CHALMERS
UNIVERSITY OF TECHNOLOGY

Applications for Data Analytics using JRU Logs from Autonomous Trains

Improving train operations by facilitating data analysis & predicting malfunction using anomaly detection

Master's thesis in Master Programme Computer Science - Algorithms, language and logic

DAVID LAESSKER & VILHELM HEDQUIST

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's thesis 2025

Applications for Data Analytics using JRU Logs from Autonomous Trains

Improving train operations by facilitating data analysis & predicting malfunction using anomaly detection

DAVID LAESSKER
VILHELM HEDQUIST

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2025

Applications for Data Analytics using JRU Logs from Autonomous Trains
Improving train operations by facilitating data analysis & predicting malfunction using anomaly detection
DAVID LAESSKER
VILHELM HEDQUIST

Supervisor: Nir Piterman, Department of Computer Science and Engineering
Examiner: Wolfgang Ahrendt, Department of Computer Science and Engineering

Master's Thesis 2025
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: An Alstom operated train in the LKAB iron ore mine in Kiruna.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2025

Applications for Data Analytics using JRU Logs from Autonomous Trains
Improving train operations by facilitating data analysis & predicting malfunction using anomaly detection
DAVID LAESSKER
VILHELM HEDQUIST
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

In Alstom's autonomous railway systems used for mining operations, vast amounts of operational data are recorded by onboard Juridical Recording Units (JRUs). However, these logs are complex and difficult to interpret. This thesis addresses the challenge of parsing and structuring JRU log data to enhance its readability and enable advanced data analysis. A custom log parser was developed to convert raw logs into a structured, readable format. To explore the potential of this data for predictive analysis, an LSTM Autoencoder neural network was trained for anomaly detection based on temporal patterns. The results demonstrate the feasibility of using machine learning for operational insights, and suggest promising future applications in automated fault detection and predictive maintenance.

Keywords: railway, trains, JRU, machine learning, LSTM, autoencoder, anomaly detection, logs, data analysis, predictive maintenance.

Acknowledgements

We would like to thank our supervisors at Alstom, Pierre-Louis Bullot and Nicola Bottini, as well as our colleagues Linus Wass and Jakob Bjerke for the great guidance in the project when needed and willingness to accommodate for the change of main thesis topic. Secondly, our supervisor Nir Piterman and examiner Wolfgang Ahrendt gave us valuable insights into how to approach our problems with a research mindset. Finally we want to thank everyone else at Alstom who gave us advice and helped us understand the complex train systems and applications, as well as family and friends who supported us along the way.

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AI	Artificial Intelligence
BPTT	Backpropagation Through Time
CNN	Convolutional Neural Network
CSV	Comma Separeted Value
CTC	Centralized Traffic Control
JRU	Juridical Recording Unit
LLM	Large Language Model
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
ML	Machine Learning
RNN	Recurrent Neural Network

Contents

List of Figures

1

Introduction

Efficient train transport relies on all railway equipment functioning properly as often as possible, and as with all complex, moving systems incidents are inevitable. When an incident event takes place it is in everyone's best interest to as accurately as possible determine the cause in order to prevent the same kind of event from happening again. By doing this the amount of possible incidents will diminish over time, and although a totally incident-proof system is not realistic, it can and should be the goal to strive for in the railroad industry.

Alstom is responsible for many train systems operating all over the world, from public trains and trams in cities to large mining operations in rural Australia as well as the largest mine in Sweden located in Kiruna. During operation, these trains record and store a lot of operational data onto an onboard Juridical Recording Unit (JRU), which functions similarly to the black box on an airplane. In the event of an accident, this data needs to be readily available to be interpreted by error investigators to be able to recreate the error and figure out what went wrong. Currently, post-accident/malfunction analysis is done completely manually by experts which costs a lot of time and money, and because this data is very unintuitive it becomes a difficult task to reach the proper conclusion as to what caused the incident.

1.1 Background

Some research which discusses a similar topic in the same industry has been done previously, and this paper seeks to learn from and build upon this.

In order to be able to further expand post train crash analysis, the Federal Railroad Administration in the United States of America started requiring additional data collection from "black boxes" starting in 2011. In the press release leading up to this [4] FRA Administrator Joseph H. Boardman said "*We are making sure that investigators have more and better information available when working to find the cause of an accident. The more we can learn from train accidents, the more we can prevent them from occurring.*"

In the paper "Characteristics of event recorders in automatic train control systems" [5] M. Jacyna et al. writes about the future of automatic train operation. By using automatic train control systems more trains can be ran at a lower cost while

simultaneously lowering the risks of accidents. It is highlighted how, in order to implement such a system, collecting and analysing data parameters of trains under operation is critical to be able to tune the system and get satisfactory results. Since there is no real standard or legal definition of what characteristics a JRU requires the data type and quality can vary by a large amount, making proper data analysis even more important.

I. Naish writes in a paper for the International Railway Safety Conference [6] about the importance of collecting data for railway crash investigations. He points out that *"In the event of a very serious accident, investigators, regulatory bodies and industry might want to ask themselves: could this horrible event have been anticipated? Good data resources and good analysis of the data will always help to answer this question."* While this is true, there is no standardized tool that facilitates this post-accident or malfunction data analysis. As important as the data itself is, it becomes meaningless if we cannot interpret it in a clear and understandable way.

There exists quite a few studies on anomaly detection in logs, a few of which are compared in a study by V.-H. Le and H. Zhang [7]. Many of these studies focus on logs taken from digital systems such as online services. While the logs taken from these systems are quite large and contain a substantial amount of data, they differ greatly from the logs handled in this project due to the fact that no moving parts are involved. The JRU logs involved are quite complex and need to accomodate a wide variety of interface specifications and error message formats for various trains and train control systems. The area closest to railway concerning this topic seems to be anomaly detection in flight data recorders, as observed in these papers by C.-H. Lee et al. [8] or S. Das et al. [9]. These logs do handle large vehicles, but they are quite a bit more standardised and held to a higher regard of safety. This thesis aims to apply similar strategies to railway logs, as to the best of our knowledge there exists quite a small amount of insight into the area.

1.2 Scope and Objectives

The general goal of this thesis work is to make use of the JRU log data to enhance safety and reliability of train operations as well as post incident analysis. Any progress with handling the data in a more convenient way than it already is can be seen as a success, as the current log data reading and evaluation process is long and cumbersome with no real organization or readability of the data. In order to do this a thorough analysis of the raw data was first conducted to identify the structure and determine what parts of the data would be interesting to study. This made it possible to later convert it into a more succinct format that can be analysed by a computer and a human.

When the data is sorted out and stored in a format which is more usable, a machine learning model can be trained on the data in order to extract more interesting information from it. After some consideration, the data analysis method chosen for this paper was anomaly detection. This was due to the fact that similar research has been made in adjacent fields, alongside there being multiple interesting use cases for

further research or system implementations (discussed in chapter 5).

As such, the **primary goal** of this thesis is to **facilitate data analysis of JRU logs to help prevent accidents or malfunctions**. This will be accomplished by:

Thesis Objectives

- Processing the data to store it in a readable format
- Analysing the processed data using machine learning
- Exploring possible applications for the extracted information

1.3 Delimitations

When taking on a project like this it can naturally be hard to predict what exactly can be completed in the allocated time frame. This project depends heavily on what the analysed data looks like and what information can be extracted from it. Therefore certain limitations were set on the project which served to contain it within a reasonable scope.

1.3.1 Complete Anomaly detection product out of scope

As the project is both time limited and heavily dependent on the type of data provided, it is most likely too ambitious to promise a full product that relies on advanced data analysis. Regardless, the steps taken in this process of analysing and processing the train log data will provide grounds and guidelines for further research and implementation. The work will also be relevant for insight into future products in areas handling large data logs with uncategorised data.

1.3.2 Integration with already existing infrastructure

Due to working closely with a company, the any developed project should ideally work well with their already established system architecture. If this turned out to be too complicated there was a consideration to be made when deciding how much time should be put into making it work with the system and when it should be re-prioritised to out the time into a more research oriented focus instead. This ultimately made the final result limited to a more investigative role than initially planned and there is little to no integration with Alstom's systems due to coupling difficulties.

2

Theory

This chapter serves as a brief description of concepts the reader should have an at least rudimentary understanding of to follow along in the contents of the thesis.

2.1 Log files

Most of the work of this thesis is based on data from the logs created by the Juridical Recording Units onboard every train in Alstom's rail network. The purpose of a Juridical Recording Unit (JRU) is to record and document all messages sent to and from the train in a log file. In the event of an incident these log files can be examined to determine the state of the rail network which caused the event. The "juridical" in the name is due to the fact that these logs are juridically binding in the event that any blame for an accident must be put on an involved party, such as the system distributor or the end user.

The actual log files available for use in this thesis are combined logs put together by the central computer system, which contain an enormous amount of data regarding every train, set of points, derailer, etc. present in the entire rail network. This results in very large log files which are nigh impossible to understand "raw", as much of the data is formatted according to many different protocols on top of handling such a large amount of data. Additionally, Alstom has railway networks installed on 4 different continents so far with wildly varying information protocols and network sizes which adds further complexity to comprehending the logs. As such, a lot of work has gone into parsing and pre-processing these logs in order to be able to extract any valuable data from them. Listing 2.1 shows an example of what one of these log files may look like.

```
Timestamp, Protocol, Version, Message
10:00:01, EULYNX_V1, 1.2, <TrainData><ID>T42</ID><SpeedKmh>85 ...
10:00:02, PCL, 3.0,  {'type':'point_sts', 'id':'P101A' ...
10:00:03, RAW_TRK_IO, 0.9,  02 A5 FF 00 1B C3 DE F0 50 04 E0 ...
10:00:04, LEGACY_MIX, 1.0,  ##HDR##<Derailer status="ACTIVE" ...
10:00:05, CUSTOM_V_2_1, 2.1, STX;ID=T88;SPD=076.5;WGT=0987.2 ...
10:00:06, BIN_ENCASED_XML, 1.5, 0x02 0x00 0x3C 0x3F <msg>OBC ...
10:00:07, TEXT_CSV_LIKE, 1.0, TIMESTAMP_MS:1698380407123 ...
10:00:08, PCL, 3.1,  {'update': {'target':'brake_pressure' ...
10:00:09, EULYNX_V1, 1.2,   <TrackCondition TSR_ID="R004" ...
10:00:10, RAW_MSG_BYTECODE, 2.0, F0 A1 03 B2 C3 D4 E5 00 FF ...
```

Listing 2.1: Example log for illustrative purposes. Does not contrain real data or follow specific structures used at ALSTOM

2.2 Machine learning

After the log files have been parsed and reduced to keep only the relevant information, it is possible to extract useful information from them. This is done by further analysing using machine learning, in this case in the specific application of anomaly detection. The basics of anomaly detection, as well as some relevant machine learning models, will be described in this section.

2.2.1 Anomaly detection

Anomaly detection, also referred to as outlier detection, is a widely used technique with a lot of applications in the field of data analysis as described by M. Gupta et al. [10]. Some of the more common use cases are financial market prediction, network intrusion detection systems, or, as in this case, system diagnosis. The idea is to train a machine learning model to recreate temporal data. When given a certain period's worth of data to analyse the model will attempt to recreate it as accurately as possible, and areas where it differs by more than a given margin will be marked as anomalies. Since the model is trained to be able to recreate such data, a big difference indicates a possible deviation from how the data patterns usually look.

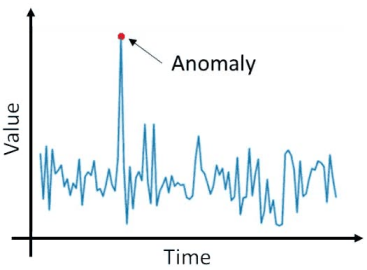

Figure 2.1: Simple time series anomaly example [1]

Training a model on anomaly detection is done either supervised or unsupervised. Supervised training relies on availability of training data that is clearly labelled as normal or anomalous. This kind of training is generally more efficient due to this labelled data, however it does have some shortcomings. Such data is generally hard to come by, as it has to be sorted or created specifically to be used as supervised training data. If the nature of the anomalies changes over time, it can also lead to a high rate of false positives.

Contrarily, unsupervised anomaly detection is achieved by training the model on unlabelled data, both normal and anomalous. This kind of data is often available in abundant quantities and can also train the model to detect some previously unseen anomalies, making it more flexible. However the training itself is more intensive on time and resources.

2.2.2 Neural networks

Neural networks are a class of machine learning models inspired by the human brain, consisting of layers of interconnected nodes that process data through weighted connections and activation functions. They typically include an input layer, one or more hidden layers, and an output layer. Each layer transforms the data in a way that helps the network learn useful representations.

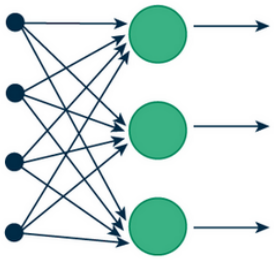

Figure 2.2: Feed forward neural network [2]

Different types of neural networks are used depending on the structure and nature of the data. Convolutional Neural Networks (CNNs) are often used for image and spatial data, while Recurrent Neural Networks (RNNs) and their variants are suited for sequential data due to their ability to retain information across time steps.

The architecture of a neural network can be adjusted to fit specific purposes by varying the number of layers, the number of nodes per layer, and the way layers are connected. For example, deep networks with many hidden layers can learn more complex patterns, while networks with specialized layers such as attention mechanisms or memory cells can be tailored for tasks requiring temporal or contextual understanding.

In the context of anomaly detection, neural networks are typically trained to model the normal behaviour of a system. Once trained, the network attempts to reconstruct or classify new data, and deviations from the expected pattern can be flagged as anomalies. [11]

In order to achieve temporal based anomaly detection using a neural network it needs to be capable of remembering patterns. The models used were chosen due to the fact that they have been used in similar applications in different industries, as studied by for example C.-H. Lee et al. [8] and S. Das et al. [9].

2.2.2.1 Recurrent Neural Networks - RNN

A Recurrent Neural Network (RNN) is, as R. M. Schmidt puts it, "a type of neural network architecture which is mainly used to detect patterns in a sequence of data" [12].

