains
Alternative hyperparameters
Validation loss
Bayesian optimization
Optimized hyperparameters
Testing loss
FCNN
FCNN-1
FCNN-2
LSTM1
LSTM1-1
LSTM1-2
LSTM2
LSTM2-1
LSTM2-2
Concatenate
Dense
y'_1
y'_2
...
y'_n
Output: delays of the n subsequent trains
Fig. 18: Train Delay Prediction DL Hybrid Model Framework proposed by (Luo et al., 2022)

2.8 Summary and Critical Analysis

Wu et al., 2021 were the first to apply deep learning approaches to predict long-term train delays for both primary and secondary delays. However, their features were limited to train operational data and did not consider external influencing factors, such as weather and peak hours' impact. Since their model data was limited to historical values of a single trip, it could not model such dynamism. Their implementation was done at a tactical level to predict train dwell times and running times one day in advance, while the prediction horizon of our project would be at an operational level to predict secondary time delays at specific stations.

Wen et al., 2019 tested the generalization and robustness of their model by testing it on different railway sections, but its evaluation could have been improved by considering additional prediction outputs, such as the departure delay at the current station. Their study considered peak travel hours but did not consider weather conditions. Moreover, their model only made single-step predictions, whereas a more dynamic approach could forecast a vector of multiple future time steps. Furthermore, their study did not distinguish between primary and secondary train delays. Our study aims to investigate secondary delays as prediction outputs.

Tatsui et al., 2021 considered data from a number of previous stations; however, they did not consider many features such as running times, the effect of peak hours and weather as contributing factors to delay. Additionally, they considered the time step sequences between stations as uniform, which is rarely the case realistically, as they change depending on the distance, which was not accounted for. Our study will target delays at individual stations rather than the delay of one train over multiple stations.

In their study, Huang et al. 2020 integrated non-operational features and utilized LSTM networks to capture delay propagation across sequences of stations and trains, with each LSTM component modeling the corresponding distribution. However, their model predicted delays for a single train at a downstream station and did not investigate delay propagation across a sequence of trains for a specific station. Our study aims to predict delay propagation at a single station across a sequence of trains rather than a sequence of downstream stations.

Luo et al., 2022 took into account the effect of multiple previous time-step train features to predict a vector of delays multiple time steps ahead, with each train arrival at the station considered a time step. However, they restricted their study to train operational parameters and did not consider external factors like weather or peak hours.

2.9 Recommendations for Further Work

The reviewed literature shows that most studies that predict train delays have focused on predicting delays for a single train at downstream stations. In contrast, research that delves into delays for a sequence of trains at individual stations is limited. This aspect, involving the analysis of time series models and the consideration of secondary delays that propagate across trains due to primary delays, remains relatively unexplored. Furthermore, most studies are based in Europe or China and have focused on High-Speed Railways with stations spread across cities, which have unique characteristics such as train overtaking. Light rail systems like Dubai Metro have not received much attention in this area of research as there are not as many studies.

To bridge the research gap, the study proposes the development of a model that forecasts train delays for a specific station in Dubai's Metro Line Network. The model will account for delay propagation in the line by incorporating both delays at preceding stations and those encountered by a certain number of preceding trains at the station. This approach is intended to forecast subsequent secondary train delays at the targeted station, consequent to an initial delay surpassing a specified temporal threshold.

3. Methodology and Experiment Setup

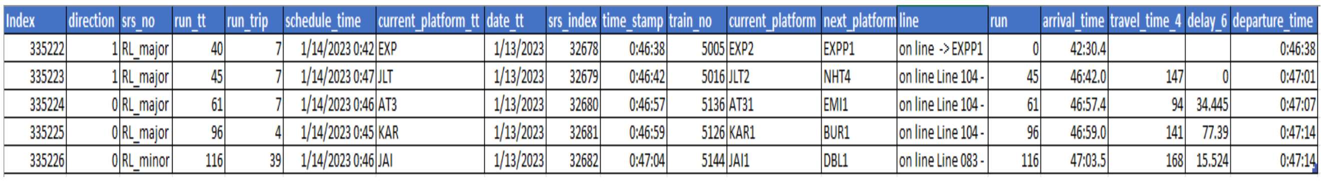
3.1 Dataset Overview

The dataset received from the Dubai Metro Operations Team contained train operational records from January 2, 2023, to January 13, 2023, and included records from all operating lines and platforms. It has a total of 335,227 records and comprises 18 total features. The dataset's features were extracted by the team from the Metro Operations Schedule and real-time Trace System, resulting in the inclusion of some similar and duplicate entries. Further data relating to weather conditions were not considered as the climate of Dubai is stable through almost all of the year with very few rainy days on average.

The train operational dataset included the following key features:

- Line: The line on which the train operates, categorized as Green Line, Red Major Line, or Red Minor Line.
- Direction: A binary variable indicating the train's direction on the line.
- Scheduled Departure Time: The planned departure time for each train.
- Actual Departure Time: The actual departure time of each train.
- Actual Arrival Time: The actual arrival time of each train.
- Date: The date of the recorded train instance.
- Platform Name: This included different columns for the station identifier and the unique platform identifier, irrespective of the direction.
- Next Platform Name: The next platform in the trip sequence.
- Delay: This is the variation between the scheduled and actual departure time; however, this is rule-based and does not always reflect the exact time variation between the timestamps. A more accurate calculation was performed later on. This feature will serve as the target variable.
- Scheduled Travel Time: The scheduled travel time for a train between stations.

Additionally, the dataset included other features such as the trip numbers, and unique train identifiers, among others. Fig. 19 displays the last five rows from the provided dataset.



Index	direction	srs_no	run_tt	run_trip	schedule_time	current_platform_tt	date_tt	srs_index	time_stamp	train_no	current_platform	next_platform	line	run	arrival_time	travel_time_4	delay_6	departure_time
335222	1	RL_major	40	7	1/14/2023 0:42	EXP	1/13/2023	32678	0:46:38	5005	EXP2	EXPP1	on line -> EXPP1	0	42:30.4			0:46:38
335223	1	RL_major	45	7	1/14/2023 0:47	JLT	1/13/2023	32679	0:46:42	5016	JLT2	NHT4	on line Line 104 -	45	46:42.0	147	0	0:47:01
335224	0	RL_major	61	7	1/14/2023 0:46	AT3	1/13/2023	32680	0:46:57	5136	AT31	EMI1	on line Line 104 -	61	46:57.4	94	34.445	0:47:07
335225	0	RL_major	96	4	1/14/2023 0:45	KAR	1/13/2023	32681	0:46:59	5126	KAR1	BUR1	on line Line 104 -	96	46:59.0	141	77.39	0:47:14
335226	0	RL_minor	116	39	1/14/2023 0:46	JAI	1/13/2023	32682	0:47:04	5144	JAI1	DBL1	on line Line 083 -	116	47:03.5	168	15.524	0:47:14

Fig. 19: Sample of Train Dataset

3.2 Data Pre-processing

3.2.1 Overview of Process

For this project, all data programming, pre-processing, and exploration work will be performed using Python and its libraries through Jupyter Notebook, a web-based interactive computing platform. Jupyter Notebook allows a combination of code, visualizations and building an explanatory text narrative, making it an ideal tool for the project.

The pandas and numpy libraries will be utilized for data exploration, preprocessing and preparation of the data for further modeling. These libraries offer excellent data manipulation and analysis capabilities. In addition, the matplotlib and seaborn libraries will be used for informative visualizations to gain further insight.

The pre-processing stage involves several key steps:

1. Exploring and Understanding the Data: In-depth exploration of the dataset will allow an understanding of the feature significance and an analysis of their data types and distributions providing insight before taking further decisions.

2. Checking for Duplicate and Null values: Handling null and duplicate values is critical to ensure data quality during analysis and allows for more efficient and effective model performance further on.

3. Feature Engineering and Feature Selection: Necessary to improve model performance and efficiency.

4. Data Preparation for Modeling: Before feeding the data into a model it is crucial to split it into test and train sections, scale it to reduce bias of certain features and ensure it fits the relevant format as per the use case and for the model algorithm.

Following this comprehensive pre-processing pipeline, the aim is to prepare the data optimally as per the use case and ensure effective and efficient modeling and analysis subsequently, which is a key objective of the project.

3.2.2 Initial Data Cleaning

After loading the dataset and gaining an understanding of the data and its variables, the following steps were taken for initial data cleaning:

1. Handling Null and Duplicate Values: The dataset was checked for null and duplicate values. No duplicate values were found. However, around 0.229% of records had 11 values out of 18 features missing. These missing values included critical information like actual arrival, departure times, and real-time timestamps, essential for determining train delays. As these rows lacked crucial information and couldn't be imputed, they were dropped.

2. Null Value Handling for Arrival, Departure and Travel Times: After removing the initial null value records, only the arrival time, departure time, travel time, and delay features had remaining null values. Approximately 0.002% and 0.003% of the arrival and departure time values were null, respectively. An analysis revealed that 98.56% of the arrival time values were identical to the timestamp values, so the missing arrival time values were imputed using the timestamps column, which was subsequently dropped. The records with missing departure time values were dropped due to their insignificantly small number and unavailability for imputation. The missing travel time values, which accounted for only 0.5% were also dropped as they could not be reliably imputed.

3. Handling Delay Values: After further cleaning, the only feature with missing values remaining was the delay column. As previously mentioned, the provided delay column was rule-based and did not always provide the exact time variation between the scheduled and actual departure delay. To rectify this, a new delay feature was calculated by subtracting the actual and scheduled departure times, both of which were now available for all records. The previous delay column was dropped, leaving the cleaned dataset with no more null values and it was now ready for further analysis.

3.2.3 Feature Engineering and Selection

In addition to the existing features, ancillary train operational features that have been used in previous studies and have proven relevant to forecasting train delays can be derived and their predictive power analyzed.

The following additional features were found in the literature to be relevant when developing train delay models:

1. Dwell Time: This represents the time a train remains stationary at a platform or station during passenger boarding, it is the difference in time between the actual arrival time and actual departure time. Since the planned arrival time is not available in the dataset the planned dwell time cannot be calculated and only the actual dwell time will be found. Multiple studies have used the dwell time in their previous studies (Huang et al. 2020; Luo et al., 2022; Wu et al., 2021).

2. Time Intervals: This usually represents the interval time between trains sequentially arriving at a platform but can also be extended to trains departing from a platform. The actual and planned departure time interval as well as the actual arrival interval will be found from the dataset. The arrival time interval has been used as a feature in previous studies (Huang et al. 2020; Luo et al., 2022).

Additionally, the propagation of delays from previous platforms is another important spatial factor to consider when trying to predict the delay at a platform. To extract and include the delays from previous platforms as new features for a record the data frame will need to be re-sorted.

The previous platform delays are extracted with the following steps:

1. Sorting Data Frame: The data frame is sorted in a hierarchical order by line of operation, the train direction, and the unique train ID, which remains constant during each trip. The existing timestamp sequence is maintained ensuring the sequential train records for each trip are arranged together.

2. Creating Mask: To correctly include delays from the previous sequential platform for a given trip, specific conditions are applied:
   - The date value of the previous record must match the date value of the current record for which the new platform delay features will be added.
   - The train number of the previous record must match the current record to ensure the platform delay value is from the same trip.
   - The current train platform is the same as the next platform value in the previous record to ensure that the platform delay value is from the previous sequential platform and not another platform in case the previous platform's record is missing.

3. Adding Previous Platform Delays: Based on the mask conditions, delays from the previous platform are shifted into the current record as new features, representing the previous platform delays. If the mask conditions are not met, the new feature value is set as zero since it can be assumed that the delay value from the previous platform is either missing or there is no previous platform in the sequence. In the study when considering delays from multiple previous platforms similar masks are created to ensure that the previous platform sequence is maintained. In this project, the delay

values of three previous platforms are added as new features for each record for further analysis.

With the same data frame arrangement and mask conditions, the actual travel time between platforms can also be found as opposed to the existing rule-based value by subtracting the arrival time at a platform from the departure time at the previous platforms. For records where this condition is not satisfied, the value can remain the same as the rule-based travel time value of the record. Upon creation of this more accurate travel time feature the previous travel time column is dropped and is no longer considered for further analysis.

The following features were created in this step:

- Delay One Platform Back
- Delay Two Platforms Back
- Delay Three Platforms Back
- Actual Travel Time Between Platforms

Furthermore, as the aim of the project is to investigate the impact of secondary delays and propagation of delays over time across trains, assessing the temporal impact of lagged delay values as features on future delays is critical to evaluate.

The future delay values for a platform are extracted with the following steps:

1. Sorting Data Frame: The new sorting will be done hierarchically by platform and scheduled departure time as the time variable in the time series. This arrangement ensures that records are organized in a sequence of trains at a specific platform.

2. Creating Mask: Similar to creating features for delays from previous platforms specific conditions are applied to add leading delay features in a record:
    - The date value of the current record must match the next record.
    - The train platform from the current record matches the next record.

3. Adding Future Delays: Considering both conditions is important as sequential records for a single platform on a single date form the time steps for each time series. Therefore, only future delay values from the same time series should be added as a record’s feature.

Delays from the next 10 sequential trains at a platform were added as features for each instance for further analysis.

Using the same data frame arrangement, the scheduled and actual departure time intervals, as well as actual arrival time intervals between trains at the platform features can be calculated. It should be noted that some train records in the sequence might be missing, which introduces a limitation. In this case, the leading delays and time intervals are calculated from the last known timestep of the day. In case there is no preceding train due to it being the first train at the platform for the date the value for the time intervals shall be zero.

The following features were created in this step:

- Scheduled Departure Interval Time between Trains at a Platform
- Actual Departure Interval Time between Trains at a Platform
- Actual Arrival Interval Time between Trains at a Platform

Investigating the summary statistics of the created features revealed that around 22 records had their actual departure time intervals between trains below zero, implying that the previously scheduled train in the sequence departed after the next one, which does not make logical sense. Hence, these records were considered outliers and were dropped from the data frame.

After completing feature engineering, the next step involves analyzing the correlation and predictive power of the features on the train delay and future time step delays. To accomplish this, we employed the Pearson Correlation Coefficient, which examines the linear correlation between two continuous variables. The obtained correlation results between all continuous train operational features were then effectively visualized using a heatmap, as shown in Fig. 20. This visualization provides valuable insights into the relationships between the features and delays, shedding light on potential patterns and trends in the data.

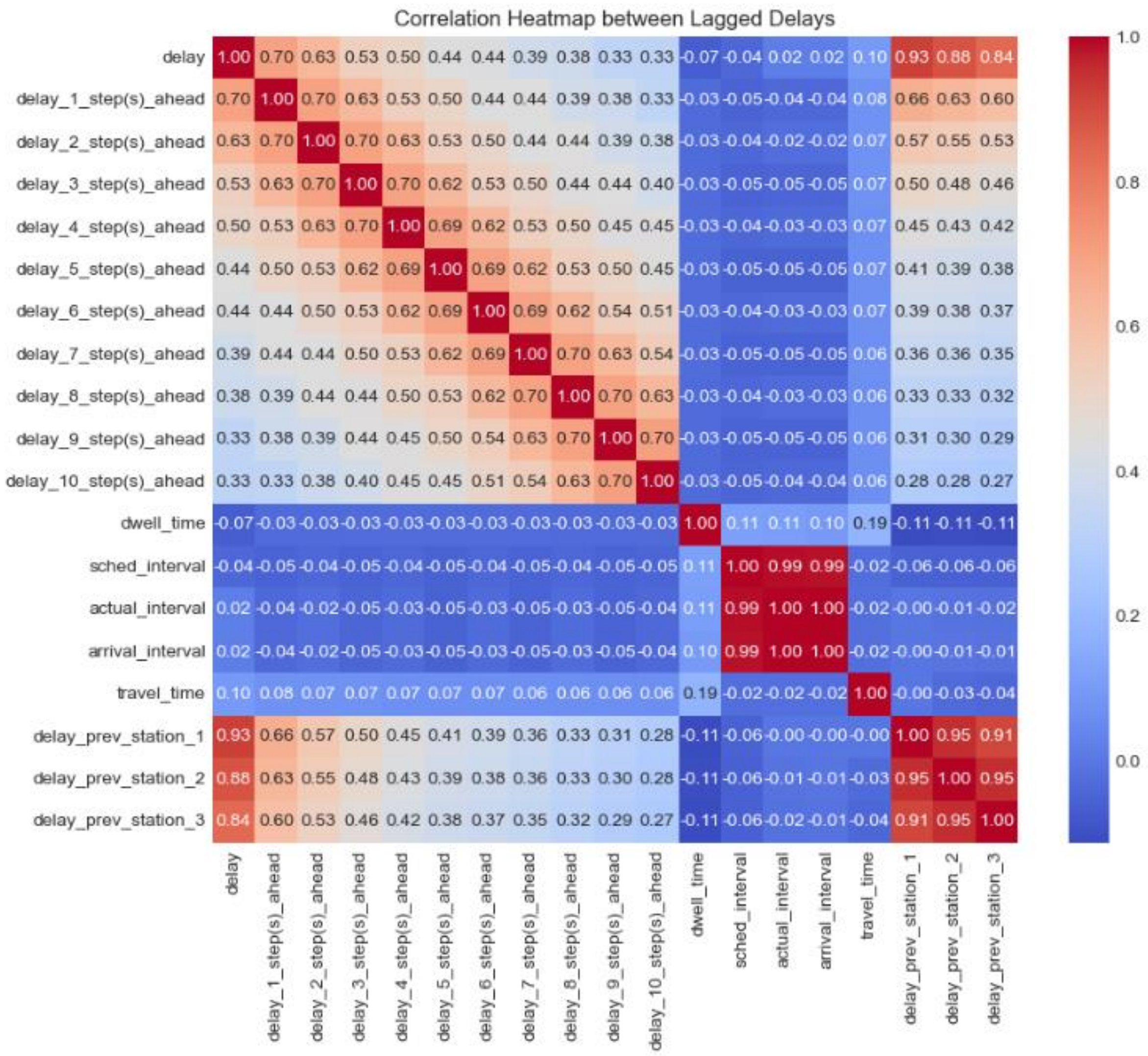


Fig. 20: Correlation Analysis Heatmap

Based on the correlation heatmap, it is evident that the delay value is moderately correlated with future time step (*i*) values, with a coefficient of 0.7 with the delay at the next time step. This correlation progressively decreases with further future time-step delays. Additionally, when considering the spatial dimension (*j*), the delay values across stations exhibit an even stronger correlation with delays from previous stations station with the correlation delay from one platform back being 0.93 with the delay for the same trip, or time step.

On the other hand, train operational features like dwell time, actual and scheduled intervals between train arrivals at a platform, and travel time between stations show minimal correlation with the current or future delay values.

Going back to the existing features, as a result of organizing the data into sequential trains at a platform within the same day, a single time series is created. Within this context, some features are constant for the time series and act as zero-variance predictors. Consequently, they are irrelevant for further modeling and analysis. These attributes encompass the following:

- Current and Next Platforms
- Date
- Direction
- Train Line

Finally, non-operational features such as the trip and train IDs, which are categorical variables, are also dropped as these have not been shown to contribute towards capturing delay patterns.

With this, it becomes evident that spatial delays from previous stations and delays from previous time steps exhibit suitable predictive power for capturing the future delay of a train at a platform. However, the other features with weak correlations will be incorporated into the models to assess if other non-linear or temporal dependencies exist that the correlation analysis might not have been able to capture.

3.2.4 Exploratory Data Analysis

The previous steps covered feature engineering through the creation of new operational features. Before further pre-processing, the delay variable, which is the target, was explored and visualized for evidence of trends or seasonality. This aimed to determine the stationarity of the delay variable and, consequently, the previous train ($d_{i-n,\ j}$) and previous platform ($d_{i,\ j-n}$) delays, which are derived from the delay variable. Here, "*i*" signifies distinct time steps, "*j*" signifies different platforms, while "*n*" is the number of timesteps or platforms displaced from the current state. The same notation will be used when referring to spatial and temporal dimensions going forward. Fig. 21 illustrates the time series line charts highlighting the top 5 platforms with the highest average delays. From there, it can be seen that there seems to be no visually discernible trend or seasonality in departure delay for the delay at any of the platforms, while the patterns of delays are similar for the platforms belonging to the same line of operation confirming the impact and propagation of delays across platforms found in the correlation analysis from the preceding section.

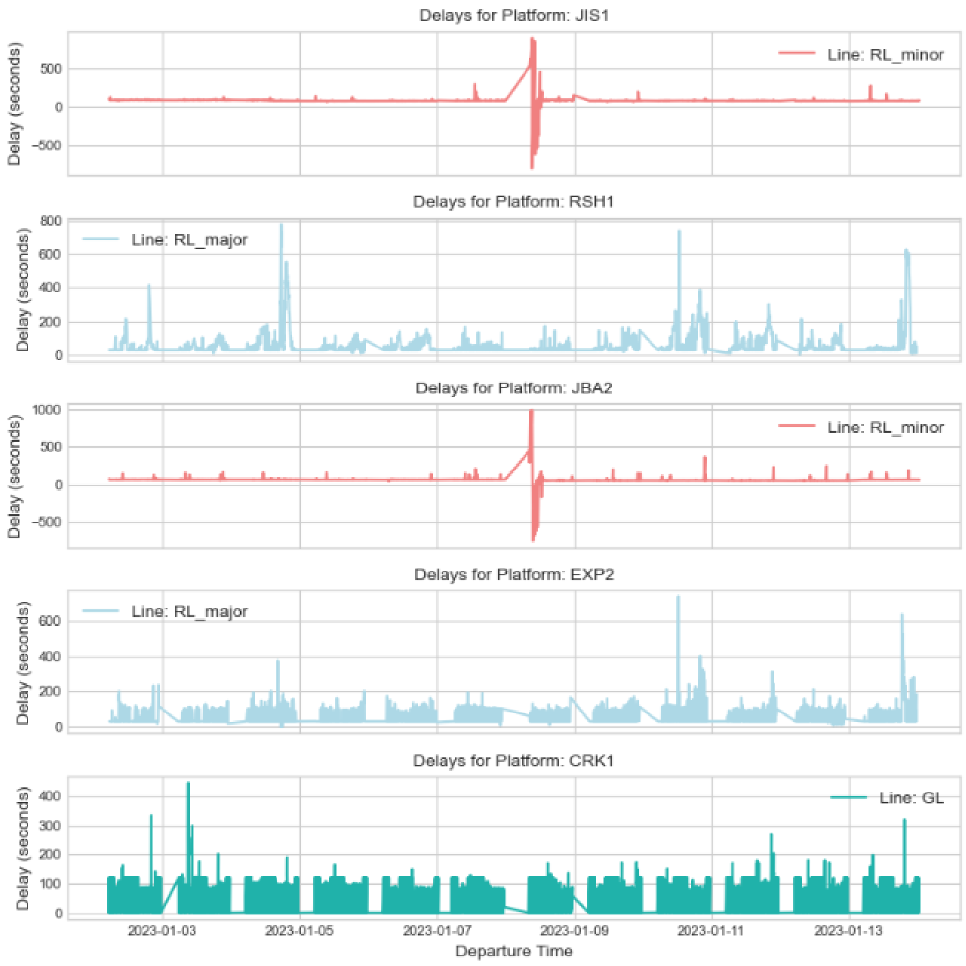


Fig. 21: Line Chart of Departure Delays from the Top 5 Most Delayed Platforms

3.2.5 Data Preparation for Modeling

The specific use case of this project encompasses the following objectives:

1. Primary Departure Delay Prediction: The first objective is to predict whether a primary departure delay of a train at a platform beyond a predefined threshold (in seconds) occurs. Once an instance of this primary delay is identified, the subsequent objective is to predict the secondary delay of the next train at the platform.