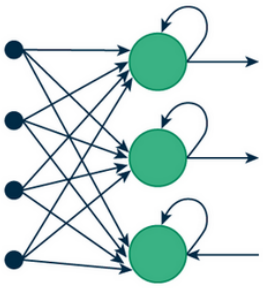

Figure 2.3: Recurrent neural network [2]

These neural networks expand on the functionality of a feedforward neural network by having a recurrent step which feeds information back into itself. This means that when taking the next step in the calculation, they consider not only the current input X_t but also previous ones $X_{0:t-1}$. This is done by using Backpropagation Through Time (BPTT), as compared to the backpropagation commonly used to train feedforward neural networks.

While effective, RNNs suffer from vanishing and exploding gradients. Because the same values are used repeatedly when training, very small (<1) or very large (>1) values multiplied with themselves quickly results in the gradients shrinking or growing more and more with each layer before vanishing completely or exploding into enormous values. This can make the result largely unusable. In order to fix this, Long Short-Term Memory units were introduced.

2.2.2.2 Long Short-Term Memory - LSTM

Long Short-Term Memory (LSTM) is an evolution of the RNN designed to be more resilient to the problem of vanishing or exploding gradients. It accomplishes this by storing information outside of a traditional network structure in gated cells. These cells enable storing of certain information over long periods of time, while discarding pieces of information which are deemed irrelevant.

An LSTM cell has three gates; an output gate O_t to read the cells entries, an input gate I_t to read new data into the cell, and a forget gate F_t to forget the data stored in the cell. The computations for these cells are as follows:

$$O_t = \sigma(X_t W_{xo} + H_{t-1} W_{ho} + b_o) \tag{2.1}$$

$$I_t = \sigma(X_t W_{xi} + H_{t-1} W_{hi} + b_i) \tag{2.2}$$

$$F_t = \sigma(X_t W_{xf} + H_{t-1} W_{hf} + b_f) \tag{2.3}$$

In these equations, W are weight matrices, b are their biases, X_t is the current input value, and H is the hidden layer values. σ is the sigmoid activation function used to transform the outputs to values between 1 and 0. A deeper explanation of this, along with a visual explanation, can be found by reading "Recurrent Neural Networks (RNNs): A gentle Introduction and Overview" by R. M. Schmidt [12].

LSTMs, while more effective than RNNs, are more complicated to understand and implement as well as requiring more computational power to train.

2.2.2.3 Autoencoder

An autoencoder is a type of neural network architecture typically used when there is a need for analysis of temporal data, and the model training is to be unsupervised. O. I. Provotar et al. describes it as "one of the best machine learning methods" when analysing time series which do not have a clear period [13]. It is composed of two parts, an encoder and a decoder. The encoder uses neural network layers to reduce the input data to its underlying features, by training it to ignore "noise" and parts of the data that are deemed irrelevant. The decoder then uses more neural network layers to reconstruct the data from these underlying features. These network layers can be based on a feedforward neural network, but as the temporal aspect of the data is crucial in this application LSTM layers are used to construct the autoencoder. An visual example of an autoencoder's structure can be seen in figure 2.4.

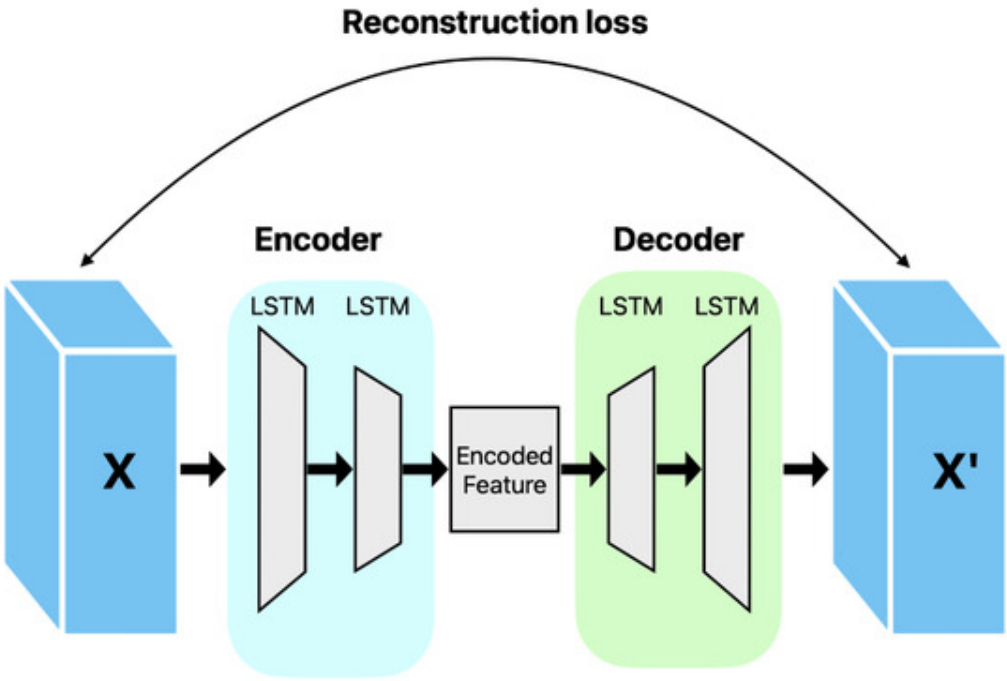

Figure 2.4: Visualisation of the layers of an LSTM autoencoder from [3], licensed under CC BY 4.0.

For the purpose of detecting anomalies in complex log files, the LSTM based autoencoder architecture is a good choice. The major advantage is that it learns without the need for labelled data (unsupervised learning). Given that anomalies in these extensive logs are often rare, unlabelled, and occur in unseen ways, this approach enables learning patterns of "normal" data without human intervention or labelling. An autoencoder excels at this by learning to compress (encode) normal log sequences into a compact latent representation and then accurately reconstruct (decode) them.

When an anomalous sequence occurs, the autoencoder will struggle to reconstruct it, resulting in a high *reconstruction error* which is interpreted as an anomaly.

2.3 Replaying log data

One part of the work is to develop a log replayer tool. This will be an internal software application used to re-send a JRU log file to the internal Centralized Traffic Control (CTC) system, which is what represents the system data to a user through a GUI. Such a tool would be of great aid in the process of incident investigation, where it would be possible to review an event multiple times in order to precisely figure out the state of the system which caused the incident to occur.

A big challenge when developing this tool is the large size of the log files generated by the system. Since a given log entry often contains a single piece of information it is difficult to know how far back in the log to look in order to figure out the exact state of the system. Naïvely, in order to get accurate information the whole log would need to be iterated through from the beginning every time a rewind was to be performed. This is not really a feasible solution, as this would require the tool to keep the entire log file in memory while also becoming incredibly slow.

3

Methods

This chapter outlines the methodology used to process and analyse raw log data for anomaly detection and replay functionality. The system is designed to be modular, scalable, and adaptive to various logging formats and hardware capabilities. Initially the overall system architecture is presented, followed by detailed descriptions of the preprocessing pipeline; including adaptive parsing, preparation for log replay, and formatting for machine learning models. Finally, a description of the implementation of an LSTM-based autoencoder used for time series anomaly detection.

3.1 System Architecture

In order to be able to analyse the logs they first have to be interpretable to both the human eye and and a machine. In their raw state they are neither. Therefore the first step in the process is to parse them and store them in an interpretable format. In order to create a system that is efficient and adaptive the underlying architecture needs to be carefully planned.

Firstly, the system needs to be modular and adaptive. As mentioned in section 2.1 the logs are complex and diverse. The system needs to be able to handle many different types of logging which are all parsed differently, all in one large log file. Secondly, system performance is to be considered. Given the large size of many log files, a naïve approach to parsing will lead to very long processing times and it is likely that the computer runs out of memory. Finally, there is an advantage to adapting the parsing to different purposes. If the specific purpose is known before parsing, certain protocols that do not contain any information relevant to the purpose can be discarded in advance.

Given these considerations, the structure pictured in figure 3.1 was implemented. The system consists of two Python modules. Firstly, the *General Parser* which takes care of parsing the log files and imports parsing specifications for different protocols/subsystems. Secondly the *Object Handler* which takes care of storing information regarding specific objects such as trains or sets of points. This can extract relevant information for machine learning applications such as *Anomaly Detection* and locating object specific log messages in order to replay logs. Finally there is a *Replayer* application written in Java with the main purpose of replaying log events to the company's internal system, enabling visual retrospective analysis of the logs.

The parsing and machine learning applications were written in Python due to the scripting heavy nature of the operations, availability of relevant libraries, and personal familiarity. The replayer was required to be written in Java due to compatibility requirements of already existing systems at Alstom. An in depth description of the specific parts can be found in section 3.2.

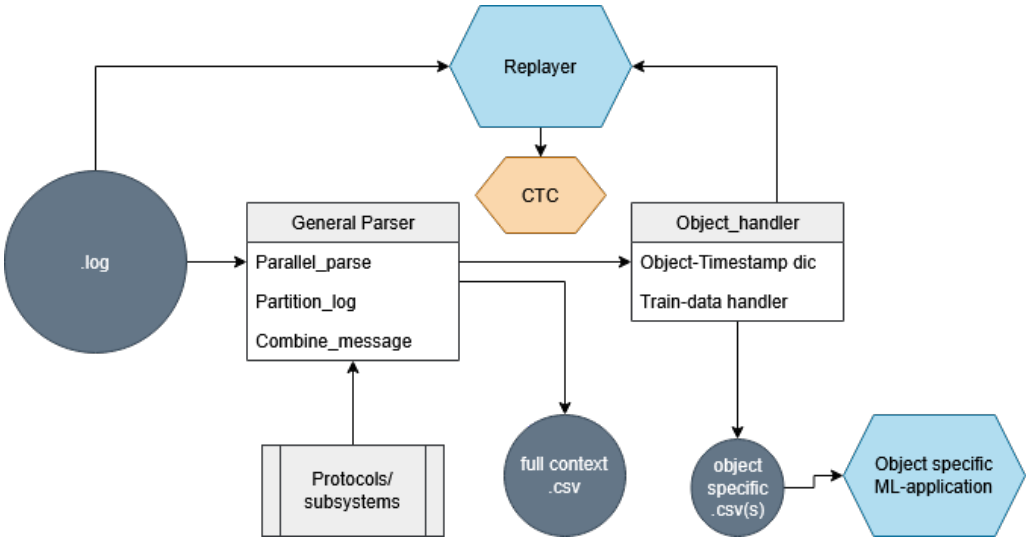

Figure 3.1: Program Structure Overview (object means any type of object on the train tracks such as: trains, derailers & set of points).

3.2 Data Preprocessing

This section will describe the preprocessing of the data on a deeper level. Not everything will be covered, but some things are important to highlight since they involve significant challenges that were needed to be overcome, and anyone doing this in the future will likely encounter similar challenges.

3.2.1 Adaptive Parsing

In order to be able to interpret the logs the parsing needs to be adaptive and dynamic. The first step in this process is combining the log message structure. Logs contain both long and short message formats, but for the purpose of the future analysis the messages are normalised into a short format. This simplifies the structure for parsing and makes it possible to efficiently process logs in parallel.

Once the entries are combined, the parsing is optimised using parallelism by dividing the log entries across the available CPU threads. Because each log entry is now self-contained and independent, parallel parsing becomes feasible without introducing data dependency issues. However it is important that the logs are still ordered post parsing, so care is taken to make sure that the order of elements is not disrupted by the parallelisation.

Each log entry includes a header that specifies its protocol and subsystem. This information allows the system to dynamically call the corresponding protocol-specific parser. The design is modular, each protocol is implemented as a separate class with its own parsing logic. This makes the system highly extensible, future developers can easily integrate support for new protocols or protocol versions by simply implementing a new parser class and place it in the folder of parsing classes. If the correct protocol and subsystem is defined within the class, the classes parsing function will automatically be called on the message part of the log entry when parsed.

Depending on the parsing objective specific protocols or subsystems that are not relevant can be ignored, significantly reducing processing time and memory usage when only a subset of the data is required. This workflow can be observed in figure 3.2.

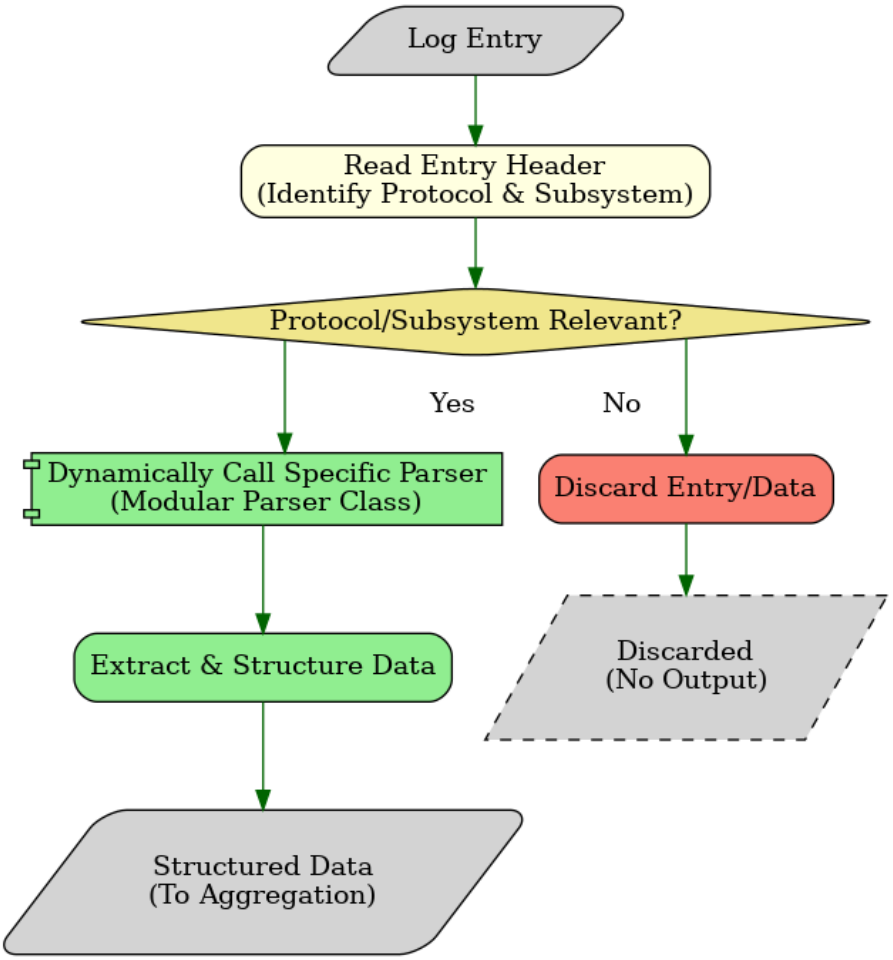

Figure 3.2: Description on how each log entry is treated. A larger overview can be seen in Appendix A.