2. Temporal Impact Analysis: To understand the temporal impact and potential propagation of the primary delay, as well as preceding sequences of delays, might have on the next train, a window of the previous 'n' trains' features in the platform sequence up to the primary delay will be included as inputs. These features will be utilized along with the features of the secondary delayed train, for which the delay is being predicted.

The objectives are inspired by previous work done by Luo et al., 2022, who developed a similar model approach. With these use case objectives, custom data preparation will have to be done with user-defined functions to match the requirements.

The first function defined as *create_dataset* serves primarily as an intermediary step in the data preparation process; however, it can be utilized for cursory analysis as well. It involves creating a new data frame with only the relevant records or rows that meet the objective criteria. The function filters out irrelevant records before further processing is carried out to create input and outputs for the actual Machine Learning Models in the next function.

The following are the key elements of the *create_dataset* function:

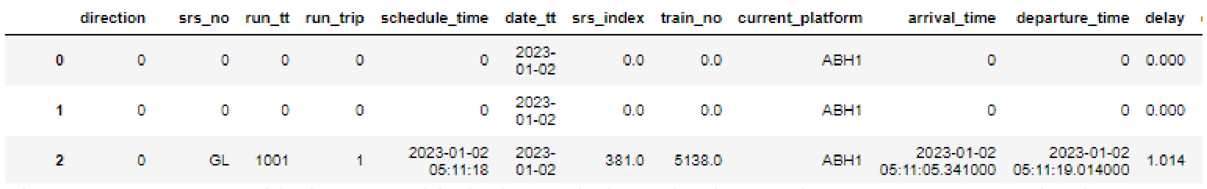
1. Function Parameters: The function takes in the original data frame, the user-defined delay duration, and the window of previous time step sequences (trains at platform) as parameters.

2. Filter Records for Each Platform and Date: Records for each unique platform that commence from the start of each new date form a distinct time series. Consequently, the records from only the time series at the current iteration in the loop are included and the others get filtered out.

3. Checking for Delay Criteria: If a record matches the specified delay criteria a window of the previous 'n' trains' records in the platform sequence along with the delayed instance and the subsequent train record, also referred to as the secondary delay record, are added to the data frame as relevant records for the use case. For example, if the window of previous time steps of 2 is specified as a parameter, then a total of 4 records will be added. Additional condition checks have been added allowing for more flexibility, which include:

    - In case a record from the first few rows of the time series matches the delay criteria but there are not enough previous records available, as specified by the window size, then additional zero-padded rows, which have all their variable values set as zero, will be added before the delayed instance record to ensure that the relevant record is added to the new data frame and is not omitted. For example, if a window size of 2 has been specified and there is only one previous record then one additional zero-padded record will be added to satisfy the window size criteria. Fig. 22 shows an example of the addition of two zero-padded rows when the window size is 2 and the primary delayed duration is set as 1 second. Only the date and current platform variables are not set as zero since these are identifiers that the records belong to a certain time series.

    - In case the window size parameter is set to zero then only the delayed instance and the secondary delay record are added to the data frame as the user does not want to investigate the impact of previous train sequences to predict the secondary delayed instance.



	direction	srs_no	run_tt	run_trip	schedule_time	date_tt	srs_index	train_no	current_platform	arrival_time	departure_time	delay
0	0	0	0	0	0	2023-01-02	0.0	0.0	ABH1	0	0	0.000
1	0	0	0	0	0	2023-01-02	0.0	0.0	ABH1	0	0	0.000
2	0	GL	1001	1	2023-01-02 05:11:18	2023-01-02	381.0	5138.0	ABH1	2023-01-02 05:11:05.341000	2023-01-02 05:11:19.014000	1.014

Fig. 22: Two Zero-Padded Rows Added when Window Size is 2 and Target Delay Duration is 1 second

Further to the *create_dataset* function, the next step is to use the data frame generated by this function to create input and output matrices. These matrices are intended for use with Supervised Machine Learning Models, specifically sequential RNN models such as LSTM and GRU, which capture temporal dependencies from time steps. For this, a new *time_series_generator* function is defined.

The following are the key elements of the *time_series_generator* function:

1. Function Parameters: Since this function incorporates the *create_dataset* function it also takes in the same parameters including the original data frame, the user-defined delay duration, and the window of previous time steps (trains at platform). Furthermore, it allows the user to select the list of features to include and set the target variable for the input and output matrices. For further flexibility and investigation, it allows the user to select whether to include the features of the secondary delay train record for which the delay value is being predicted.

2. Data frame Creation: The original data frame is filtered out using the *create_dataset* function and only the relevant records remain in the new data frame.

3. Filter Records for Each Platform and Date: The records from only the time series at the current iteration in the loop are included and the others get filtered out.

4. Checking for Delay Criteria: The delay criteria are checked then the relevant data records with specified features are added to an input list, while the target variables are added to the output list.

5. Current Time Step Parameter check: If current time step inclusion is set as true the features of the secondary delay record are also added to the input list. However, it should be noted that if the delay variable has been included in the features parameter, then its value will be set to zero in the input list for this record since this is the actual target value and must only be included in the output list. If the current time step parameter is set as False, then only the rows of previous 'n' trains' records along with the delayed instance and their respective features will be added to the input. For both cases, the delay value of the target record will be added to the output list.

6. Transforming Input and Output Lists to Tensors: The final input and output lists are transformed into tensors that are ready to be input into RNN models.

By creating the *create_dataset* and *time_series_generator* functions the raw data has been transformed into the relevant input and output formats and is ready to be input into Machine Learning models for predictions.

3.2.6 Train – Test Split for Dataset

With every new date on each platform being an individual time series the dataset and forecasting problem becomes multivariate. There are a total of 113 unique platforms and 12 date values, resulting in a total of 1356 distinct delay time series present in the dataset. Since the entire data will be trained and tested on one model, it must be ensured that each time series has equal representation as per their proportions in the dataset, while also maintaining the temporal order to prevent testing on earlier time step values than those found in the training set for each time series. To ensure this a new *test_train_split* function is defined, where each time series is equally represented in the test and train splits.

The following are the key elements of the function:

1. Function Parameters: The function takes in the original data frame and the test split proportion size, which is the percentage of data to be allocated to the test set.

2. Filter Records for Each Platform and Date: The records from only the time series at the current iteration in the loop are included and the others get filtered out.

3. Split the dataset: The test split proportion size will be used to find the split index of the data frame where it will divide all records before the index into the train set and all those beyond into the test set. They will then be added to each list before moving on to the next iteration and doing the same for the next time series. The final train and test lists are converted into data frames.

After the creation of the test and train data frames each set is transformed and pre-processed into time series data independently to avoid leakage of training dependencies and patterns into the test set, ensuring a more reliable evaluation of the model's performance on unseen data.

3.3 Building a Benchmark Model

3.3.1 Overview

A crucial step to begin the modeling stage involves building a foundational model, which will serve as a benchmark for more complex models to follow. To this end, the ensemble Random Forest Regressor Algorithm was chosen as the ideal benchmark. The Random Forest Algorithm consists of multiple decision trees that each train on random subsets and features of data then make output the final prediction as the average of all individual trees for regression problems (Géron, 2023). They notably display robustness to overfitting and can identify feature importance for the target variable. Compared to Neural Network models they are also computationally cheaper and have fewer hyperparameter considerations (Nabian et al., 2019). These considerations make it an appropriate algorithm to build a benchmark. From this benchmark, the data parameters, such as window size and feature combinations, that are most relevant for forecasting will be identified and used in the rest of the more complex deep learning models.

The evaluation of the model will be done using the RMSE and MAE metrics. The Random Forest Regression model was imported from the scikit-learn library. The aim of the model-building task is not only to improve performance but to also do it computationally efficiently as possible, so the performance improvement against training time will be a further point of consideration. It should be noted that training time depends on processor speed and other factors which may vary with each iteration.

3.3.2 Delay Duration Decision

Before building a model, it is pertinent to decide on the duration for the primary delay which will decide how many cases there are for training and testing of the model. From Fig. 23 we can see the probability distribution curve of train departure time deviations, or delays mostly fluctuate around 0, which can be considered be considered a random and normal level of deviation. To predict more meaningful secondary delays a larger primary delay duration which is less likely to be attributed to a normal time variation must be chosen and a sufficient number of cases should be present allowing for appropriate model training and testing.

From experimenting with different time durations, a primary delay of 30 seconds was chosen, which generated a total of 72,182 cases from the entire dataset of 332,531 records. From this, the cases are divided into train and test data of 75% and 25% consisting of 52,645 and 19,537 cases each based on an equal proportion representation of each time series from the custom test train split function.

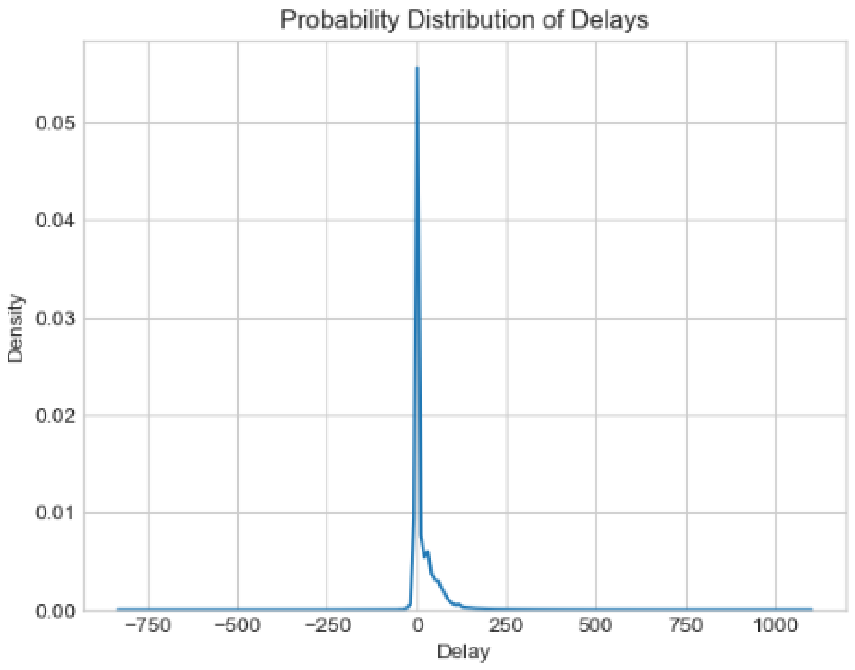


Fig. 23: Probability Distribution of Departure Delays (Time Variation)

3.3.3 Experimenting with Different Features

From the Feature Selection step and further experiments with the Random Forest Model, it was identified that spatiotemporal delay features, which consist of previous sequential train delays (d_{i-1}) and delays from previous platforms (d_{j-1}), were the only features that displayed a meaningful correlation for predicting the secondary delay for the next train, which is why a combination of these features will be checked. Table 1 lists the combinations that will be checked with the benchmark Random Forest Model. Since Random Forest Models are meant for supervised classification and regression predictions, they do not have inherent the ability to handle the temporal dimension like RNN models. Hence, the tensor dataset created needs to be reshaped into a 2D matrix before being used as model input.

Additionally, the following are the default data parameters and model hyperparameters initially set and previously decided upon for this experiment:

- Delay duration is 30 seconds.
- Features from the Secondary Delay Record are included.
- Window Size of Previous time steps before primary delay is set as 1. (Total 3 Timesteps including primary and secondary delay instances)
- 75/25 is the train-test split.
- Feature Scaling is done with Min-Max Scaler which showed better performance than Standard Scaler.
- The number of Estimators in the Random Forest Model is set as 100.
- Random State Hyperparameter is set to 42 to ensure consistency in selecting a subset of samples and features for the model each time.

Configuration Number	Feature Description	Total Features
1.	• Delay (One Train Back) (2 Timesteps) ($d_{i-1, j}$, $d_{i-2,j}$)	2
2.	• Delay (One Train Back) (2 Timesteps) ($d_{i-1, j}$, $d_{i-2,j}$) • Delay (One Platform Back) (3 Timesteps) ($d_{i, j-1}$, $d_{i-1,j-1}$, $d_{i-2,j-1}$)	5
3.	• Delay (One Train Back) (2 Timesteps) ($d_{i-1, j}$, $d_{i-2,j}$) • Delay (One Platform Back) (3 Timesteps) • Delay (Two Platforms Back) (3 Timesteps)	8
4.	• Delay (One Train Back) (2 Timesteps) ($d_{i-1, j}$, $d_{i-2,j}$) • Delay (One Platform Back) (3 Timesteps) ($d_{i, j-1}$, $d_{i-1,j-1}$, $d_{i-2,j-1}$) • Delay (Two Platforms Back) (3 Timesteps) ($d_{i, j-2}$, $d_{i-1,j-2}$, $d_{i-2,j-2}$) • Delay (Three Platforms Back) (3 Timesteps) ($d_{i, j-3}$, $d_{i-1,j-3}$, $d_{i-2,j-3}$)	11
5.	• Delay (One Platform Back) (3 Timesteps) ($d_{i, j-1}$, $d_{i-1,j-1}$, $d_{i-2,j-1}$) • Delay (Two Platforms Back) (3 Timesteps) ($d_{i, j-2}$, $d_{i-1,j-2}$, $d_{i-2,j-2}$) • Delay (Three Platforms Back) (3 Timesteps) ($d_{i, j-3}$, $d_{i-1,j-3}$, $d_{i-2,j-3}$)	9

Table 1: Feature Configurations for Benchmark Model

The target variable is the secondary delay ($d_{i, j}$). Once completed, the best features identified from this experiment shall be used in further experiments to develop the benchmark model.

3.3.4 Excluding the Secondary Delay Step Features

It is important to assess the ability of the model in predicting secondary delay durations from just the primary delay and previous sequential window features, without considering the features of the secondary delay sequence as in real-life scenarios dispatchers can take better action knowing the delay of the next train at the platform when there is more time to take action then predicting the delay when a train in the trip, or sequence, has already been delayed and departed from the last platform which leaves less time for effective intervention.

In this experiment the model's performance and impact with and without secondary delay sequence features will be evaluated and assessed to determine the level of importance including secondary delay features has on the model. The rest of the model's parameter and hyperparameter configurations will remain the same as those decided upon in the previous feature combination experiment to maintain consistency.

3.3.5 Experimenting with Different Time Step Window Sizes

Deciding on the window size of past time steps, or sequences, to the primary delay to include as model features can help identify how prevalent the impact of propagation of delays is across time steps on the secondary delay being predicted.

For this experiment window sizes of 0 to 4 were explored. A window size of 0 simply focuses solely on the impact of the primary delay instances' features in predicting the secondary delay duration. Conversely, a window size of 4 includes the features of the 4 preceding time step trains along with the primary delay instance to predict the secondary delay duration. The rest of the model's parameters and hyperparameters remain the same as before. The features of the secondary delay sequence have also been included in this experiment.

3.3.6 Grid Search for Hyperparameter Tuning

Once the parameters for the data have been established, the final step in developing the best benchmark model is to decide on the hyperparameters that are best able to capture the data's fit and underlying patterns. Employing Grid Search, a comprehensive exploration of a variety of Hyperparameters and their combinations will be utilized to give the best model performance. This process involves performing cross-validation on the training data from the Random Forest model. Grid Search will be carried out with the *GridSearchCV* class from the scikit-learn library. The cross-validation value for the search is kept as 5 as an increased value would lead to significantly increased training time and computational cost, while the scoring metric was set to mean squared error. The hyperparameters and their test values that will be explored are listed below in Table 2.

Hyperparameters	Test Values
random_state	0, 42, 99
n_estimators	50, 100, 200
max_depth	None, 5, 10
min_samples_leaf	1, 2, 4
bootstrap	True, False

Table 2: List of Hyperparameters and Tested Values for Grid Search in the Random Forest Model.

The following is a brief description of each of these hyperparameters:

- **random_state**: Ensures that for a single random state, each time a Random Forest Model is run the subset of features and samples is the same.
- **n_estimators**: The number of individual decision trees in the Random Forest Model.
- **max_depth**: The maximum number of levels each decision tree can have.
- **min_samples_leaf**: The minimum number of samples a leaf node must have.
- **bootstrap**: Whether to include bootstrap samples, where random sampling with replacement is done to create subsets of data for each tree (Géron, 2023).

The results of the grid search will reveal which combination of hyperparameters performed the best on the training data. The configurations will then subsequently be evaluated on the test set to see if the same combination of hyperparameters also performs optimally on unseen data.

Building upon the optimal data parameters and hyperparameters derived through the experiments, the forthcoming more complex deep learning models will aim to surpass the benchmark Random Forests in terms of evaluation metrics.

3.4 Building Deep Learning Models

3.4.1 Baseline LSTM Model

Upon establishing the benchmark Random Forest model, more complex Deep Learning Models will be built. The next model to be evaluated will be the LSTM model, which is better suited for sequential data and capturing time series dependencies. The data parameters found to be the best from the Random Forest Model will be tested as foundational considerations, except larger sequence window sizes will also be considered. The larger window sizes are explored to explore and exploit temporal patterns that may have not been identified by the limits of the Random Forest model.

Various combinations of the following hyperparameters will be checked when developing the initial model architecture that includes:

- Number of LSTM and Dense Layers (Excluding Output Layer)
- Number of LSTM and Dense Layer Neurons
- Optimizer Learning Rate Configuration
- Number of Epochs
- Total Training Batch Size

A combination of LSTM and Dense Layer architectures will be experimented with to try effectively capture temporal dependencies and more complex non-linear patterns that may be present. The different architectural variations will be undergoing evaluation against the test set. In Neural Networks, the weights and biases undergo random initialization. To ensure consistency and reproducibility, a constant random seed value of 42 will be maintained throughout. Subsequently, the architecture that exhibits superior performance while also being computationally efficient will be set as the foundational LSTM model.

During the development of the baseline model, the activation functions for the LSTM layers will be kept as the default Hyperbolic Tangent (Tanh) and those of the Dense, except the output layer, were kept as Rectified Linear Unit (ReLU). For this project, it was decided that the neurons and activation functions for each layer will be kept the same.

The evaluation metrics of the models will remain as the RMSE and MAE. Further hyperparameter tuning and experiments that will be conducted will compare against this baseline model.

3.4.2 Random Search for Hyperparameter Optimization

Deep Learning models like the LSTM model consist of an enormous amount of hyperparameters with most permutations not able to be feasibly explored due to time and computational restrictions. Thus, employing an exhaustive method like Grid Search was not deemed viable for the exploration of the LSTM model's hyperparameters. Instead, the more feasible Random Search method will be employed, which randomly combines and evaluates a predefined number of specified hyperparameters. From this method, other unexplored and potentially superior hyperparameters can be identified than those used to build the baseline LSTM model. The Random Search class is imported from the Keras tuner library. A total of 20 trials will be conducted to thoroughly explore various architecture and hyperparameter combinations. The hyperparameters and their test values that will be explored are detailed below in Table 3.

Hyperparameters	**Test Values**
LSTM Layers	1 – 4
Dense Layers	1 – 4 (Excluding Output Layer)
LSTM Layer Neurons	128, 256
Dense Layer Neurons	30, 40, 50
Dense Layer Activation Functions	ReLU, Tanh, Sigmoid
Learning Rate	0.005, 0.001, 0.0005

Table 3: List of Hyperparameters and their Corresponding Test Values for Random Search in the LSTM model.

Certain hyperparameters were held constant throughout the Random Search process:

- LSTM Layer Activation Function
- Optimizer
- Training Batch Size
- Number of Epochs

These hyperparameters remain the same as the baseline as exploring them in addition to the others in the random search would not be computationally efficient and would mean more trials would need to be conducted. However, further experiments will be conducted to explore the impact of these hyperparameters individually.

3.4.3 Experimenting with Different Epochs

Once the architecture of the LSTM has been finalized based on the outcome and performance of the models from the previous experiment this experiment will further explore different numbers of epochs to try and maximize the model's efficacy and capability while ensuring that it does not overfit due to an excessive number of epochs leading to worse test data performance or, conversely, inadequate utilization due to a low number of epochs. Another point of exploration is if there is an improvement in performance with the increase in epochs whether it is significant or only marginal, potentially reaching a point of diminishing returns with increased computational demands.

The number of epochs that will be explored is:

- 20
- 50
- 100
- 150

3.4.4 Experimenting with Different Batch Sizes

Further investigation of the batch size is required to evaluate the performance of models while trying to limit the computational costs. Smaller batch sizes are better able to capture the noise in the data and make better generalizations on unseen data and converge toward lower loss values but can be computationally expensive while training. In contrast, higher batch sizes are more computationally efficient but are not able to generalize as well. A larger batch size that attains equal or improved performance is preferred to a smaller one.

The different batch sizes that will be explored are:

- 10
- 50
- 100

3.4.5 Experimenting with Different Learning Rates

While learning rates were previously investigated through a random search, this experiment reexamines them once again with higher values to ascertain if they achieve similar or better loss convergence faster within a fewer number of epochs. This would make them more computationally efficient.

The different learning rates that will be explored are:

- 0.001
- 0.01
- 0.05

3.4.6 Experimenting with Different Deep Learning Models

Once the best configuration of the LSTM model has been identified, its performance on the test set and evaluation metrics will be compared to the Benchmark Random Forest Model. Additionally, two alternative deep learning models will be investigated, namely the GRU and Feedforward ANN models, which will be compared to the established LSTM model.

The GRU model is another RNN variant that exhibits the capability to capture temporal dependencies. A distinguishing feature it has is its fewer parameters due to its simpler structure for an identical number of neurons and layers. Therefore, it will be more efficient to train than an LSTM and can potentially be prone to less overfitting. In this experiment, for a fair and consistent comparison, the hyperparameters and architecture for the GRU will be kept the same as the previously established LSTM, which will then evaluate the performance and training time for the model.

In contrast to both the LSTM and GRU models, the Feedforward ANN model does not specifically handle sequential or temporal data. Nonetheless, if sequences are relatively short then the model might be sufficient to capture patterns. This experiment serves the purpose of gauging the ANN's proficiency and revealing the extent to which long-term dependencies influence the data's performance. The hyperparameters for the ANN will remain the same with the exception of the addition of LSTM layers. Similar to the Benchmark Random Forest Model since the ANN is not designed with time dependencies in mind the data input generated from the pre-processing stage must be reshaped from a tensor into a 2D matrix, similar to the Random Forest, to be processed by the ANN.

3.5 Building a Support Vector Machine Model

For further model exploration beyond the Benchmark Random Forest and Deep Learning Models another conventional Machine Learning, namely the Support Vector Machine (SVM) Regression model will be implemented. The introduction of the SVM model is driven by a dual purpose: firstly, to assess the performance impact of the Model in comparison to Random Forest and Deep Learning Models; secondly, to further assess the influence of temporal sequences on model performance. This is because the SVM does not inherently handle temporal sequences.

The SVM model will be imported from sci-kit learn and its main hyperparameters that will be tweaked are the regularization parameter *C*, which affects the model's complexity and the ϵ value that affects the margin width (Géron, 2023). The underlying kernel function will remain fixed as the Radial Basis Function (RBF), which is well suited for non-linearity.