Because log files of this nature are often quite large, the system supports optional partitioning of large log files. When enabled, the parser divides the input log into smaller segments, each of which can be parsed independently. This is particularly useful for large logs that may not fit into memory or would otherwise cause perfor-

mance bottlenecks. Partitioning can be toggled on or off.

With these implementations the system is adaptive to computers of different performance levels, logs of different sizes, and it will be compatible with new systems after minimal development time implementing a new protocol. An overview of the system can be seen in figure A.1.

3.2.2 Preparing for replay

To enable replay functionality, a separate Java-based replay application is necessary. The naïve approach to replaying logs would be to sequentially stream all log entries to the frontend. However, this method lacks flexibility, particularly the ability to jump backwards in time and would require restarting from the beginning to revisit earlier states. To solve this, a solution that aims to enable replay and reduce computational load was constructed.

During the parsing phase, a dictionary is constructed that maps each unique object in the system (e.g., trains, derailers, or sets of points) to a chronological list of references. Each entry in this list is a tuple containing the timestamp at which the object was mentioned and the corresponding row number in the log. This data structure remains lightweight, as it only stores metadata rather than full log entries, yet it provides a powerful indexing mechanism for replay.

This dictionary enables efficient lookup operations during replay. When the user initiates a rewind or jump to a specific timestamp, the replayer queries the dictionary to determine the most recent state of each object at that point in time. For each object, a binary search is performed on the list of timestamps to locate the nearest previous entry. The corresponding row number can then be used to fetch the relevant log entry directly from the original file, without re-parsing.

This architecture minimizes the computational load during replay and ensures that the frontend receives a consistent and up-to-date snapshot of the system's state at any given timestamp. By offloading indexing to the parsing stage the replay experience becomes significantly more responsive and scalable. A summary of this architecture can be seen in figure 3.3.

3.3 Time Series Anomaly Detection

Detecting anomalies in time series data is a well established approach for identifying unexpected patterns or behaviours that deviate normal behaviour. Normally this would be difficult to perform accurately on driver operated trains, as different drivers would exhibit different driving patters. This would make it very difficult for the model to learn consistent patters and identify when the behaviour deviates from that. Instead, the analysis is performed on driverless trains where each train is being controlled by a central programming system. In this case all trains should operate and behave in the same general way under the same circumstances which makes the identification of abnormal behaviour more reliable, as deviations are more likely

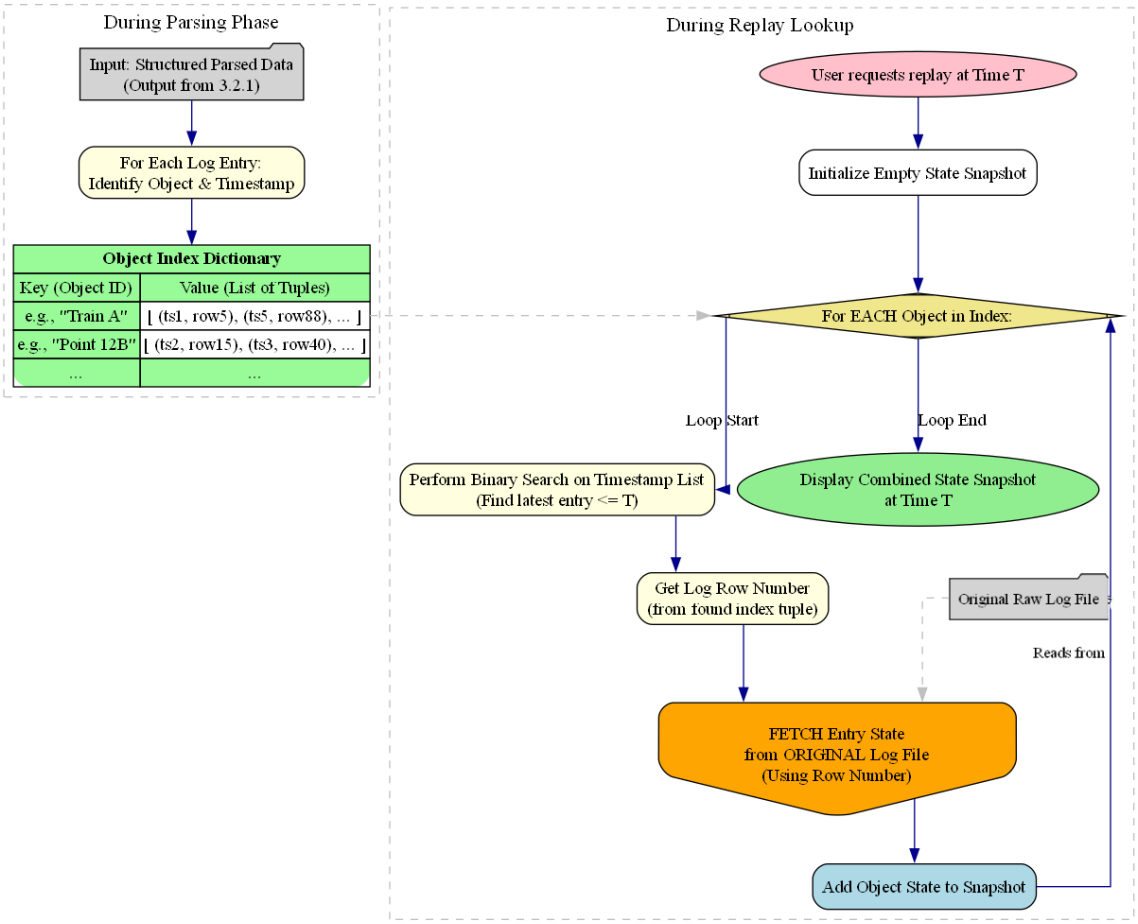

Figure 3.3: Parsing for replay description

to reflect true anomalies rather than individual driving styles or human-induced variation. Indicated anomalies should not always be immediately accepted as such, especially when dealing with a complex task like this. This section describes the decisions made to ensure that the findings are as reliable as possible while taking into account the inherent challenges.

3.3.1 Why anomaly detection fits

Anomaly detection can be very useful when dealing with the vast and complex datasets generated by systems like freight trains. Raw log files, as detailed in section 2.1, can be overwhelming to analyse, making manual identification of unusual or problematic behaviours exceedingly difficult. The main reason for anomaly detection in this context is to automate the discovery of instances where the system's behaviour deviates significantly from the norm, which could indicate emerging failure.

As mentioned above, determining a "normal" behaviour is difficult on driver-operated trains. Therefore this study focuses on data from driverless trains controlled by a central scheduling system, ensuring a rather high degree of consistent behaviour under similar conditions. This inherent predictability makes anomaly detection particularly potent. Deviations from the learned norm are more likely to signify

genuine operational irregularities or system anomalies. Still, the results of the model should not be blindly accepted without further insight.

By training a model to learn the typical operational patterns from historical data, it can then be used to scrutinize new or existing data for sequences that it struggles to reconstruct or recognize. High reconstruction error events should trigger flags in the system, drawing attention to segments of data that could warrant further investigation. A few strategies to interpret the resulting data and examples of respective use cases is discussed in sections 5.2.1 and 5.2.4.

3.3.2 Preparing Data for Machine Learning

Once the data is preprocessed from a log file into a .csv file preparation for machine learning purposes can begin. The first step is to extract train-specific data, if this has not already been performed during parsing for this specific purpose as described in subsection 3.2.1. Now, all of the logged data entries that mention that specific train are available and can be further adapted for machine learning purposes.

3.3.2.1 Data Aggregation

Data categories such as speed and weight must be further analysed before being fed into a model. Naïvely, one can simply extract the column from the train specific .csv file and provide it to the model. This may work in cases where data is logged frequently and regularly, however when working with diverse logs and industrial systems like this one should not assume that things are logged in a systematic way. As such the data given to the model needs to be handled with consideration.

In the case of the particular logs looked at in this example, the metric of speed was logged both sporadically and regularly. Most of the time, the speed was logged five times within a few milliseconds every three seconds. If this data were to be taken to the model sequentially, it would not represent how the speed metric itself changes over time, as the small millisecond steps will be interpreted as equally significant to the gap of three seconds. In addition, other irregularities occurred such as gaps of ten seconds of logging, and sometimes even more frequent logging than every three seconds. An illustrative example of this logging can be seen in figure 3.4.

In order to avoid this issue, the data was sampled every five seconds with the mean recorded across the five second window. This solution avoids the problem of irregular logging, as the data points represent actual change over time instead of change per logging occurrence. This is very important for a model that has the purpose of learning behaviours over time. It can now be guaranteed that the sequentiality of the data represents the behaviour of the data points over time, while not being influenced by the sporadic logging frequency. These new temporal data points can be observed in figure 3.5. It is however worth noting that this solution does not avoid sequences of time where no metric is being logged at all. There is no valid fix for this, since there simply does not exist any data for this period. In this particular solution, these empty periods are simply being filled with the previous data point. The ideal solution to this issue would simply be more frequent logging by the system

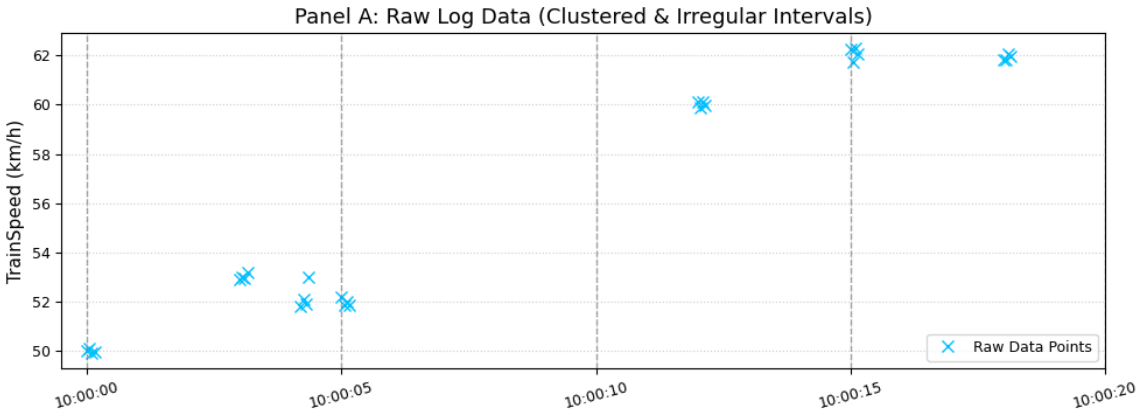

Figure 3.4: Sporadic logging by train JRU.

in question.

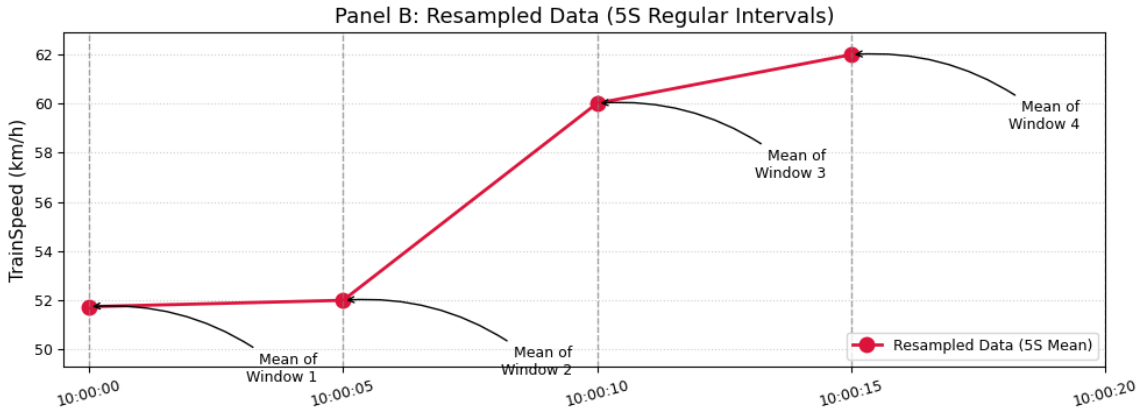

Figure 3.5: Resampling every five seconds to avoid sporadic logging. Source data can be seen in figure 3.4.

3.3.2.2 Numerical Scaling

The `StandardScaler` standardizes features by removing the mean and scaling to unit variance. The standard score of a sample x is calculated as:

$$z = \frac{(x - \mu)}{\sigma}$$

where:

- z is the standardized value (the output of the scaler).
- x is the original feature value.
- μ (mu) is the mean of the feature values in the training set.
- σ (sigma) is the standard deviation of the feature values in the training set.

This transformation centres the data around zero (mean of zero) and scales it so that the distribution has a standard deviation of one.

Consider the primary metrics of the example: train speed and weight.

- **Speed** might range, for example, from 0 km/h to 120 km/h.
- **Weight** might range from 500 tonnes to 5000 tonnes.

If these features are fed directly into a model the feature with the larger absolute values (weight, in this case) could dominate the learning process or the calculation of error metrics, not because it is inherently more important, but simply due to its scale. In addition, data points are not always stored in a specific format such as km/h. In one observed case, speed was stored as a byte value from 0-256. Standardisation will ensure that no matter the numerical standard of the numbers, each feature will contribute more equally to the distance computations and gradient updates, in turn preventing features with larger magnitudes from overshadowing others. It also helps algorithms that assume data is centred around zero, and can make the optimization landscape smoother for gradient descent.