From this comprehensive exploration of different Deep Learning and Machine Learning models, the aim is to not only identify optimal models for the dataset but also to shed light on a combination of factors that includes model efficiency, data characteristics, and performance outcomes. The result will be the optimal model and parameters for the data.

4. Results and Evaluation

4.1 Results of Benchmark Model

4.1.1 Results with Different Feature Combinations

Table 4 displays the results of the different feature combination experiments using the Random Forest Model on the test set as well as the overall training time.

Feature Combination	Evaluation Metrics (Test Set)		
	MAE	RMSE	Training Time (seconds)
• Delay (One Train Back) (2 Timesteps) ($d_{i-1,j}$, $d_{i-2,j}$)	28.823	44.550	16.89
• Delay (One Train Back) (2 Timesteps) ($d_{i-1,j}$, $d_{i-2,j}$) • Delay (One Platform Back) (3 Timesteps) ($d_{i,j-1}$, $d_{i-1,j-1}$, $d_{i-2,j-1}$)	4.965	14.243	36.70
• Delay (One Train Back) (2 Timesteps) ($d_{i-1,j}$, $d_{i-2,j}$) • Delay (One Platform Back) (3 Timesteps) ($d_{i,j-1}$, $d_{i-1,j-1}$, $d_{i-2,j-1}$) • Delay (Two Platforms Back) (3 Timesteps) ($d_{i,j-2}$, $d_{i-1,j-2}$, $d_{i-2,j-2}$)	4.961	14.497	52.77
• Delay (One Train Back) (2 Timesteps) ($d_{i-1,j}$, $d_{i-2,j}$) • Delay (One Platform Back) (3 Timesteps) ($d_{i,j-1}$, $d_{i-1,j-1}$, $d_{i-2,j-1}$) • Delay (Two Platforms Back) (3 Timesteps) ($d_{i,j-2}$, $d_{i-1,j-2}$, $d_{i-2,j-2}$) • Delay (Three Platforms Back) (3 Timesteps) ($d_{i,j-3}$, $d_{i-1,j-3}$, $d_{i-2,j-3}$)	4.980	14.523	72.43
• Delay (One Platform Back) (3 Timesteps) ($d_{i,j-1}$, $d_{i-1,j-1}$, $d_{i-2,j-1}$) • Delay (Two Platforms Back) (3 Timesteps) ($d_{i,j-2}$, $d_{i-1,j-2}$, $d_{i-2,j-2}$) • Delay (Three Platforms Back) (3 Timesteps) ($d_{i,j-3}$, $d_{i-1,j-3}$, $d_{i-2,j-3}$)	8.589	22.555	58.56

Table 4: Results from Feature Configuration Experiments for Benchmark Model

The results indicate that the best performance on the test set in terms of RMSE is achieved by incorporating the previous time-step delays and previous platform delay from one platform (Experiment 2), returning a value of 14.243. Although, the MAE is marginally smaller than Experiment 3 at 4.965 compared to 4.961. Through this, it is confirmed that the features of Experiment 2 are sufficient when it comes to predictive accuracy. The inclusion of additional features from Experiments 3 and 4 might introduce noise and increase computation costs while increasing the training time. Their performance can be attributed to the Random Forest Model's ability to recognize feature importance, as they include all the features from Experiment 2 as well. Therefore, the features of Experiment 2 ($d_{i-1, j}$, $d_{i-2,j}$, $d_{i, j-1}$, $d_{i-1,j-1}$, $d_{i-2,j-1}$) are finalized for all further modeling and experiments.

Furthermore, the findings reaffirm the insights seen from the correlation analysis that showed delays from preceding platforms (inter-platform delays) in the spatial dimension have better predictive power at forecasting delays at future platforms compared to delays from previous time steps (inter-train delays). Experiment 1 did not include inter-platform delays and performed noticeably worse on all metrics than Experiment 5, which did not include inter-train delays from the target platform.

4.1.2 Results Excluding the Secondary Delay Step Features

Table 5 displays the comparison of the Random Forest model's performance by excluding and including the secondary delay sequence features.

Secondary Delay Features Inclusion	Evaluation Metrics (Test Set)		
	MAE	RMSE	Training Time (seconds)
Included	4.965	14.243	33.20
Excluded	26.938	42.055	28.44

Table 5: Results from Inclusion and Exclusion of Secondary Delay Features for Benchmark Model

The best feature combinations from the previous experiment are maintained for this experiment. Comparing the performance of the models it is evident that the Model from Experiment 1 significantly outperforms Experiment 2 across all evaluation metrics on the test set. The findings of the experiment reaffirm the conclusions from the previous experiment that inter-platform delays from previous platforms in the same sequence, or time-step, have a more significant impact on the delay prediction for the next platform than inter-train delays from previous time-steps on the same platform. Therefore, including previous platform delays for the target secondary delay time step is crucial for developing a better and more accurate delay prediction model from the data available. This confirms that from the existing data, it is not accurate or sufficient to predict a secondary delay at a platform solely from the primary delay and preceding train sequences and features.

4.1.3 Results with Different Time Step Window Sizes

Table 6 displays the comparison of the Random Forest model's performance with the different number of window sizes, reflecting the number of time-steps, or sequences preceding the primary delay.

Window Size	Evaluation Metrics (Test Set)		
	MAE	RMSE	Training Time (seconds)
0	5.704	16.270	22.40
1	4.965	14.243	33.74
2	4.898	13.875	48.05
3	4.962	13.977	60.58
4	5.073	14.169	71.94

Table 6: Results from Different Window Size Experiments for Benchmark Model

Maintaining the best findings from the previous experiments, the results show that Experiment 3, which incorporates the primary delayed train, and the previous 2 sequential train features have the best performance across all metrics on the test set. This could also potentially be because the Random Forest model may not be able to capture more complex temporal patterns from Experiments that incorporate a larger window size. Additionally, increasing window sizes might also introduce noise and can potentially lead to overfitting on the training set.

Furthermore, Experiment 1, where the window size is set as 0, demonstrates that despite same time step previous platform delay features having superior predictive power for forecasting delays at future platforms in comparison to previous time step delay features forecasting future train delays, relying solely on primary delay features does not give the best performance overall. It can then be concluded that a combination of both inter-train and inter-platform delay features is important to creating the best model.

4.1.4 Results of Grid Search for Hyperparameters

The results of the Grid Search CV of the training set reveal the best hyperparameters as the following:

- **random_state**: 42
- **n_estimators**: 200
- **max_depth**: None
- **min_samples_leaf**: 4
- **bootstrap**: True

However, upon evaluating these hyperparameters on the test set, it became evident that the default hyperparameters of the Random Forest Regressor, along with a random state of 42, yielded superior results. As a result, a re-evaluation was conducted for the top five hyperparameter combinations found through grid search. Table 7 displays the test set evaluation of the top five hyperparameter combinations from the Grid Search.

Hyperparameters	Evaluation Metrics (Test Set)		
	MAE	RMSE	Training Time (seconds)
• random_state: 42 • n_estimators: 200 • max_depth: None • min_samples_leaf: 4 • bootstrap: True	4.899	14.688	65.16
• random_state: 42 • n_estimators: 100 • max_depth: None • min_samples_leaf: 4 • bootstrap: True	4.921	14.680	32.54
• random_state: 42 • n_estimators: 50 • max_depth: None • min_samples_leaf: 4 • bootstrap: True	4.939	14.729	17.33
• random_state: 0 • n_estimators: 200 • max_depth: None • min_samples_leaf: 4 • bootstrap: True	4.908	14.708	73.85
• random_state: 0 • n_estimators: 100 • max_depth: None • min_samples_leaf: 2 • bootstrap: True	4.850	14.018	42.32

Table 7: Top 5 Hyperparameter Configurations from Grid Search for Benchmark Model

None of the re-evaluated top 5 hyperparameter configurations were found to be as optimal on the test set as the default hyperparameters with a random state of 42. Therefore, the default hyperparameters were maintained for the final evaluation of the benchmark model.

4.1.5 Results of Established Benchmark Model

The final data parameters established from the experiments and those previously decided upon are displayed in Table 8, while the final Hyperparameters and Evaluation Metrics for the Benchmark Random Forest Model are displayed in Tables 9 and 10 respectively.

Data Parameters	Values
Features	• Delay (One Train Back) (2 Timesteps) ($d_{i-1,j}$, $d_{i-2,j}$) • Delay (One Platform Back) (3 Timesteps) ($d_{i,j-1}$, $d_{i-1,j-1}$, $d_{i-2,j-1}$)
Secondary Delay Features Inclusion	Included
Window Size	2
Primary Delay Duration	30 seconds
Train-Test Split	75/25
Feature Scaling Method	Min-Max Scaler

Table 8: Final Data Parameters for All Models

Hyperparameters	Values
random_state	42
n_estimators	100 (Default)
max_depth	None (Default)
min_samples_leaf	1 (Default)
Bootstrap	True (Default)

Table 9: Final Hyperparameters for the Benchmark Random Forest Model

Model	Evaluation Metrics (Test Set)		
	MAE	RMSE	Training Time (seconds)
Random Forest	4.898	13.875	34.04

Table 10: Final Evaluation Metrics for the Benchmark Random Forest Model

4.2 Results of Deep Learning Models

4.2.1 Baseline LSTM Model Results

Upon finalizing the data parameters and establishing a benchmark model, further model building was conducted with more complex Deep Learning Models. Initially, only LSTM layers were considered with the addition of a single dense neuron. The neuron served the purpose of transforming the data shape into an array of prediction values that can be compared to the actual values. Further experiments integrated more hidden dense layers, which yielded improvement in model performance.

In terms of further exploration of data parameters from those identified in the Benchmark model, the increased sequences of the window size beyond 2 did not yield improved performance on the test dataset and it only increased computational cost, simultaneously increasing the potential overfitting. Hence, the parameters found from the benchmark model are retained for the LSTM model experiments.

For optimization, the RMSprop algorithm was examined and found to return generally worse results compared to the Adam algorithm. Finally, after experimenting with various architectures and hyperparameters the best model was a hybrid LSTM-ANN model. The best hyperparameters found are presented in Table 11 and the evaluation metrics of the model are displayed in Table 12.

Hyperparameters	Values
LSTM Layers	1
LSTM Layer Neurons	128
LSTM Layer Activation Function	Tanh (Default)
Dense Layers (Excluding Output Layer)	2
Dense Layer Neurons	40
Dense Layer Activation Function	ReLU
Optimizer	Adam
Learning Rate	0.001
Epochs	20
Batch Size	50

Table 11: Baseline LSTM-ANN model Hyperparameters

Model	Evaluation Metrics (Test Set)		
	MAE	RMSE	Training Time (seconds)
Baseline LSTM-ANN	6.592	13.496	1910.66

Table 12: Evaluation Metrics for the Baseline LSTM-ANN Model

4.2.2 Results of Random Search for Hyperparameters

The random search aimed to identify a better architecture and combination of hyperparameters than the one established for the baseline LSTM-ANN model. It was set to evaluate based on the RMSE. A total of 20 trials were conducted and a consistent batch size of 50 and epoch count of 20 were maintained. Table 13 showcases the top 5 results attained from these trials and the hyperparameter combinations.