3.3.2.3 Data Splitting For Specific Purpose

The strategy for splitting data into training and validation sets is particularly critical in this context for several key reasons. Firstly, since the data is temporal and sequence dependent, an arbitrary split or shuffling of the data points cannot be made without breaking that structure. Secondly, a consideration has to be made as to for what purpose the model is trained. This raises the question: how specific should the model be? For this selection three key factors can be highlighted:

- **Specific Track Segments:** Isolating data related to particular stretches of track allows for models sensitive to highly localized conditions or known problematic areas.
- **Specific Train Identifiers:** Partitioning by individual train units enables the creation of models that learn a single train's unique operational signature or compare it against fleet behaviour.
- **General Train Regions:** Grouping data by broader geographical areas helps in developing models that capture regional operational norms and anomalies, abstracting away from individual track specifics.

The selection or combination of these dimensions directly influences how training and validation datasets are constructed. Therefore, the specific goal in mind must be considered. Different combinations will have different use cases and fit different scenarios. In table 3.1 a few of these combinations can be observed with their described purpose.

Table 3.1: Illustrative Data Splitting Strategies Based on Model Purpose and Focus

Model Purpose Focus (Clarification)	**Potential Data Splitting Implication**
General train behaviour (fleet-wide, cross-track)	Train on data from a diverse set of trains and various track segments. Validate on a hold-out set of different trains and/or entirely unseen track segments to test generalization.
Train-specific behaviour (longitudinal analysis)	Train on earlier operational data from a single, specific train. Validate on later operational data from the same train to detect behavioural drift, degradation, or emerging patterns unique to that unit.
Train Area Behaviour (regional patterns)	Train on data from multiple trains operating within a defined area or a specific set of interconnected tracks. Validate on unseen time periods within the same area.
Train-Track Specific (localized interaction)	Train on data from specific trains operating on specific, well-defined track segments. Validate on new data from the same train-track combinations.

There are several ways to split the data, each with its own pros and cons. In this thesis, the **Train Area Behaviour** approach is used. This method focuses on a small, specific area with about five connected tracks. It makes the problem more manageable and helps the model learn typical patterns in that area. It also simplifies data collection and testing. Other methods, like "Train-Track Specific" or "Train-Specific", are tougher to implement properly, as they both suffer from a risk of training the model on a particular train that is already behaving improperly. Preparing the data for train-track specific purpose also requires thoughtful and time consuming preprocessing. A more detailed comparison of these methods and purposes is found in Section 5.2.3.

3.3.3 Model Architecture and Training Setup

When the data is preprocessed, an LSTM autoencoder model is constructed to perform reconstruction of the original sequence, as described in section 2.2.2.3. The model is designed with a specific number of LSTM layers and units (e.g., an encoder with layers of 64 and 32 units, and a corresponding decoder) to capture the temporal dependencies within the selected features over the defined sequence length.

A description of the model used can be seen in table 3.2. The model has two variables, *timesteps* and *features*. They represent how much time a reconstructed sequence covers and how many features should be used. With the sampling pre-

processing mentioned in subsection 3.3.2 each time step represents a sample taken every five seconds. This means that every minute of real data is represented by 12 time steps, and in order to reconstruct a sequence of five minutes 60 time steps are needed. The number of features represent how many properties the model should reconstruct. For instance, examining both speed and weight means this variable is equal to 2. Two base models with two layers of autoencoder layers of sizes *64, 32* and *128, 64* were implemented in *Python* with the use of the *keras* library for neural networks. The code for these networks can be found in appendix B.

Table 3.2: Architecture of the LSTM Autoencoder Model with input shape (*timesteps*, *features*).

Layer Type	**Parameters / Configuration**	**Output Shape**	**Section**
InputLayer	Input shape: (*timesteps*, *features*)	(None, *timesteps*, *features*)	Input
LSTM	Units: 128	(None, *timesteps*, 128)	Encoder
LSTM	Units: 64	(None, 64)	Encoder
RepeatVector	Repeats: *timesteps*	(None, *timesteps*, 64)	Bridge
LSTM	Units: 64	(None, *timesteps*, 64)	Decoder
LSTM	Units: 128	(None, *timesteps*, 128)	Decoder
TimeDistributed	Units: *features*, Activation: linear (default)	(None, *timesteps*, *features*)	Decoder

Note: The 'None' in the output shape represents the batch size.

After the model architecture is decided it is then compiled and trained. The Adam optimizer and Mean Absolute Error (MAE) are used as the loss function, more about this is described in subsection 3.3.4.The model learns by minimizing the reconstruction error on the training sequences. An early stopping mechanism is used, monitoring the validation loss to prevent overfitting and restore the model weights from the epoch that achieved the best performance on the validation set.

Once training is complete, the model's ability to reconstruct unseen data is evaluated using the validation set. The trained autoencoder processes each sequence in the validation set, generating a reconstructed version. The MAE is then calculated for each validation sequence by comparing it to its reconstruction. The premise is that the model, having learned normal patterns, will reconstruct normal sequences with low error, while anomalous sequences, which deviate from these learned patterns, will result in higher reconstruction errors.

The model can be tuned and tested with different parameters, features and sequence lengths to best suit the desired purpose. Generally, longer sequences are harder to reconstruct and will provide larger errors, but will grant more insight due to longer and more complex patterns. Analysing more features at a time, such as both weight

and speed, is preferable due to the extended context of what is happening to the train, but also makes the model struggle more with reconstruction as it has to reconstruct two sequences at the same time. In this case speed is very linear while weight is a discrete signal due to on- and offloading. A balance between these factors has to be struck in order to make the model perform optimally.

The goal is to create a model that can give as much context as possible while still being able to reconstruct the pattern of the original sequence to an acceptable degree. If given too complicated of a task the reconstruction will regress to a flat curve stuck at the mean value. However if the task is simpler but is lacking enough context the model may reconstruct the sequence very well, but there is not enough surrounding information available to learn any meaningful patterns of common behaviour.

In section 4.3.1 the results of using different sizes of context window can be observed. For that particular configuration, using two features (speed and weight), LTSM layers of sizes 128 and 64 were used. The activation function was tanh and a batch size of 64 was used for training. The model was trained on a set of eight cargo trains, and validated on two. These trains are all of the same type and operate in the same region.

3.3.4 Evaluation Metrics

To evaluate the performance of the model, we use the **mean absolute error** (MAE) as the primary metric for reconstruction accuracy. The MAE is calculated between the model's output sequence and the corresponding ground truth sequence. Given a set of predicted sequences $\hat{X} = \{\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_n\}$ and their corresponding true sequences $X = \{x_1, x_2, \ldots, x_n\}$, the MAE for each individual sequence is computed as:

$$\text{MAE}(x_i, \hat{x}_i) = \frac{1}{T \cdot F} \sum_{t=1}^{T} \sum_{f=1}^{F} |x_{i,t,f} - \hat{x}_{i,t,f}|$$

where:

- T is the number of time steps per sequence,
- F is the number of features (e.g., speed, weight),
- $x_{i,t,f}$ and $\hat{x}_{i,t,f}$ are the true and predicted values for feature f at time t in sequence i.

This formulation extends the conventional MAE, typically applied to scalar values, to multi-dimensional sequences. By averaging the absolute errors across both time steps and features, we obtain a comprehensive measure of reconstruction accuracy per sequence.

The rationale for using MAE over other metrics, such as Mean Squared Error (MSE), is its robustness to outliers and its linear nature which treats all errors equally, avoid-

ing the disproportionate penalization of larger deviations [14]. This characteristic makes MAE well suited for anomaly detection, where the objective is to identify deviations from normal patterns without allowing extreme outliers to dominate the analysis.

The result is a single MAE score per sequence, reflecting how well the sequence was reconstructed by the model. A high MAE score indicates that the model was unable to accurately reconstruct the sequence, and thus the sequence may be considered anomalous. However, as discussed in subsection 5.2.1, a low reconstruction accuracy can depend on many different factors and does not necessarily mean that something is wrong.

4

Results

After implementing and testing the systems as described above, the outcomes are discussed in this chapter.

4.1 Data analysis and formatting

The data is now structured in a much clearer and more organized format, as shown in the example CSV file in Figure 4.1, compared to the original unstructured log file shown in Listing 2.1. This improved format significantly enhances readability and usability. Users can easily filter and interpret the information directly within spreadsheet software such as Excel. Additionally, engineers can efficiently process and analyse the data using scripts or programming tools, enabling advanced filtering and automated data analysis.

MessageType	TimeStamp	SequenceNo	ObjectIdentity	Status
Indication	2025-03-10 20:40	126	4400TO410	Left
Indication	2025-03-10 20:40	127	4400TO411	Left
Indication	2025-03-10 20:40	128	4410PLI441	Right
Indication	2025-03-10 20:40	129	4420PLI420	Right
Indication	2025-03-10 20:40	130	4420PLI421	Right
Indication	2025-03-10 20:40	131	4600TO512	Left
Indication	2025-03-10 20:40	132	4600TO513	Left
Indication	2025-03-10 20:40	133	4600TO514	Left
Indication	2025-03-10 20:40	134	4100DR016	Off
Indication	2025-03-10 20:40	135	4100DR017	Off
Indication	2025-03-10 20:40	136	4100DR018	Off
Indication	2025-03-10 20:40	137	4100DR103	Off
Indication	2025-03-10 20:40	138	4100DR107	Off
Indication	2025-03-10 20:40	139	4100DR109	Off
Indication	2025-03-10 20:40	140	4100DR116	Off
Indication	2025-03-10 20:40	141	4100DR117	Off
Indication	2025-03-10 20:40	142	4100DR118	Off

Figure 4.1: Better formatting of log files. Comma-separated value (.csv) file opened in Microsoft Excel.

Given this data parsing we can now automatically generate reports. This is done very primitively in figure 4.2 and could be improved based on specifications and

LLM implementation, and combined with anomaly detection stats. This aligns well with the use case described in section 5.2.4.1.

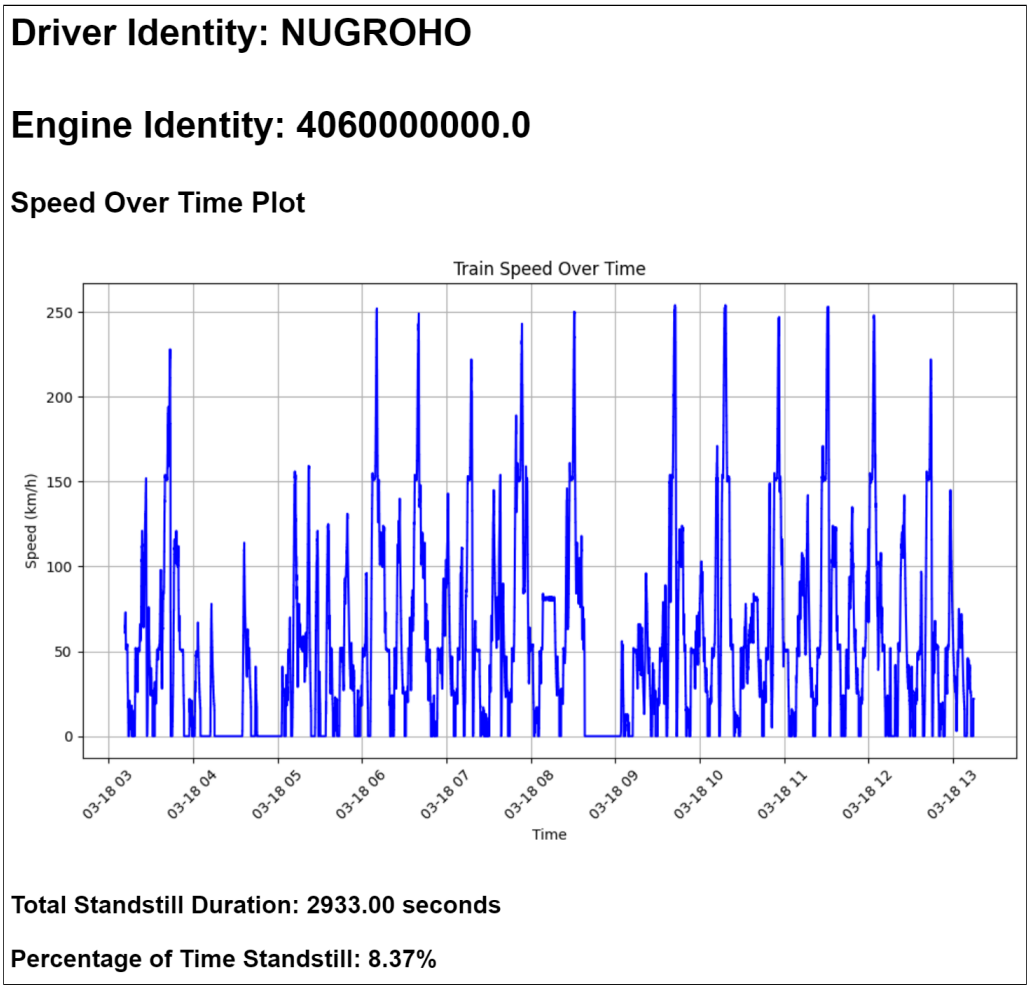

Figure 4.2: Basic markdown report automatically generated during parsing.

4.2 Log Replayer Application

The development of the replayer was started, as both a back end system and a rudimentary front end was being worked on. There were however some difficulties with integrating the program with the internal systems already in place, meaning the work was slow and highly speculative as adequate tests could not be performed. Because of this, development of the replayer was deprioritised in favour of development and experimentation of the anomaly detection. As the project never reached far enough to produce anything of value there are no relevant results to present, but there is hope that the work done will come in handy in the future for a complete development of the concept.

4.3 Anomaly detection

The results of the anomaly detection experiments demonstrate that different methodological choices and input configurations lead to observably different outcomes. This section details the model's performance, first focusing on reconstruction tasks using multiple features (speed and weight) across varying prediction window lengths.

4.3.1 Reconstruction using Speed and Weight

This subsection evaluates the autoencoder's ability to reconstruct input sequences composed of both speed and weight features. A key aspect of this evaluation is understanding how the length of the prediction window impacts reconstruction accuracy, as measured by the Average Mean Absolute Error (MAE). While shorter windows (e.g., 2 minutes) simplify the reconstruction task, potentially leading to lower MAE, they might not capture sufficient temporal context for the model to learn complex, real-world operational patterns. Longer windows provide richer context but increases the difficulty of accurate reconstruction.

Table 4.1 presents the MAE scores for prediction windows ranging from 2 to 12 minutes when using speed and weight features. A general trend of increasing MAE is observed as the window length increases. Notably, there is a substantial rise in MAE for the 12-minute window. At this point the task becomes to complicated, and the model regresses to the average.

Table 4.1: Average Mean Absolute Error (MAE) for different prediction window sizes using Speed and Weight features. (1 minute = 12 time steps)

Prediction Window (Minutes)	Equivalent Time Steps	Average MAE
2	24	0.0111
4	48	0.0129
6	72	0.0215
8	96	0.0224
10	120	0.0287
12	144	0.2740

To provide a visual understanding of the reconstruction performance, Figures 4.3 through 4.8 display examples of the model's last-step predictions for both speed and weight features compared to the actual values, across the different window lengths. Normally a visual reconstruction can only be done with the length of the used time steps. However, in order to visually compare the different configurations, several windows are combined into one plot, using the last time step from each prediction.

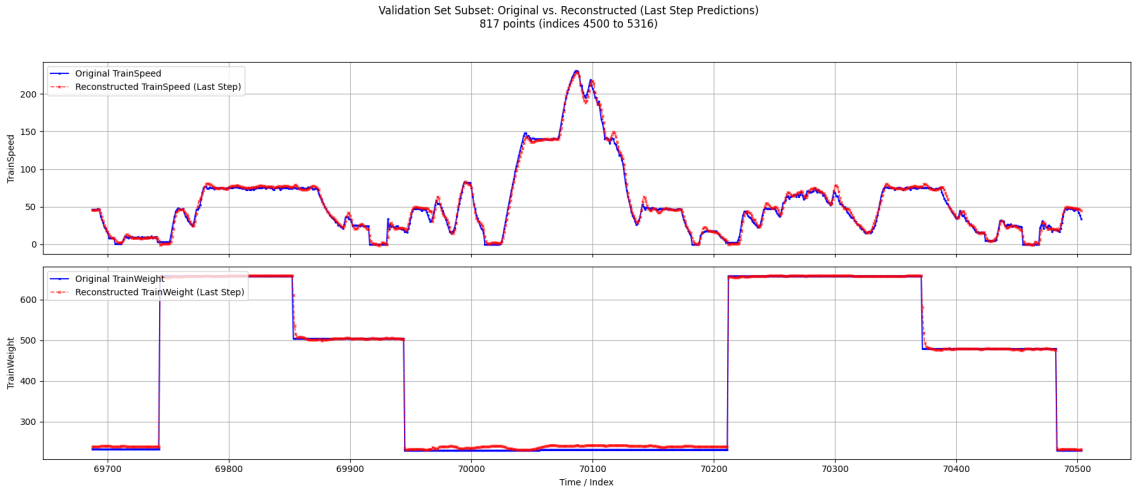

Figure 4.3: Last step predictions for speed and weight using a 2-minute window.

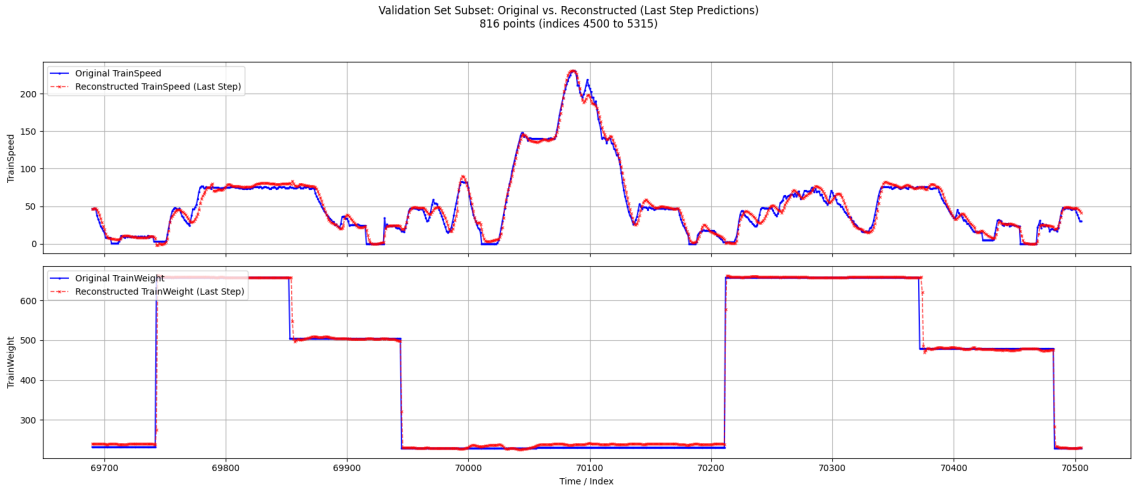

Figure 4.4: Last step predictions for speed and weight using a 4-minute window.

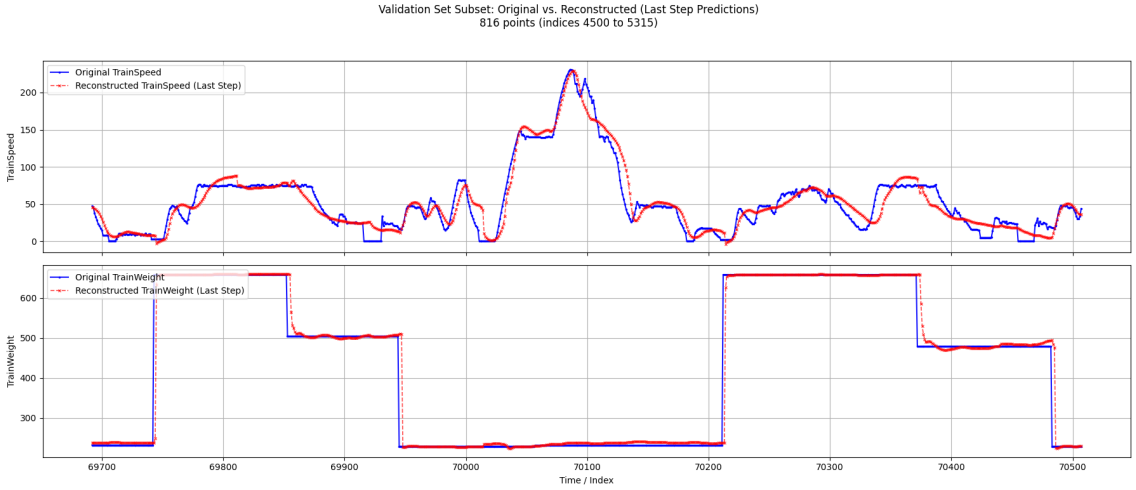

Figure 4.5: Last step predictions for speed and weight using a 6-minute window.

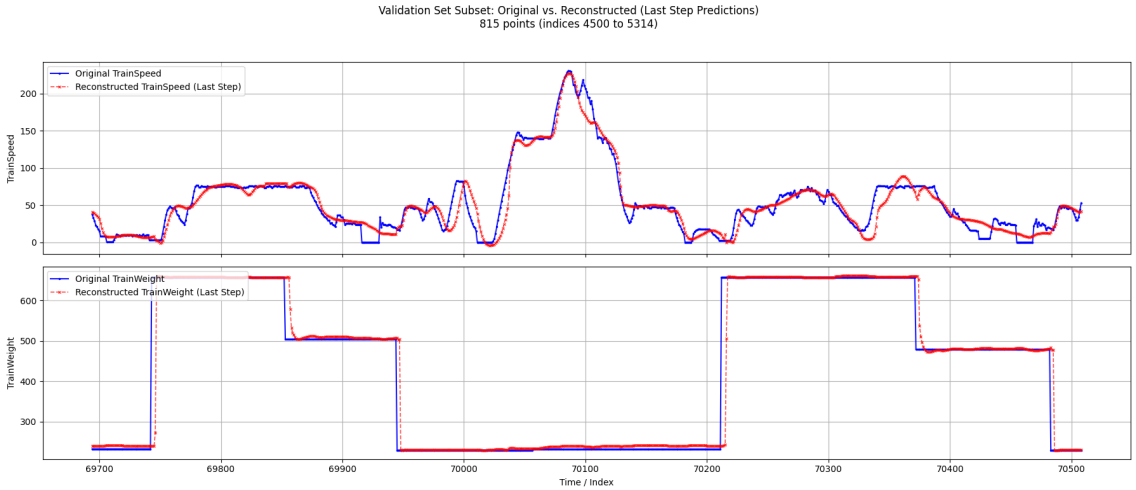

Figure 4.6: Last step predictions for speed and weight using an 8-minute window.

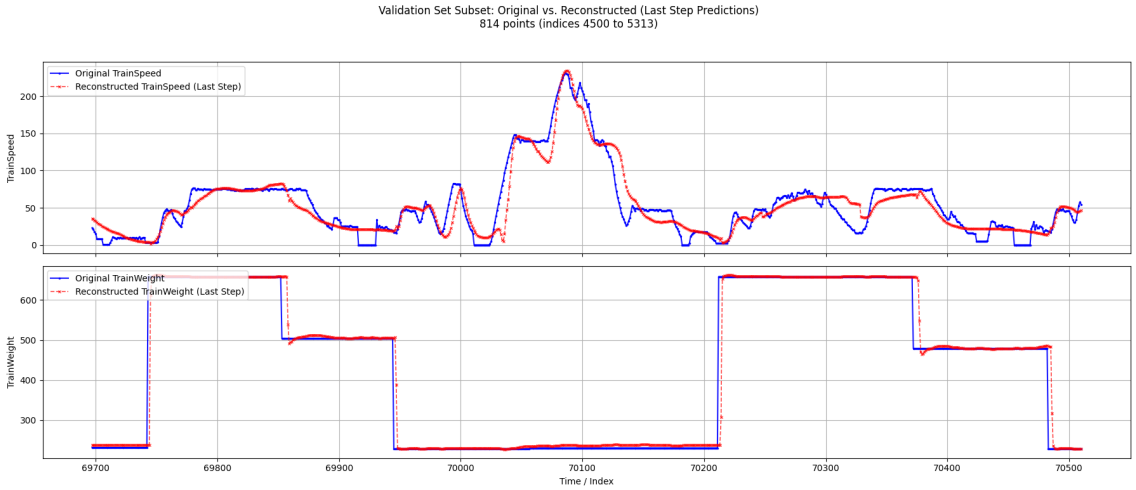

Figure 4.7: Last step predictions for speed and weight using a 10-minute window.

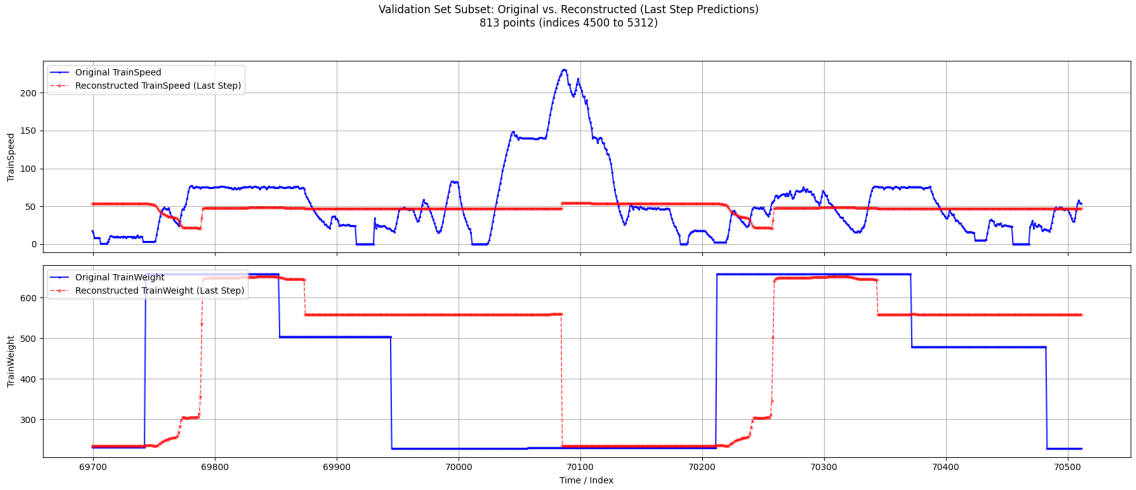

Figure 4.8: Last step predictions for speed and weight using a 12-minute window.

The figures clearly show the reconstruction deviating more and more from the original data as the context window grows. In figure 4.8, it is apparent why this particular model resulted in a much higher MAE score. It is clear, especially for the speed metric, that the model regresses towards the mean, which is common for such models when the task gets too complicated. A slight right shift can be observed in most models, especially when observing the weight in figure 4.8. This is simply due to the reconstruction plots using the last time step for this large plot.

4.3.2 Examples of High Reconstruction Error Sequences

In order to figure out what the model could potentially flag as anomalous behaviour, it is a good idea to analyse some specific sequences of high reconstruction error. For the following figures in this section, the scale of TrainSpeed is $\frac{1}{10} * speed$ in km/h.

Figure 4.9 and 4.10 both display attempts to reconstruct the same data, but with slightly different configurations. In this sequence, the weight increases abruptly while the train is still in motion. Both models can be observed to construct a speed close to zero, before the weight of the train increases. The deviation from average MAE score in this sequence was significant, a visual representation of the amplitude of the deviation can be observed in figure 4.11.