Hyperparameters	RMSE (Test Set)
• **LSTM Layers: 3** • **LSTM Layer Neurons: 256** • **Dense Layers: 4** • **Dense Layer Neurons: 30** • **Dense Activation Function: ReLU** • **Learning Rate: 0.0005**	13.344
• **LSTM Layers: 3** • **LSTM Layer Neurons: 256** • **Dense Layers: 4** • **Dense Layer Neurons: 40** • **Dense Activation Function: ReLU** • **Learning Rate: 0.001**	13.410
• **LSTM Layers: 1** • **LSTM Layer Neurons: 128** • **Dense Layers: 2** • **Dense Layer Neurons: 30** • **Dense Activation Function: ReLU** • **Learning Rate: 0.001**	13.435
• **LSTM Layers: 2** • **LSTM Layer Neurons: 128** • **Dense Layers: 1** • **Dense Layer Neurons: 30** • **Dense Activation Function: ReLU** • **Learning Rate: 0.001**	13.672
• **LSTM Layers: 2** • **LSTM Layer Neurons: 256** • **Dense Layers: 4** • **Dense Layer Neurons: 50** • **Dense Activation Function: ReLU** • **Learning Rate: 0.005**	13.794

Table 13: Top 5 Hyperparameter and Architecture Configurations from Random Search for the LSTM-ANN Model

The results from the table reveal that a wide variety of architectures and hyperparameters produce very similar outcomes, with the test set RMSE values ranging between 13.344 and 13.794 for the top 5 trials. Notably, models employing the ReLU activation function in the dense layers consistently outperform those employing sigmoid and hyperbolic activation functions, since all the top-performing models have it as their dense layer activation function. Among these trials, only the top 3 exhibit superior RMSE scores compared to the baseline LSTM-ANN model developed. Remarkably, the third-ranking trial has a far simpler architecture, featuring only 1 LSTM layer and 2 Dense Layers, while its RMSE is only 0.091 higher than the best result. This configuration is nearly identical to the baseline developed with fewer hidden dense layer neurons (30 compared to 40). Consequently, the third-ranking model is selected for all further experiments due to its simpler and more computationally efficient architecture.

4.2.3 Results of Experimenting with Different Epochs

Upon establishing the architecture of the LSTM-ANN model further experiments were conducted with hyperparameters that were not explored in the random search due to the associated escalated computational demands. Table 14 shows the results of the experiment with a different number of epochs.

Epochs	Evaluation Metrics (Test Set)		
	MAE	RMSE	Training Time (seconds)
20	6.277	13.435	238.99
50	6.083	12.786	647.82
100	5.507	12.482	1441.68
150	4.974	11.785	2233.45

Table 14: Results from Different Numbers of Epochs Experiment

The result findings indicate that increasing the number of epochs does improve the model's performance allowing it to be further refined and capture more complex patterns. Nonetheless, increasing epochs does result in a significant increase in training time and computational cost as well. The experiment with 150 epochs took 2233.45 seconds compared to 1441.68 seconds for its 100-epoch counterpart while improving the RMSE score by only 0.697. Hence, experimenting with more epochs than the ones tested was not deemed computationally efficient as well as the risk of overfitting. Consequently, 150 epochs were decided as the optimal number of epochs for the LSTM-ANN model given it had the best score across all metrics.

4.2.4 Results of Experimenting with Different Batch Sizes

Having ascertained the optimal number of epochs, the next hyperparameter explored was the effect of distinct training batch sizes. The results of the experiment are displayed in Table 15.

Batch Size	Evaluation Metrics (Test Set)		
	MAE	RMSE	Training Time (seconds)
100	5.394	12.231	751.78
50	4.974	11.785	1840.96
10	4.897	11.876	8397.00

Table 15: Results from Different Batch Sizes Experiment

The results substantiate that decreasing the batch size generally improves the model performance on test data up to a certain limit. With lower batch sizes the model can capture more nuanced patterns and variability within smaller batch subsets. Nevertheless, this seems to plateau at a point and while the returns are diminishing the training time and computational costs rise significantly. The results show that Batch sizes 50 and 10 give similar results but the training time is significantly larger at 8397 seconds as opposed to 1840.96 seconds for the lower batch size. Hence, a batch size of 50 is retained for the model.

4.2.5 Results of Experimenting with Different Learning Rates

Following the determination of optimal batch sizes, further exploration with higher learning rates that aimed to hasten convergence and were not explored in the random search was carried out. The results of the experiment are displayed in Table 16.

Learning Rate	Evaluation Metrics (Test Set)		
	MAE	RMSE	Training Time (seconds)
0.001	4.974	11.785	2142.57
0.01	4.819	11.862	2180.11
0.05	6.702	14.387	2239.39

Table 16: Results from Different Learning Rates Experiment

The results show that the higher learning rates are yielding worse results on the test set and are potentially introducing instability and overshooting the more optimal points, rather than improving the convergence of the model. Hence, the previously ascertained lower learning rate of 0.001 is retained for the model.

4.2.6 Results of Final LSTM-ANN Model

The final LSTM-ANN model architecture and hyperparameters established from the experiments are displayed in Table 17, while the final Evaluation Metrics for the Model are displayed in Table 18.

Hyperparameters	Values
LSTM Layers	1
LSTM Layer Neurons	128
LSTM Layer Activation Function	Tanh (Default)
Dense Layers (Excluding Output Layer)	2
Dense Layer Neurons	30
Dense Layer Activation Function	ReLU
Optimizer	Adam
Learning Rate	0.001
Epochs	20
Batch Size	50

Table 17: Final LSTM-ANN model Hyperparameters

Model	Evaluation Metrics (Test Set)		
	MAE	RMSE	Training Time (seconds)
Final LSTM-ANN	4.974	11.785	1808.46

Table 18: Evaluation Metrics for the Final LSTM-ANN Model

Furthermore, the test and training loss for the final LSTM-ANN model is shown in Fig. 24.

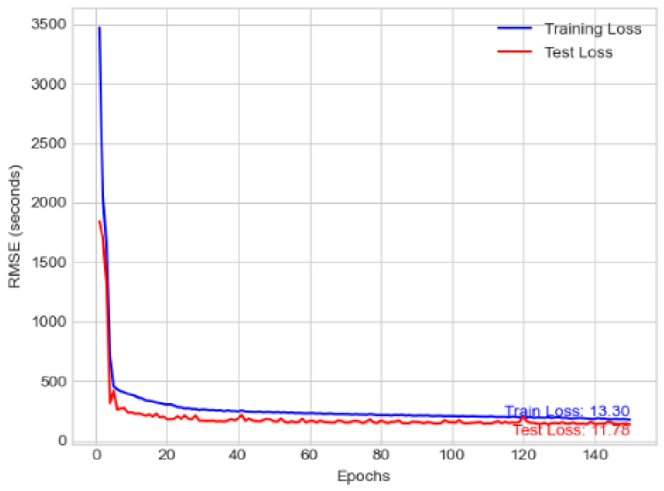


Fig. 24: Training and Test Loss for the LSTM-ANN Model

Upon establishing the Final LSTM-ANN model, subsequent alternative deep learning model types that were explored were the GRU and ANN. The established architecture and hyperparameters were maintained from Table 18. However, for the GRU the LSTM layer was replaced with a GRU one, making it a GRU-ANN hybrid model and for the ANN the LSTM layer was excluded outright making it a purely feedforward network model.

4.3 Results of the Support Vector Machine Model

The final model investigated was the Support Vector Machine (SVM). Following experimentation and hyperparameter tuning, the optimal hyperparameter values based on the performance on the test dataset entailed a C value of 5000 and ∈ of 1. This implied fitting the training data closely for capturing complex relationships while allowing a relatively small margin of error around the predicted values offered the best generalizability. Table 19 displays the evaluation metric results of the model on the test dataset.

Model	Evaluation Metrics (Test Set)		
	MAE	RMSE	Training Time (seconds)
Support Vector Machine	5.351	12.816	359.80

Table 19: Evaluation Metrics of the SVM Model

4.4 Comparison of Model Performances and Analysis

Table 20 displays the evaluation metric results from all of the models on the test dataset. Fig. 25 shows a visualized model comparison of the different evaluation metrics in the form of bar charts.

Model	Evaluation Metrics (Test Set)		
	MAE	RMSE	Training Time (seconds)
Random Forest	4.898	13.875	34.04
Support Vector Machine	5.351	12.816	359.80
LSTM-ANN	4.974	11.785	1808.46
GRU-ANN	5.100	12.498	1698.57
ANN	7.255	14.651	615.49

Table 20: Evaluation Metrics of All Tested Models

The outcomes reveal that the LSTM-ANN model has the best RMSE score from its counterparts, while the Random Forest exhibits a slightly better MAE score. This suggests that the Random Forest can capture the general pattern of delays, whereas the LSTM-ANN can better identify anomalous delay values. The SVM model also appears to be able to slightly capture anomalous delay values better than the Random Forest and has a lower RMSE.

It is noteworthy that the ANN, SVM and Random Forest models do not inherently handle temporal data, but yield similar performance to the LSTM-ANN and GRU-ANN. This suggests that the temporal dependencies of previous delays at platforms are mostly short-term and do not propagate as much across sequences for the temporal dimension.

The training time, however, reveals that the Random Forest is significantly more computationally efficient than the SVM and other Deep Learning Models, as displayed in Fig. 25, while able to adjust to non-linearity and capture the patterns to give suitable results. Fig. 26 displays each model's predicted values against the actual values for the test set and offers a cursory visual assessment of the performance exhibited by each model. Pointedly, the deviation of residual points beyond the reference dotted line serves to emphasize inferior performance.

Finally, owing to the best RMSE score the LSTM-ANN model is decided upon as the best model due to it having a good ability to in capturing non-linear relationships and effectively capture temporal patterns in the data even for outlier values compared to the alternatives.

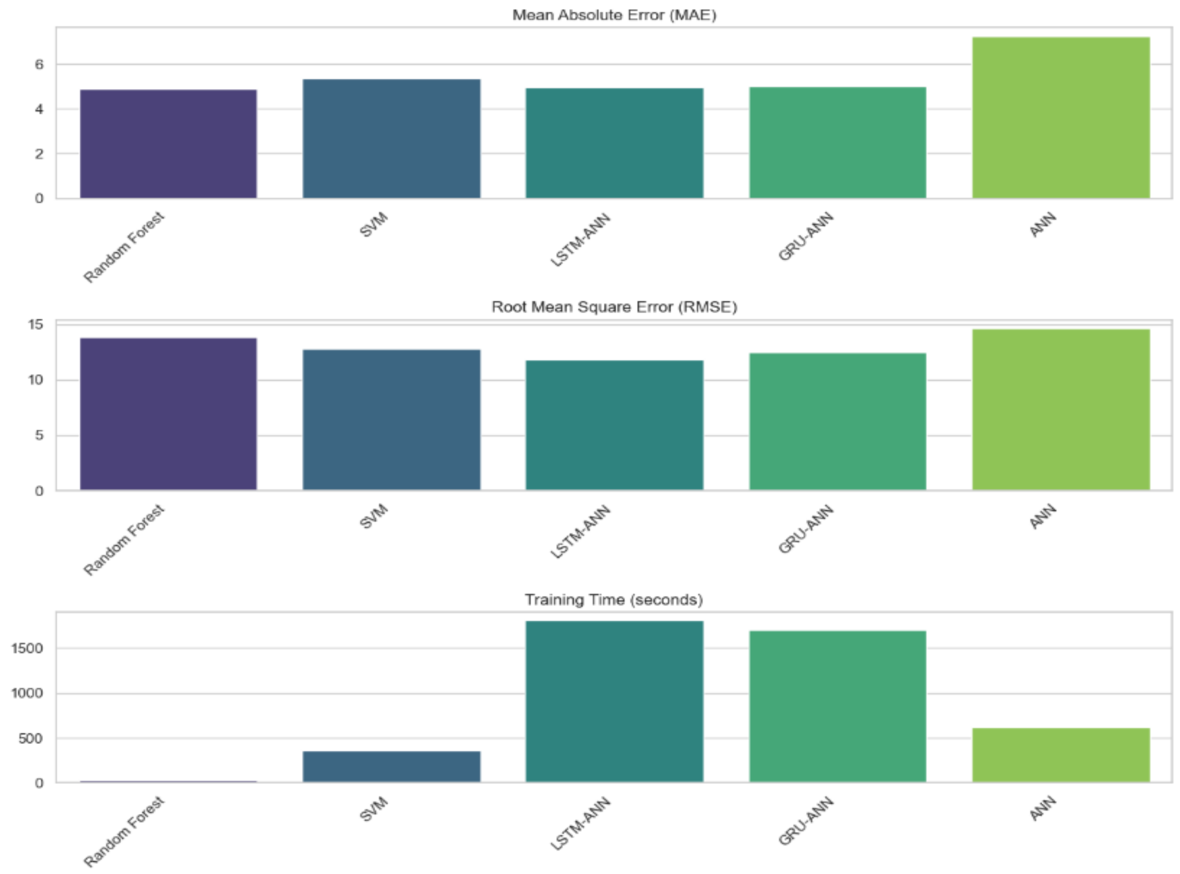


Fig. 25: Evaluation Metrics Model Comparisons for (a) MAE (seconds) (b) RMSE (seconds) (c) Model Training Time (seconds)