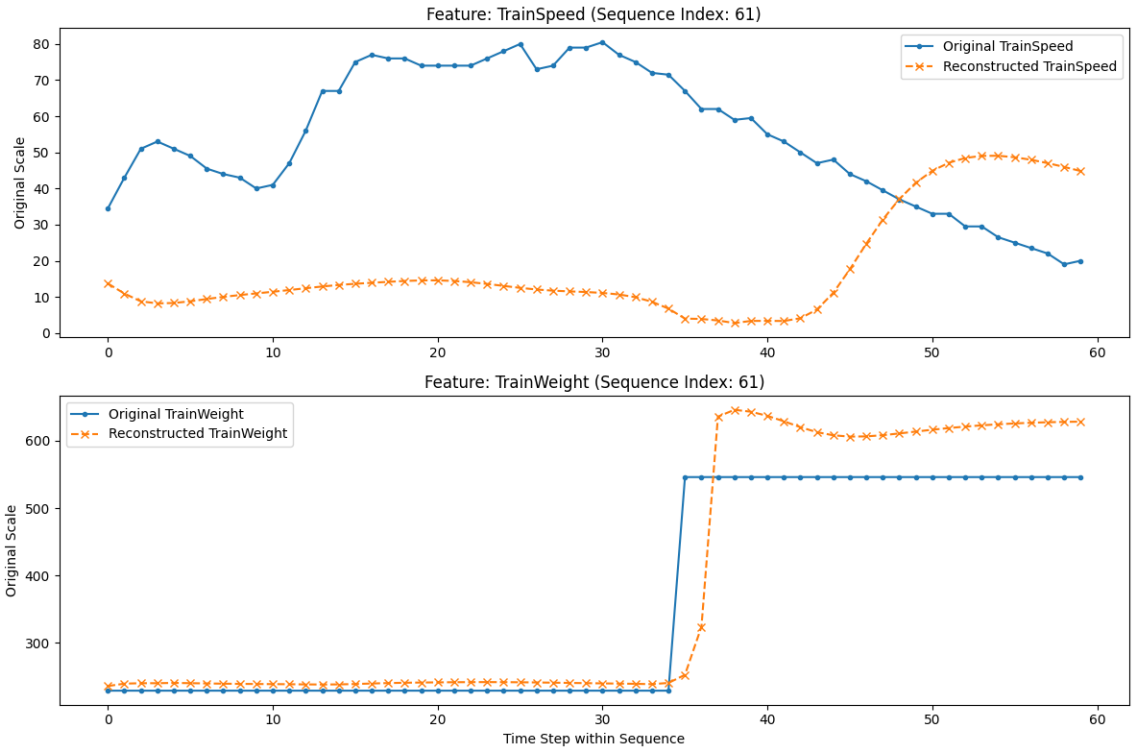

Figure 4.9: Model is unable to accurately reconstruct a sequence of the train being loaded while still moving. (5-minute window).

Figure 4.12 shows an example where the model struggled to reconstruct a sudden stop of the train, in this case without any changes to the weight. The model used in figure 4.13 did quite poorly when reconstructing the speed in this instance. Instead

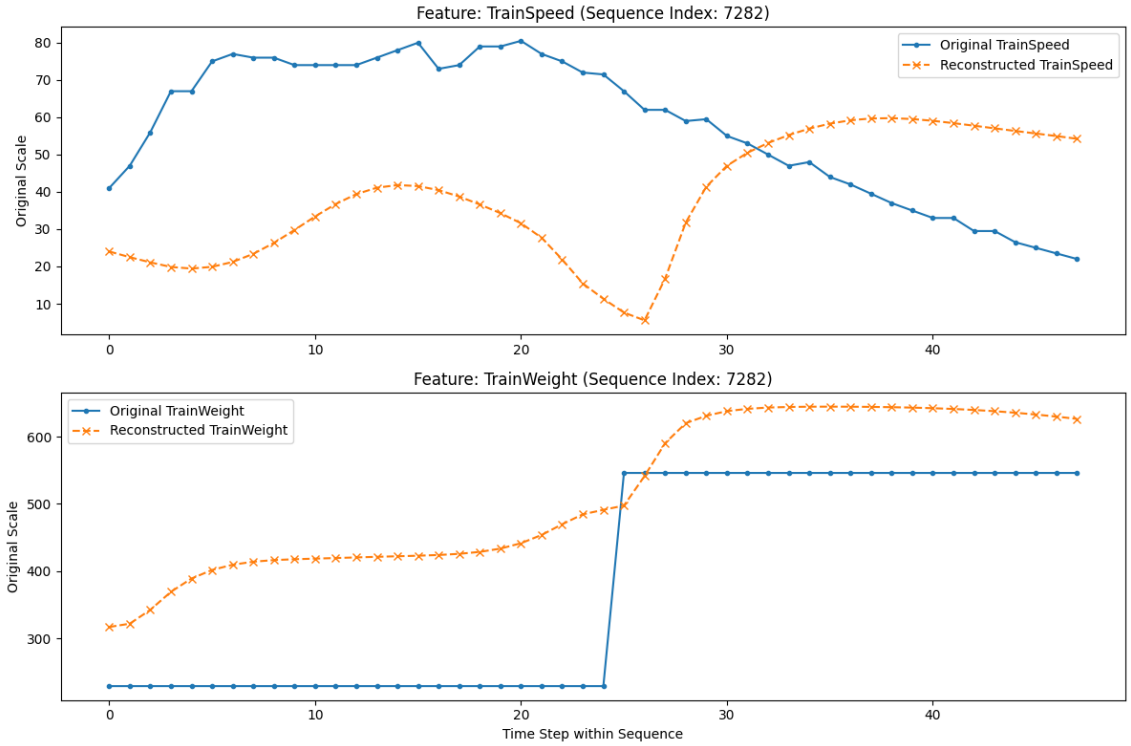

Figure 4.10: Model is unable to accurately reconstruct a sequence of the train being loaded while still moving. (4-minute window).

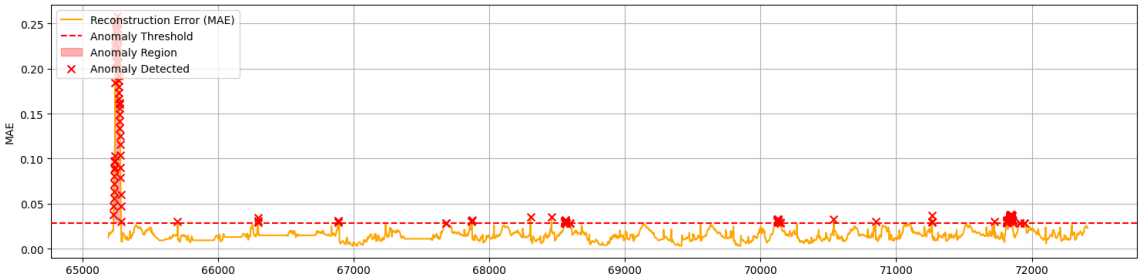

Figure 4.11: Mean Absolute Error (MAE) scores for a series of reconstructed sequences over time. The prominent peak corresponds to the high reconstruction error observed for the concurrent loading and motion event (Figures 4.9 and 4.10), illustrating its significant deviation from typical MAE scores. In this graph, a threshold of 98% is included.

of reconstruction in fluctuating pattern in speed, it regressed to reconstructing a standstill.

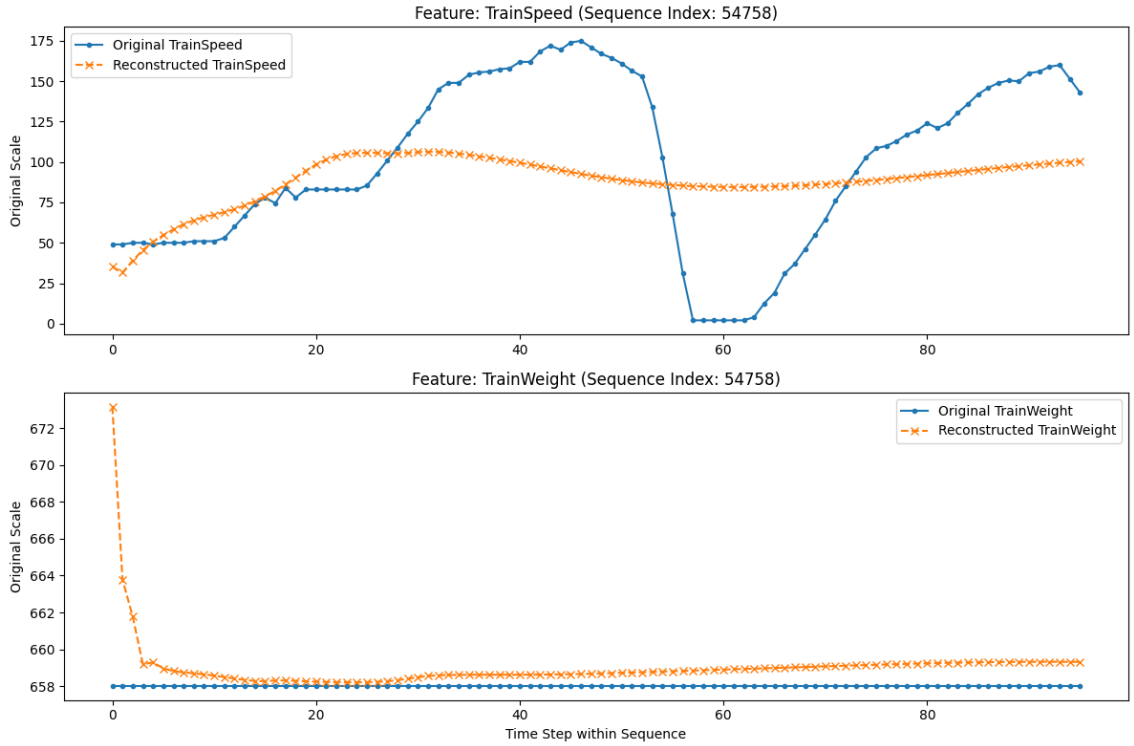

Figure 4.12: Train coming to a sudden stop, then getting up to speed again without any changes in weight. (8-minute window)

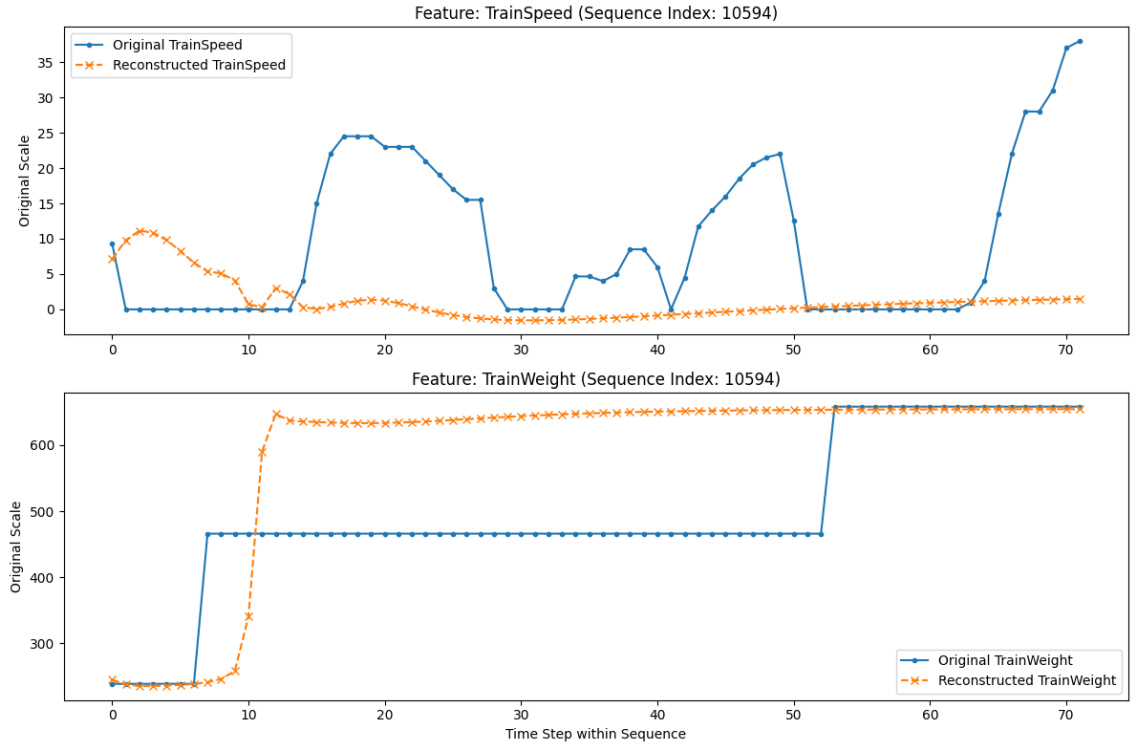

Figure 4.13: Long and complicated sequence, model struggles. (6-minute window)

5

Discussion

Building upon the methods developed and results presented in previous chapters, this chapter discusses their implications and broader context. It first examines the enhanced data analysis capabilities enabled by the new CSV log format. The focus then shifts to the anomaly detection approach, exploring the interpretation of its results, the trade-offs in model design concerning input complexity, and data splitting strategies. Lastly some potential practical use cases for the system are outlined.

5.1 Data Analysis

Simply by comparing the "raw" formatting of the logs as can be seen in listing 2.1, and the new CSV formatting in figure 4.1, the primary goal of facilitating data analysis of JRU logs can be seen as a success. From here on out, it will be easier for experts to investigate post malfunction or incident related data using the refined data storage format. In this format, it is possible to filter out irrelevant information and focus on the data that specifically needs to be investigated. An expert could simply use a spreadsheet software to only include columns where for example certain trains are included, and their relevant data points between certain time frames, instead of scrolling through endless hard-to-read log entries in a gigantic text file.

This can also be improved upon by implementing a user friendly program, where any non-expert could intuitively select a log file and filter the information via drop-down menus. In such a program, graphs such as the one seen in figure 4.2 could also be automatically generated to aid in visual understanding of the data. The user could select certain data points for certain objects within a certain time span to access specific desired information from the logs.

An additional possible improvement to this system could be an LLM implementation, where a user could ask an "AI assistant" of certain aspects of the log. For example, "how many times per day does a particular train pass a set of points at a speed of over 70 km/h?".

5.2 Anomaly detection

Anomaly detection can be very useful when attempting to detect system parameters that are out of the ordinary. However, as mentioned previously it is important that the results are carefully interpreted.

5.2.1 Interpretation of the results

The results presented in this report primarily represent how well a certain machine learning model can reconstruct the patterns of train behaviour over time. Interpreting these results, however, is a task that ultimately falls to human experts.

The first step of this implementation of anomaly detection is to train the model to be able to reconstruct specific patterns over time. The result is then presented, and the idea of anomaly detection is to find areas that deviate from the actual data. Areas that the model failed to reconstruct well could be interpreted as anomalies. However, the nuance here is of utmost importance. For every reconstructed sequence from the model, it is possible to mathematically quantify the deviation from the ground truth. But this is only the beginning, in order to draw any conclusion from the reconstruction error it must be considered why the model failed to reconstruct it well.

The next step is to interpret the results. The reconstruction errors can be interpreted in several ways, here three of them are highlighted as they fit well with the three proposed use cases in subsection 5.2.4.

Strategy A: Regularly identify the train with the highest anomaly score (e.g., weekly) for targeted inspection.

Strategy B: Set a high reconstruction error threshold that, when frequently exceeded, triggers expert review.

Strategy C: Select a fixed percentage of the most anomalous data for post-investigation to highlight irregular train behaviours.

Optimally, a failure from the model to reconstruct the sequence well means that the model has not seen, or has rarely seen, this type of behaviour from the train before. Examples of this could be: the train accelerating too fast under heavy load, sudden fluctuations in the measured weight, or the train reaching an unusually high speed given the load. These kinds of anomalies are things that are very useful to detect, as they could reasonably indicate that something is wrong with the train's behaviour or the sensors.

However, anomalies may also frequently occur in areas where the choice of model simply is not good at reconstructing that type of behaviour. This is definitely worth highlighting, especially during the multidimensional approach used in this thesis. The model is being asked to output weight which in the case of freight trains is a discrete signal due to the nature of loading and offloading. At the same time it is outputting a linear signal in terms of speed. Both of these have to be interpreted by the model, and it becomes a very complicated behaviour for a model that is best

suited for modelling linear behaviour. This however does not have to mean that this type of analysis is something to avoid, but it is worth noting when presented with results.

Anomalies can also be the cause of many other things that are not actual problems, such as sensor noise or rounding effects in the measurement systems. As mentioned in section 2.1, the logs are inconsistent and large gaps in time of certain metrics can lead to larger than usual jumps in metrics, even though the actual value was not anomalous. In some cases, maintenance or calibration procedures can temporarily change the weight or speed profile in a way the model has not seen before. Furthermore, certain seasonal or weather-related effects, like wheel slip in icy conditions, may cause outliers in speed measurements without representing a fault.

Because of this, the results of the model should not be treated as a black and white truth of weather something is wrong or not. On the contrary, anomalous behaviour should be expected from time. However repeated anomalous behaviour from a train, especially with very high reconstruction errors, should prompt a further look from an expert in the field.

5.2.1.1 Interpretation of High Reconstruction Error Sequences

As mentioned, data used for training this model did not contain explicitly labeled or predefined anomalous behaviors. Consequently, a high reconstruction error primarily indicates deviations from the patterns observed during training, which may or may not represent true anomalies requiring intervention. Following are some discussions regarding the results presented in section 4.3.2.

Loading While Moving (Figures 4.9 and 4.10)

These figures illustrate a scenario where the train's weight abruptly increases while it is still in motion, rather than at a standstill. This behaviour is particularly interesting because, in the observed system, the 'loaded' status is assigned to a train upon entering a designated loading zone rather than precisely at the moment cargo is added. This means the system registers a weight increase even before the train has necessarily started the actual loading process.

The model, likely trained on a majority of sequences where loading occurs at or near zero speed, struggles to reconstruct this 'loading while moving' pattern. Its prediction for TrainSpeed, as seen in the figures, tends towards zero, indicating its expectation for a stationary train during a weight change. This sequence, while flagged by the model, might not be a true operational anomaly but rather a normal, albeit less common, operational sequence within the system's specific configuration. If this specific 'loading zone' behavior was underrepresented or absent in the training data (perhaps because the model generalized from other areas where trains do stop first), then it will logically generate a high reconstruction error. Therefore, it is important to understandand the operational context and potentially tailoring data splitting strategies (e.g., using a 'Track-specific' approach or even 'Train-Track Specific' models for such zones) to account for regional operational nuances.

Sudden Stop (Figure 4.12)

Figure 4.12 depicts a sudden and significant deceleration of the train, from approximately 17 km/h down to a full stop within about 30 seconds, followed by re-acceleration. Notably, this event occurs without any corresponding change in the train's weight.

The model's poor reconstruction of the speed profile suggests it is unaccustomed to such abrupt stops in the absence of loading or offloading events. Most often, significant speed changes are correlated with cargo operations or planned stops at stations or signals. An emergency stop or an unforeseen track obstruction could be reasons for such rapid deceleration without weight changes. The question then becomes: is this an emergency stop, and therefore an anomaly requiring investigation, or is it a rare but legitimate operational maneuver (e.g., an unexpected signal stop, or a temporary halt for track maintenance)? The model's struggle to reconstruct it highlights its rarity in the training data, regardless of its true 'normal' or 'anomalous' status. Further expert review would be necessary to classify this specific event and determine if it points to a problem with train operation, sensor data, or an infrequent but valid operational procedure.

Long and Complicated Sequence (Figure 4.13)

Figure 4.13 presents a particularly challenging sequence for the model, characterized by fluctuating speed and multiple discrete changes in weight. This kind of behaviour is often associated with *shunting* operations within a train yard, where individual railcars or sets of railcars (*rakes*) are added to or removed from the train. Unlike typical mining train operations where weight in Alstom's system largely alternates between a 'loaded' and 'unloaded' state, shunting can lead to multiple weight changes within a single sequence. This is a relatively rare occurrence compared to the frequent loading and offloading cycles of mining trains that likely dominate the training data.

The model struggles to reconstruct this complexity for several reasons: the rarity of such multi-stage weight changes in its training data, the rapid and erratic fluctuations in speed, and the overall deviation from the smoother, more predictable profiles it has learned. Its tendency to 'regress to standstill' during the sudden stop indicates a lack of confidence in predicting such dynamic and non-linear speed patterns. This example strongly demonstrates how events that are genuinely uncommon but potentially legitimate operational procedures can lead to high reconstruction errors. While not necessarily indicative of a malfunction, these sequences are prime candidates for further review by human experts to understand the underlying operational context and ensure they represent safe and intended behaviour. If such patterns are indeed normal for specific operational zones, they would ideally need to be explicitly represented in the training data for models designated for those areas.

5.2.2 Balancing Input Complexity and Reconstruction

A critical consideration in designing sequence-based anomaly detection systems is the trade-off between the richness of the input context and the model's ability to faithfully reconstruct that input. The "context" here refers to both the temporal length of the input sequence (the "window") and the number and nature of features (e.g. speed, weight) included at each time step.

The motivation for incorporating a longer context window and a greater number of features is easy to see. A longer temporal window allows the model to learn more complex, longer-term dependencies and patterns inherent in normal operational behavior. For instance, understanding the typical deceleration profile of a train might require observing its speed and load characteristics over several minutes leading up to a stopping event. Similarly, including multiple relevant features provides a more holistic view of the system's state, potentially enabling the detection of subtle anomalies that manifest as unusual correlations between different parameters (such as unexpectedly high speed for a given load and track gradient). This rich, multi-dimensional context, does in theory equip the model to establish a more nuanced baseline of normality.

However, increasing the length of the input sequences and the dimensionality of the feature space also significantly raises the complexity of the reconstruction task for the autoencoder. The model must learn to compress a larger, more intricate representation of the input into its latent space and then accurately decompress it back to the original sequence. As the information content of the input grows, the potential for reconstruction inaccuracies, even for entirely normal sequences, also increases.

As observed in chapter 4.3.1 and table 4.1: as the amount of time steps increases, the MAE score of the model decreases. For this particular case, a context window of 2 minutes is probably too short for the model to learn any real-world patterns, and a context window of 12 minutes is too long as it gets too complicated for the model to learn. The ideal context window length lies somewhere in between. A conceptual image of this trade-off can be observed in figure 5.1. The length of these windows will vary depending on factors such as model and data complexity, and needs to be tested in order to find the optimal window length.

5.2.3 Consequences of training data splits

Choosing how to split the data is crucial as it changes depending on the desired results of the anomaly detection. Recall table 3.1; here four different data splitting approaches are mentioned: *General train behaviour*, *Train-specific behaviour*, *Train Area Behaviour*, and *Train-Track Specific*. All of these methods have their own strengths and limitations.

In this thesis, the *Train Area Behaviour* approach was chosen. This decision was driven by the nature of the system, where cargo trains are autonomously controlled by a central computer that issues uniform commands regardless of individual train

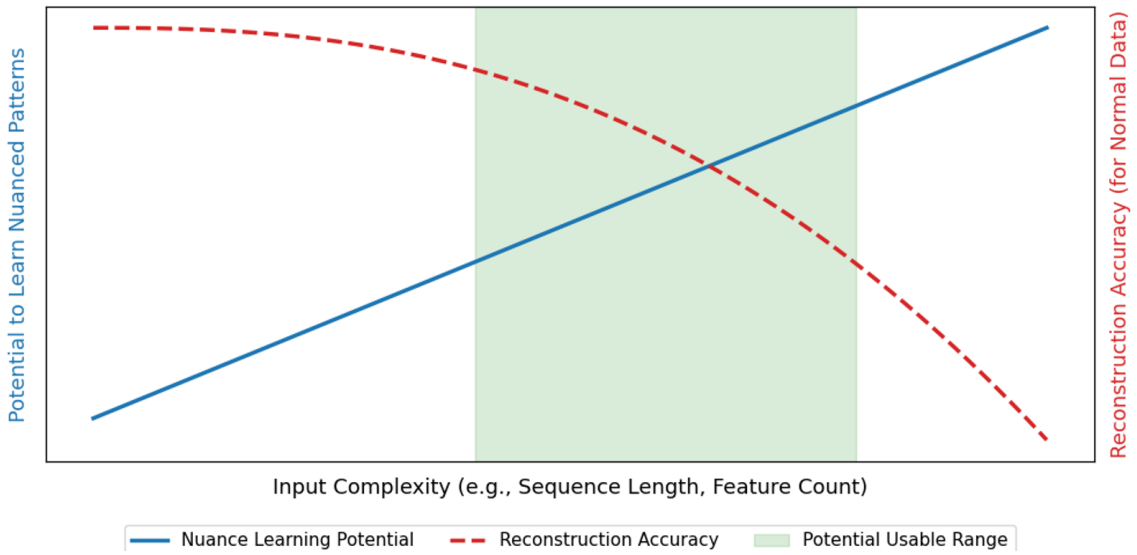

Figure 5.1: Trade-off between performance and reconstruction. Backed by table 4.1 but not based in actual data.

characteristics. This method also simplified data collection and made model training more feasible. By training on data from a specific area the model learns expected operational patterns for that region, allowing it to detect deviations. However this approach does not account for train-specific traits, and assumes the fleet is largely homogeneous. This can lead to overgeneralization and reduce the model's performance in diverse operational conditions.

An alternative is to create separate models for each train or even each train-track combination. This strategy avoids overgeneralization and can better capture the nuances of individual operations. This however makes for a challenge when selecting what data to train the model with. Since machine learning models typically perform better with more data, extensive data collection over longer periods, potentially weeks or months, would be needed before such a model could be deployed.

Many different factors need to be accounted for when training over such a long time and trying to be specific. If the goal is to model precise, healthy operation, factors such as weather variability, cargo differences, and mechanical wear must be considered. For example, if a train requires maintenance every 12 months, one must ask: should the model be trained on data from the beginning of that cycle, or data from just before service? While this method has the most potential by being precise and avoiding generalization, preparing the data requires careful consideration and, optimally, consultation with local experts on the area and type of train being used.

In summary, while the scenario explored in this thesis provides a practical and scalable solution, it risks losing accuracy in diverse fleets. More specific models offer better precision but require significantly more effort in data collection and validation. Which one is the best choice depends will depend on the circumstances. A visualized approximation of the suggested methods can be seen in figure 5.2, however keep in mind that it is possible to take other approaches to data splitting or to combine the

different methods for a more robust deployment.