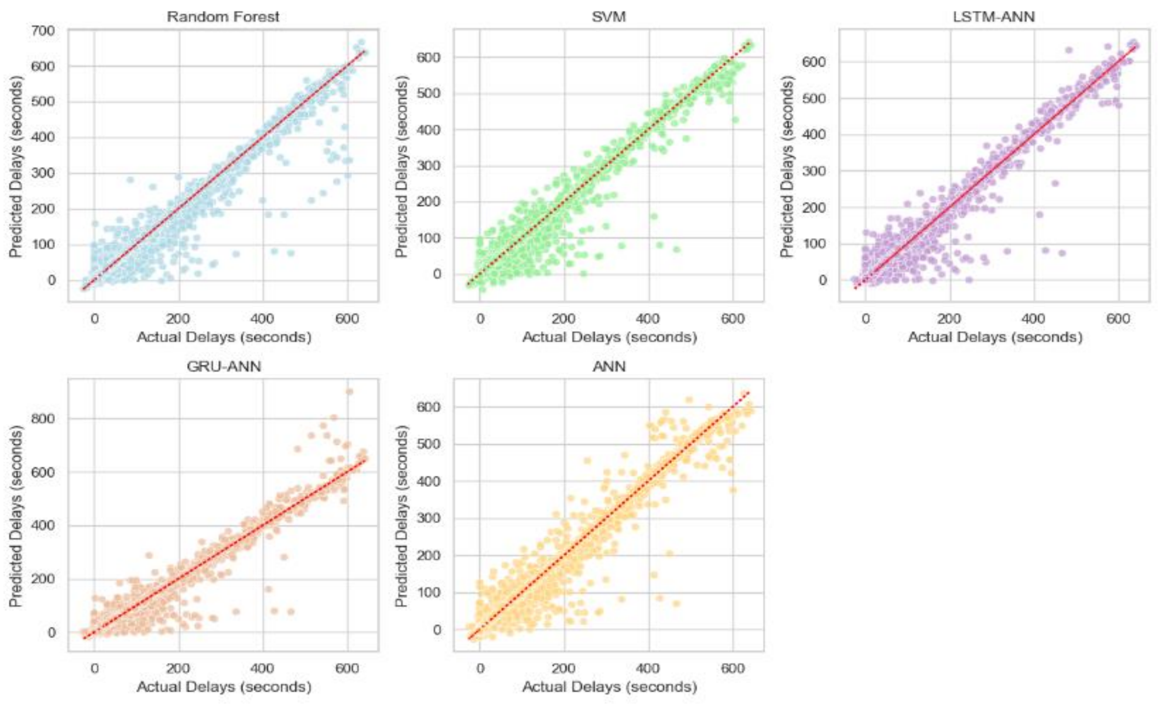


Fig. 26: Actual and Predicted Secondary Delays for Each Model

4.5 Challenges and Limitations

A notable challenge faced during the feature engineering and subsequent feature selection stage was trying to find and create features that independently correlated with the delay prediction target. However, the features created such as the dwell time, planned and arrival intervals and travel times all displayed a weak linear correlation with the target and did not offer any discernible improvements when incorporated into any of the models either. Furthermore, the multicollinearity observed between platform delay and train delay features presents a challenge in the model's interpretability. While these features display correlation with each other, the trade-off was made in favor of achieving better predictive performance.

Another challenge pertained to minimizing the error metrics within reasonable computational limits. It became apparent that all the models start plateauing at similar points which could be due to efforts of the models to adapt to outlier delay values. Further efforts to minimize the loss increased the computational cost significantly and had diminishing returns. As a result, all tested models have similar final error margins with RMSE spanning between the intervals 11.785 and 14.651.

Moreover, the accuracy of the model to predict future delays of multiple trains, or time-steps ahead is limited due to the platform's delay strongly correlating with the previous platform's delay far more than it is correlating with the primary delay from the previous train that arrived on the platform, or any of the previous time step features, which would mean that dispatchers have limited time to make meaningful decisions.

Concurrently, data availability was a limitation faced as train operation data was only available for 11 days from 2nd January 2023 to 13th January 2023. Further availability and training of data with primary delay duration parameters exceeding those encapsulated within the current dataset could lead to a more robust model. Furthermore, the inclusion of an additional month's worth of data for validation purposes could test the generalizability and effectiveness of the developed model.

5. Conclusion and Future Work

5.1 Conclusion

In this project, we sought to forecast the duration of secondary train departure delays at specific platforms within the Dubai Metro Network, which would be triggered when a preceding train's primary delay exceeded a predefined threshold. The motivation behind this was to accurately quantify the impact and influence exerted by primary delays on the subsequent train's departure punctuality. This would enable dispatchers to implement effective mitigation measures.

To achieve the project's aim we developed custom pre-processing functions to extract relevant records for platforms meeting the criteria of primary departure delays surpassing a pre-defined temporal threshold of 30 seconds, which was chosen as it generated a sufficient number of cases for appropriate modeling beyond those attributed to normal departure time fluctuation. This data was then transformed into a suitable 3D tensor for use in complex time series models.

Given the multivariate nature of the problem that consisted of non-consecutive records from different platforms, classical time series methods like ARIMA were deemed unsuitable. Instead, only Machine Learning and Deep Learning models were employed. A Random Forest model served as the benchmark for establishing baseline performance and identifying data features and parameters.

Various models including LSTM-ANN, GRU-ANN and SVM were then trained, evaluated, and compared to the Benchmark with hyperparameter tuning and different experiments carried out to maximize their performance. Notably, delays from previous platforms in the same sequence exhibited a stronger influence in predicting delays for the next platform than inter-train delays from previous time steps on the same platform. This suggests that secondary delays at a platform cannot be accurately predicted solely from the primary delay and preceding train sequences and features from the data available.

Results revealed that the LSTM-ANN hybrid model achieved the lowest RMSE at 11.785. Interestingly, ANN, SVM and Random Forest models, despite not inherently handling temporal data yielded similar performance. This suggested that the temporal dependencies of previous delays at platforms are primarily short-term and have a limited impact. This aligns with the notion that Deep Learning and RNN derivative models are better suited for larger datasets with more intricate temporal dependencies (Brownlee, 2018; Dong et al., 2021). In contrast, conventional Machine Learning models suffice in scenarios where temporal dependencies are less pronounced.

5.2 Future Work

The primary delay threshold for this project was fixed at 30 seconds, however, it can be expanded for predicting more major significant delays given the availability of sufficient data accurately evaluates the model's proficiency in quantifying the subsequent secondary delays.

This project only considered forecasting the immediate secondary delay following a primary delay at a single platform. A more comprehensive model could be extended to provide multi-step predictions that illustrate the extent of the knock-on effect a primary delay has on subsequent trains. This enhancement would allow more effective action to be taken by dispatchers in mitigating secondary delays.

Extending this research’s use case developed for forecasting secondary delays could encompass the forecasting of primary delays as well, leveraging the knowledge of previous platforms and previous time-step delays. By incorporating additional data features, it can become more feasible to develop a classifier model that predicts potential fault causes. This classification would greatly benefit operations and maintenance teams.

Appendix

Project Plan

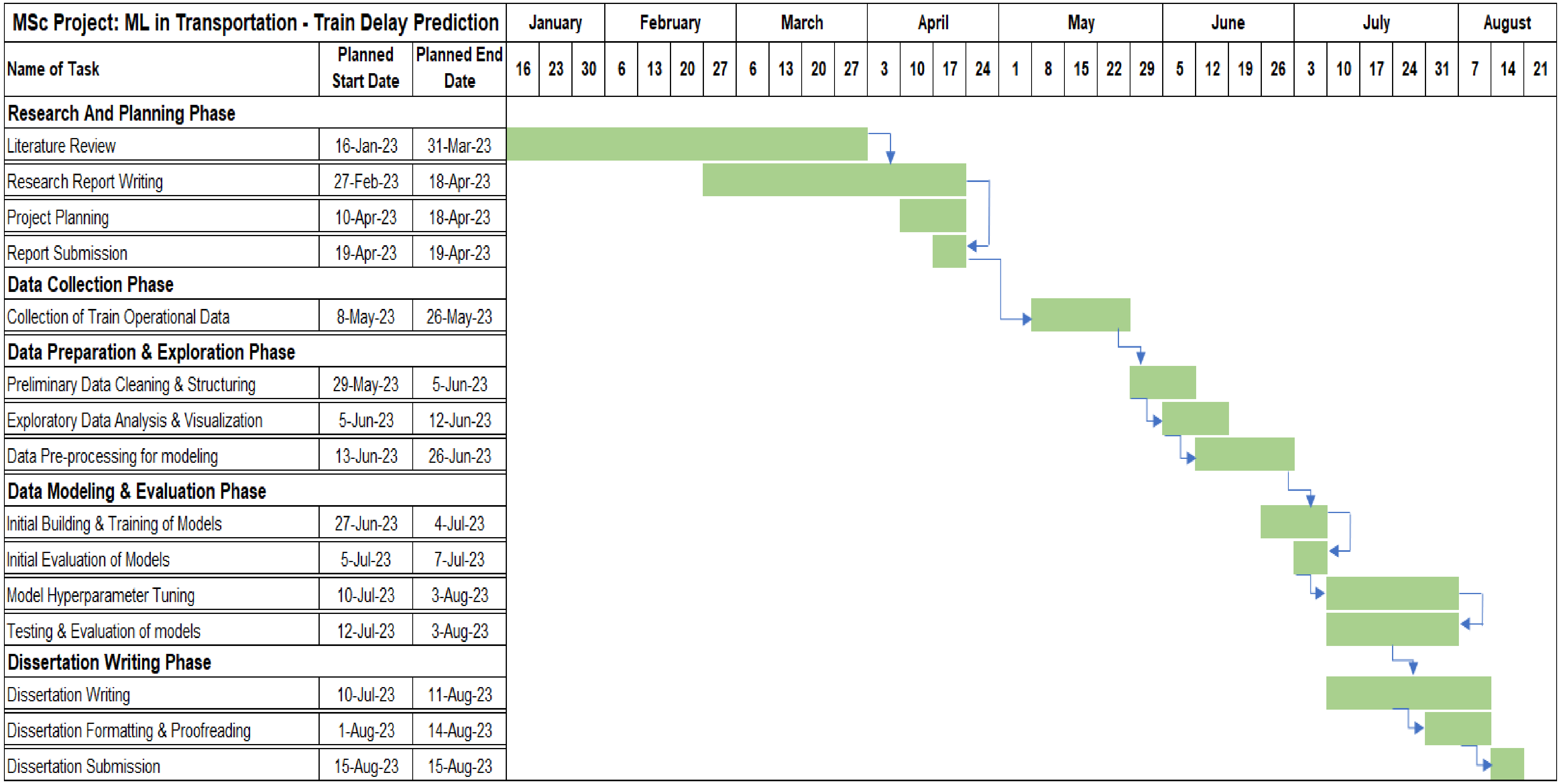
1. Project Schedule

The project has been split into five phases as described below:

1. Research and Planning Phase: This phase involved literature review, project planning and research report writing. It concluded with the submission of the Research report for the project.
2. Data Collection Phase: This phase involved meeting with the RTA to discuss the availability of relevant and usable train operational data. Transfer of data in a usable format for further programming and implementation.
3. Data Preparation and Exploration Phase: This phase involved all the cleaning and pre-processing steps mentioned in the methodology. It also includes carrying out Exploratory Data Analysis.
4. Data Modeling & Evaluation Phase: This phase involved building, training, and experimenting with different DL & ML models. Evaluating their accuracy with the different evaluation metrics mentioned previously. It also includes tuning model hyperparameters and checking their impact on accuracy.
5. Dissertation Writing Phase: This phase involved reporting the experiment set-up and discussing their results in the dissertation. The dissertation will conclude with recommendations for further work based on the experiment results. The final deliverable dissertation will be submitted by the 15th of August.

Fig 27. shows the project Gantt. Chart

Table 21 displays the Project Risk Analysis



MSc Project: ML in Transportation - Train Delay Prediction

Name of Task	Planned Start Date	Planned End Date
Research And Planning Phase		
Literature Review	16-Jan-23	31-Mar-23
Research Report Writing	27-Feb-23	18-Apr-23
Project Planning	10-Apr-23	18-Apr-23
Report Submission	19-Apr-23	19-Apr-23
Data Collection Phase		
Collection of Train Operational Data	8-May-23	26-May-23
Data Preparation & Exploration Phase		
Preliminary Data Cleaning & Structuring	29-May-23	5-Jun-23
Exploratory Data Analysis & Visualization	5-Jun-23	12-Jun-23
Data Pre-processing for modeling	13-Jun-23	26-Jun-23
Data Modeling & Evaluation Phase		
Initial Building & Training of Models	27-Jun-23	4-Jul-23
Initial Evaluation of Models	5-Jul-23	7-Jul-23
Model Hyperparameter Tuning	10-Jul-23	3-Aug-23
Testing & Evaluation of models	12-Jul-23	3-Aug-23
Dissertation Writing Phase		
Dissertation Writing	10-Jul-23	11-Aug-23
Dissertation Formatting & Proofreading	1-Aug-23	14-Aug-23
Dissertation Submission	15-Aug-23	15-Aug-23

Fig. 27: Project Gantt Chart

2. Risk Analysis

Risk	Probability	Impact	Mitigating Action(s)
Dataset from RTA Unavailable or unsuitable for project requirements	Medium	High	• Discuss the possibility of an alternative dataset with supervisor. • Find another public dataset.
Dataset contains extensive amount of missing data or is heavily imbalanced.	Medium	High	• Discuss with RTA to try and obtain the complete or balanced dataset or else find another public dataset.
RTA revokes dataset access	Low	High	• Discuss the possibility of an alternative dataset with supervisor. • Find another public dataset.
Computer Failure and loss of data	Low	High	• Keep the dataset, code, and documentation on OneDrive cloud storage as a backup. • Use version control programs, such as GitHub
Inadequate computing resources	Medium	High	• Look for external cloud-based resources, such as Google Colab.
The project falls behind schedule	High	Medium	• Plan tasks with additional buffer time.
Author Illness or emergency leave	Medium	Medium	• Ask for an extension of the submission deadline if the case is severe.
Supervisor Illness or emergency leave	Medium	Medium	• Use Microsoft Teams as an online meeting tool. • Discuss with the university in a critical case.

Table 21: Project Risk Analysis & Mitigation Plan

References

Ahmed, N.K. et al. (2010) “An empirical comparison of machine learning models for time series forecasting,” Econometric Reviews, 29(5-6), pp. 594–621. Available at: https://doi.org/10.1080/07474938.2010.481556.

Antony, E. *et al.* (2021) “Data preprocessing techniques for handling time series data for Environmental Science Studies,” *International Journal of Engineering Trends and Technology*, 69(5), pp. 196–207. Available at: https://doi.org/10.14445/22315381/ijett-v69i5p227.

Banoula, M. (2023) What is Perceptron: A Beginners Guide for Perceptron, Simplilearn.com. Simplilearn. Available at: https://www.simplilearn.com/tutorials/deep-learning-tutorial/perceptron#:~:text=A%20Perceptron%20is%20a%20neural,value%20%E2%80%9Df(x). (Accessed: March 27, 2023).

Bergmeir, C. and Benítez, J.M. (2012) “On the use of cross-validation for time series predictor evaluation,” *Information Sciences*, 191, pp. 192–213. Available at: https://doi.org/10.1016/j.ins.2011.12.028.

Brownlee, J. (2018) *Deep Learning for Time Series Forecasting Predict the Future with MLPs, CNNs and LSTMs in Python*. Machine Learning Mastery.

Deb, C. *et al.* (2017) “A review on time series forecasting techniques for building energy consumption,” *Renewable and Sustainable Energy Reviews*, 74, pp. 902–924. Available at: https://doi.org/10.1016/j.rser.2017.02.085.

Dong, S., Wang, P. and Abbas, K. (2021) “A survey on Deep Learning and its applications,” Computer Science Review, 40, p. 100379. Available at: https://doi.org/10.1016/j.cosrev.2021.100379.

Frost, J. (2021) *Using moving averages to smooth time series data*, *Statistics By Jim*. Available at: https://statisticsbyjim.com/time-series/moving-averages-smoothing/ (Accessed: March 21, 2023).

Géron Aurélien (2023) *Hands-on machine learning with scikit-learn, keras and tensorflow: Concepts, tools, and techniques to build Intelligent Systems*. Sebastapol, CA: O'Reilly.

Hippke, M. *et al.* (2019) “wōtan: Comprehensive time-series Detrending in python,” *The Astronomical Journal*, 158(4), p. 143. Available at: https://doi.org/10.3847/1538-3881/ab3984.

Huang, P. et al. (2020) “Modeling train operation as sequences: A study of delay prediction with operation and weather data,” Transportation Research Part E: Logistics and Transportation Review, 141, p. 102022. Available at: https://doi.org/10.1016/j.tre.2020.102022.

Hyndman, R.J. and Athanasopoulos, G. (2015) *Forecasting principles and practice: A comprehensive introduction to the latest forecasting methods using R. Learn to improve your forecast accuracy using dozens of real data examples*. 2nd Edition. USA: O texts.

Ismiguzel, I. (2022) *Imputing missing data with simple and advanced techniques*, *Medium*. Towards Data Science. Available at: https://towardsdatascience.com/imputing-missing-data-with-simple-and-advanced-techniques-f5c7b157fb87 (Accessed: March 20, 2023).

Jin, X. et al. (2019) ‘Prediction for time series with CNN and LSTM’, Proceedings of the 11th International Conference on Modelling, Identification and Control (ICMIC2019), pp. 631–641. doi:10.1007/978-981-15-0474-7_59.

Kamarudeen, N., Sundarakani, B. and Manikas, I. (2020) “An assessment of the Dubai Metro Service’s performance using SCOR model and arena simulation,” *FIIB Business Review*, 9(3), pp. 167–180. Available at: https://doi.org/10.1177/2319714520925749.

Lazo, L. (2019) “Amtrak’s chronic delays are costing millions of dollars, report says,” *The Washington Post*, 18 October. Available at: https://www.washingtonpost.com/transportation/2019/10/18/amtraks-chronic-delays-are-costing-millions-dollars-report-says/ (Accessed: February 28, 2023).

Lindemann, B. *et al.* (2021) “A survey on long short-term memory networks for time series prediction,” *Procedia CIRP*, 99, pp. 650–655. Available at: https://doi.org/10.1016/j.procir.2021.03.088.

Luo, J., Huang, P. and Peng, Q. (2022) “A multi-output deep learning model based on Bayesian optimization for sequential train delays prediction,” *International Journal of Rail Transportation*, pp. 1–27. Available at: https://doi.org/10.1080/23248378.2022.2094484.

Marković, N. *et al.* (2015) “Analyzing passenger train arrival delays with support vector regression,” *Transportation Research Part C: Emerging Technologies*, 56, pp. 251–262. Available at: https://doi.org/10.1016/j.trc.2015.04.004.

Nabi, J. (2019) *Recurrent neural networks (RNNs), Medium*. Towards Data Science. Available at: https://towardsdatascience.com/recurrent-neural-networks-rnns-3f06d7653a85 (Accessed: April 1, 2023).

Nabian, M.A., Alemazkoor, N. and Meidani, H. (2019) ‘Predicting near-term train schedule performance and delay using bi-level random forests’, Transportation Research Record: Journal of the Transportation Research Board, 2673(5), pp. 564–573. doi:10.1177/0361198119840339.

Norwawi, N.M. (2021) “Sliding window time series forecasting with multilayer perceptron and multiregression of covid-19 outbreak in Malaysia,” Data Science for COVID-19, pp. 547–564. Available at: https://doi.org/10.1016/b978-0-12-824536-1.00025-3.

Park, J. *et al.* (2022) “Long-term missing value imputation for time series data using Deep Neural Networks,” *Neural Computing and Applications* [Preprint]. Available at: https://doi.org/10.1007/s00521-022-08165-6.

Patrikar, S. (2019) *Batch, Mini Batch & stochastic gradient descent, Medium*. Towards Data Science. Available at: https://towardsdatascience.com/batch-mini-batch-stochastic-gradient-descent-7a62ecba642a (Accessed: March 28, 2023).

Rail Transport Global Market Report 2023– By Type (Passenger Rail Transport, Rail Freight), By Distance (Long-Distance, Short-Distance), By Destination (Domestic, International), By End-Use Industry (Mining, Construction, Agriculture, Other End Use Industries) – Market Size, Trends, And Global Forecast 2023-2032 (2023) *The Business Research Company*. Available at: https://www.thebusinessresearchcompany.com/report/rail-transport-global-market-report (Accessed: February 28, 2023).

Saha, S. (2022) *A comprehensive guide to Convolutional Neural Networks - the eli5 way, Medium*. Towards Data Science. Available at: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (Accessed: March 31, 2023).

Spanninger, T. *et al.* (2022) “A review of train delay prediction approaches,” *Journal of Rail Transport Planning & Management*, 22, pp. 1–17. Available at: https://doi.org/10.1016/j.jrtpm.2022.100312.

Tang, R. *et al.* (2022) “A literature review of artificial intelligence applications in Railway Systems,” *Transportation Research Part C: Emerging Technologies*, 140, pp. 1–10. Available at: https://doi.org/10.1016/j.trc.2022.103679.

TATSUI, D. *et al.* (2021) “Prediction Method of Train Delay Using Deep Learning Technique,” *Quarterly Report of RTRI*, 62(4), pp. 263–268. Available at: https://doi.org/10.2219/rtriqr.62.4_263.

Tiong, K.Y., Ma, Z. and Palmqvist, C.-W. (2023) “A review of data-driven approaches to predict train delays,” *Transportation Research Part C: Emerging Technologies*, 148, pp. 1–18. Available at: https://doi.org/10.1016/j.trc.2023.104027.

Wen, C. *et al.* (2019) “A predictive model of train delays on a railway line,” *Journal of Forecasting*, 39(3), pp. 470–488. Available at: https://doi.org/10.1002/for.2639.

Wu, J. *et al.* (2021) “A hybrid LSTM-CPS approach for long-term prediction of train delays in multivariate time series,” *Future Transportation*, 1(3), pp. 765–776. Available at: https://doi.org/10.3390/futuretransp1030042.

ZHOU, Z.H.I.-H.U.A. (2022) “Introduction,” in *Machine learning*. S.l.: SPRINGER VERLAG, SINGAPOR, pp. 1–1.

Ziganto, D. (2018) *Introduction to time series, https://dziganto.github.io/*. GitHub Pages. Available at: https://dziganto.github.io/python/time%20series/Introduction-to-Time-Series/ (Accessed: March 20, 2023).