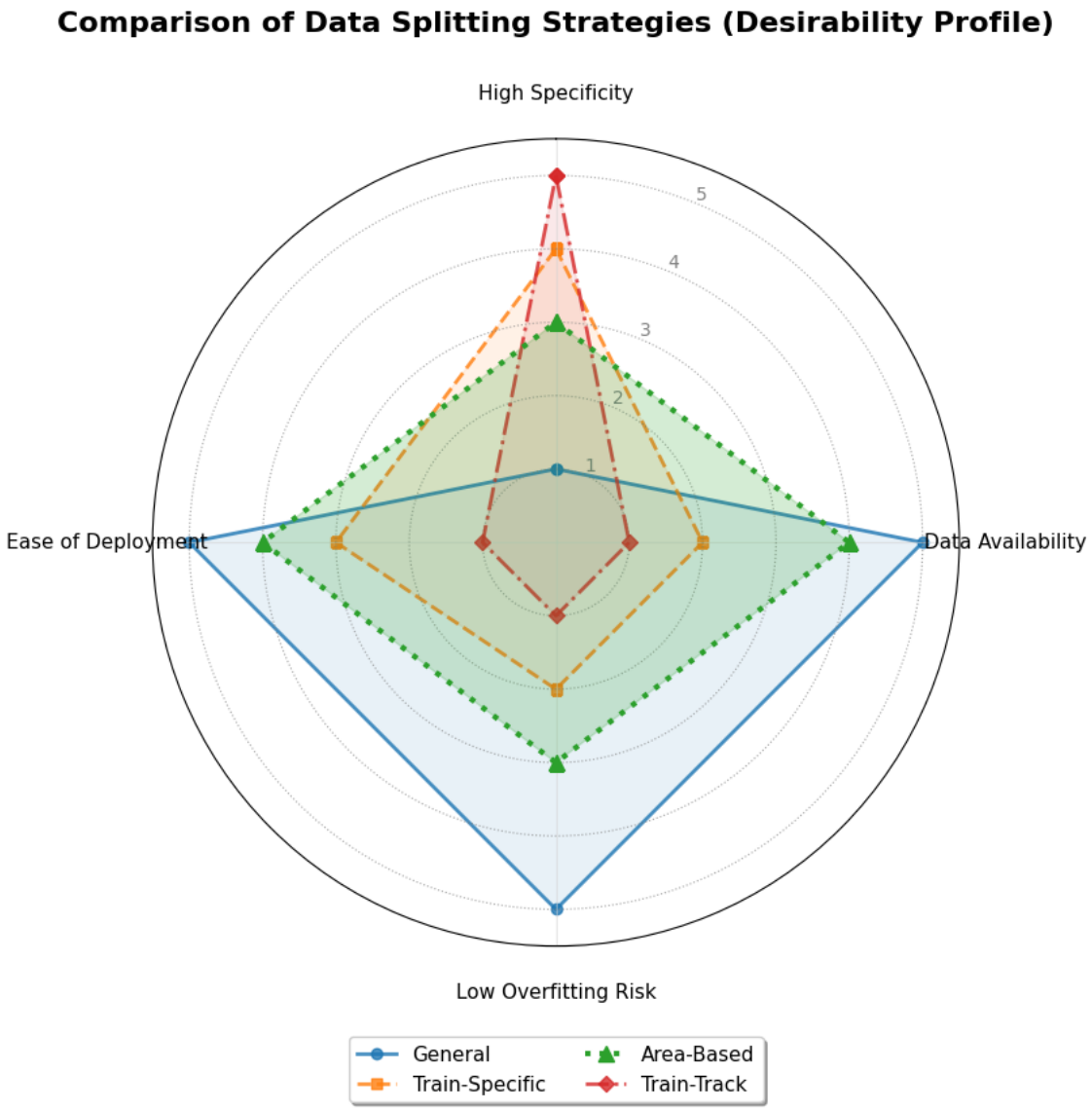

Figure 5.2: Desirability chart of different data splitting strategies. Not based in any actual metrics. For illustrative purposes.

5.2.4 Use cases

The anomaly detection system, based on a model's ability to reconstruct train behaviour patterns, offers valuable insight, however as previously mentioned it requires careful interpretation. The final practical application of the data depends heavily on how the system results are interpreted and reacted to. The interpretation strategies outlined in section 5.2.1 provide a framework for translating anomaly scores into actionable information. Below are three examples of use cases, note however that this is far from an exhaustive list.

5.2.4.1 Predictive Maintenance

The goal of predictive maintenance is generally to predict system degradation. This approach aligns best with the discussion in section 5.2.1. This can be employed with **Strategy A** where all trains are compared against each other. The system keeps track of each train's accumulated anomaly score during a set time period (e.g. a week), at the end of the period the train with the highest reconstruction error could be flagged for an extra inspection.

This approach could be combined with automatically generated reports which would be provided at the end of the period. A version of the anomaly scores would be included in the report, in which case local operators could be provided with additional data and context to trains that are behaving our of the norm, and make decisions on whether an extra inspection or intervention is of interest.

Dependent on the data-splitting strategy, many of the pitfall mentioned in section 5.2.3 are avoided, since anomaly scores are compared against each-other. Fluctuations dependent on weather or temporary track maintenance should be minor, as it would effect all trains rather equally.

5.2.4.2 Live Monitoring

By asking the model to analyse a log in real time as the log is generated, this application aims to detect potentially critical deviations as they happen. It employs **Strategy B**, where a high reconstruction error threshold is set, and frequent exceedances flags the train in question for inspection. The objective is to provide timely warnings about train behaviour that falls significantly outside the norm, potentially indicating an emerging or immediate problem requiring attention during operation. This alert generated by the system in the case of an extraordinary event would typically be routed to an operations or monitoring center, where an expert can review the train data while the train is still in service.

As an example, sustained high errors related to speed and vibration could indicate a developing mechanical issue. This could trigger an urgent inspection upon arrival, or be flagged as a non-issue dependent on the experts opinion. As mentioned in section 5.2.1, the indications of a system like this is not to be taken as ground truth of whether something is wrong or not, but an indication that something *may* be wrong.

5.2.4.3 Locating Problematic Events

The primary goal here is to identify and understand specific past incidents or periods where a train deviated from its established operational patterns. This is particularly useful for diagnostic purposes when a problem is suspected after the fact. **Strategy C** is used. After processing a historical log which is known or suspected to contain anomalous events, the system identifies the data points or time segments exhibiting the highest reconstruction errors. It then highlights the segments of the log with anomaly scores exceeding the determined threshold. An expert can then examine the

context of these flagged anomalies, correlating them with speed, load, track section, or other factors, to determine the nature and cause of the unusual behaviour.

6

Conclusion

In this thesis log files from Juridical Recording Units were analysed with a few goals in mind; making the logs easier to work with for Alstom employees, and analysing the data using a suitable machine learning model to see what additional information could be extracted from the data.

The first goal has generally been a success. The initial idea was to create an internal software application for the company in order to be able to replay the logs on the Centralised Train Control module, although this was largely cut short due to difficulties with integration with already established systems as well as an overall shift in priorities. The parsing that has been implemented however is a large benefit when working with the logs. Although not able to replay events as initially planned, parsing of the logs enable an analyst to easily look at the log data using a spreadsheet tool. In this way it is relatively simple to extract whatever information is desired from a given log, like only data from a given train or only data from trains heading to a specific area.

On the machine learning front, the specific model of interest investigated in this thesis was anomaly detection using an LSTM autoencoder. Although not yielding a resulting program that is ready to be implemented, the work has held more of an investigative role to understand if there is any valuable information available to be extracted using a model of this kind. While the model showed strength in low context windows, it failed to reconstruct the data at an appropriate level when the context window became too large. The results also indicate that there is a need for more specific models than the train region split used. For future endeavours, data splitting should be made in consultation with experts with awareness of track layouts and conditions. Overall the prospects in the area seem promising, although as always with machine learning it should be used in conjunction with experts trained in the area. As a result of the testing, a few use cases and examples of future work in the area have been proposed.

6.1 Future work

While the work conducted during this thesis has resulted in the data from the JRU logs being stored in a more readable and usable format, there is room for improvement. At the time of writing it can be used by a user to read the logs, however the current implementation is quite clunky and cumbersome to use. A reasonable next step would be to develop a simple GUI for the parsing process, making it more usable for a layman without having to read a complicated manual first.

As mentioned above, while the machine learning implementation used in this thesis supports the claim that the area of research holds value it is not ready to see deployment in its current state. There are a few different reasonable ways to look further into this. As the data used during the construction of the model has been limited, a more thoroughly developed model with more diverse data to train on would be necessary in order to take the concept into production in any capacity. With that, one of the use cases discussed in section 5.2.4 could be developed and implemented into the train maintenance workflow.

Bibliography

[1] Enchord, "Anomaly detection in time series." Available at: https://encord.com/blog/anomaly-detection/, n.d. Accessed: May 22, 2025.

[2] GeeksforGeeks, "Anomaly detection in time series data." Available at: https://www.geeksforgeeks.org/anomaly-detection-in-time-series-data/, n.d. Accessed: May 22, 2025.

[3] Y. Zhao, T. Liu, F. Zhang, Y. Guo, J. Ma, J. Liu, and Z. Li, "Real-time anomaly detection for multivariate time series based on feature reconstruction," *Sensors*, vol. 24, no. 9, 2024.

[4] S. Kulm, "Fra to require stronger "black boxes" and more data collection to help train accident investigations," *U.S. Department of Transportation*, 2005. https://railroads.dot.gov/elibrary/fra-require-stronger-black-boxes-and-more-data-collection-help-train-accident.

[5] M. Jacyna, E. Szczepański, M. Izdebski, S. Jasiński, and M. Maciejewski, "Characteristics of event recorders in automatic train control systems," *Archives of Transport*, vol. 46, 2018. DOI: https://doi.org/10.5604/01.3001.0012.2103.

[6] I. Naish, "Use of collected data in accident investigations," *Railway Safety Conference*, 2013. https://international-railway-safety-council.com/wp-content/uploads/2017/09/naish-use-of-collected-data-in-accident-investigations.pdf.

[7] V.-H. Le and H. Zhang, "Log-based anomaly detection with deep learning: How far are we?," in *Proceedings of the 44th international conference on software engineering*, pp. 1356–1367, 2022.

[8] C.-H. Lee, H.-S. Shin, A. Tsourdos, and Z. Skaf, "Anomaly detection of aircraft engine in fdr (flight data recorder) data," in *IET 3rd International Conference on Intelligent Signal Processing (ISP 2017)*, p. 3, IET, 2017.

[9] S. Das, S. Sarkar, A. Ray, A. Srivastava, and D. L. Simon, "Anomaly detection in flight recorder data: A dynamic data-driven approach," in *2013 American Control Conference*, pp. 2668–2673, 2013.

[10] M. Gupta, J. Gao, C. C. Aggarwal, and J. Han, "Outlier detection for temporal data: A survey," *IEEE Transactions on Knowledge and data Engineering*, vol. 26, no. 9, pp. 2250–2267, 2013.

[11] J. E. D. A. Filho, "A review of neural networks for anomaly detection," *IEEE Access*, 2022.

[12] R. M. Schmidt, "Recurrent neural networks (rnns): A gentle introduction and overview," *arXiv preprint arXiv:1912.05911*, 2019.

[13] O. I. Provotar, Y. M. Linder, and M. M. Veres, "Unsupervised anomaly detection in time series using lstm-based autoencoders," in *2019 IEEE International Conference on Advanced Trends in Information Theory (ATIT)*, pp. 513–517, 2019.

[14] D. Burch, "Mean absolute error in machine learning: What you need to know," *Arize AI Blog*, 2023.

A

Supplementary Figures

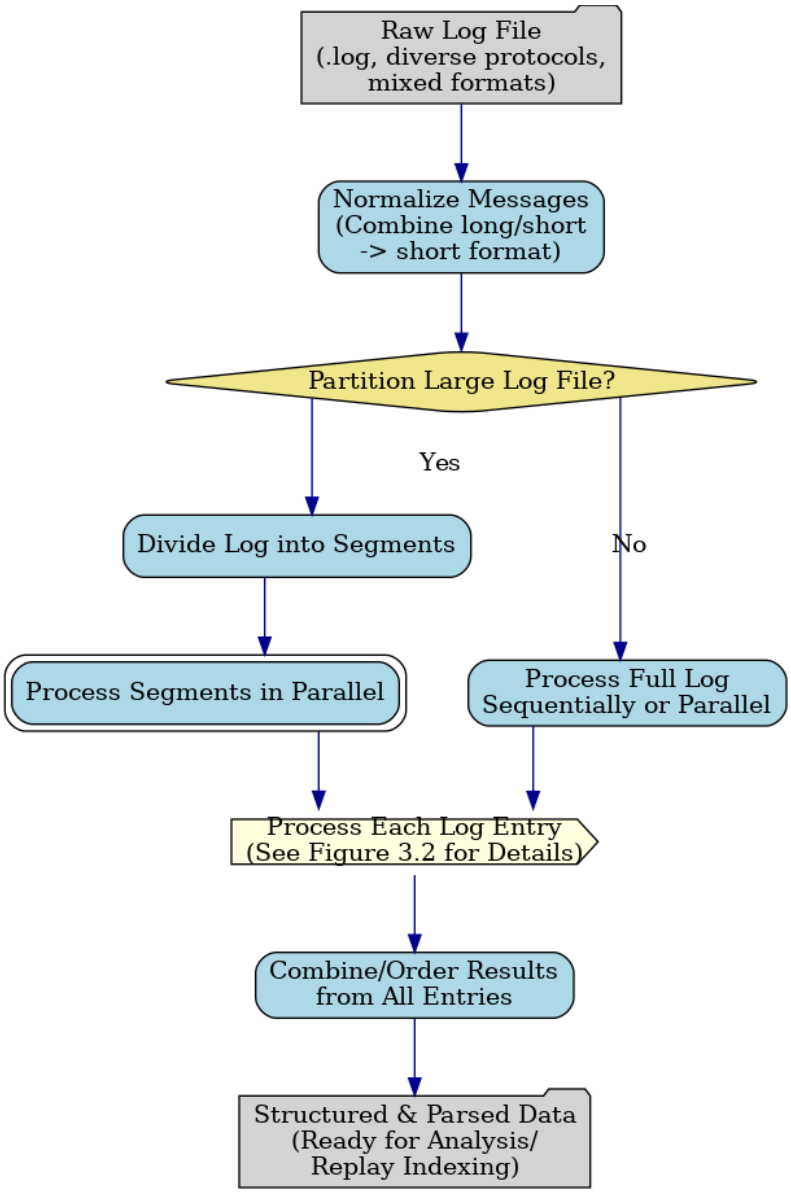

Figure A.1: Higher level parsing description.

B

Code Snippets

B.1 LSTM Model Code

```python
import keras

def lstm_64(sequence_length, num_features):
    model = keras.Sequential(
        [
            keras.layers.Input(shape=(sequence_length, num_features)),
            # Encoder
            keras.layers.LSTM(64, return_sequences=True),
            keras.layers.LSTM(32, return_sequences=False),
            keras.layers.RepeatVector(sequence_length),
            # Decoder
            keras.layers.LSTM(32, return_sequences=True),
            keras.layers.LSTM(64, return_sequences=True),
            keras.layers.TimeDistributed(keras.layers.Dense(num_features))
        ]
    )
    return model

def lstm_128(sequence_length, num_features):
    model = keras.Sequential(
        [
            keras.layers.Input(shape=(sequence_length, num_features)),
            # Encoder
            keras.layers.LSTM(128, return_sequences=True),
            keras.layers.LSTM(64, return_sequences=False),
            keras.layers.RepeatVector(sequence_length),
            # Decoder
            keras.layers.LSTM(64, return_sequences=True),
            keras.layers.LSTM(128, return_sequences=True),
            keras.layers.TimeDistributed(keras.layers.Dense(num_features))
        ]
```

```
    )
    return model

```

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY